Le Manuel de Référence Canonique de Csound

Version 5.10

Barry Vercoe, MIT Media Lab
et. al.

Le Manuel de Référence Canonique de Csound: Version 5.10

par Barry Vercoe et et. al.

Table des matières

Préface ... xxix
- Préface du Manuel de Csound ... xxix
- Histoire du Manuel de Référence Canonique de Csound ... xxx
- Mentions de copyright ... xxxi
- Débuter avec Csound ... xxxiii
- Les nouveautés de Csound 5.10 ... xxxv

I. Vue d'Ensemble ... 1
- Introduction ... 4
- Développements Récents ... 5
  - Caractéristiques de Csound 5 ... 5
  - Caractéristiques de CsoundAC ... 6
- La commande Csound ... 8
  - Ordre de priorité ... 8
  - Description de la syntaxe de la commande ... 8
  - Csound command line ... 10
  - Options de Ligne de Commande (par Catégorie) ... 20
  - Variables d'Environnement de Csound ... 30
  - Format de Fichier Unifié pour les Orchestres et les Partitions ... 33
    - Description ... 33
    - Exemple ... 35
  - Fichier de Paramètres de Ligne de Commande (.csoundrc) ... 36
  - Prétraitement du Fichier Partition ... 36
    - La Fonction Extract ... 36
    - Prétraitement Indépendant avec Scsort ... 36
- Utiliser Csound ... 38
  - Comment Csound5 fonctionne ... 38
    - Valeurs d'amplitude dans Csound ... 39
  - Audio en temps-réel ... 41
    - Entrées/Sorties en temps-réel sur Linux ... 42
    - Windows ... 47
    - Mac ... 48
  - Optimisation de la Latence Audio en E/S ... 48
- Configuration ... 50
- Syntaxe de l'Orchestre ... 51
  - Instructions de l'En-tête de l'Orchestre ... 52
  - Instructions de Bloc d'Instrument et d'Opcode ... 52
  - Instructions Ordinaires ... 53
  - Constantes et Variables ... 53
    - Initialisation de Variable ... 54
  - Expressions ... 55
  - Répertoires et Fichiers ... 55
  - Nomenclature ... 56
  - Macros ... 56
  - Instruments Nommés ... 57
  - Opcodes Définis par l'Utilisateur (UDO) ... 59
- La Partition Numérique Standard ... 61
  - Prétraitement des Partitions Standard ... 61
    - Carry ... 61
    - Tempo ... 62
    - Sort ... 62
  - Instructions de Partition ... 63
  - Symboles Next-P et Previous-P ... 63
  - Ramping ... 64

Macros de Partition ... 65
Partition dans Plusieurs Fichiers ... 67
Evaluation des Expressions ... 68
Chaînes de caractères dans les p-champs ... 69
Frontaux ... 71
CsoundAC ... 71
CsoundVST ... 73
TclCsound ... 76
L'interpréteur Tcl : cstclsh ... 76
Cswish: le shell de fenêtrage ... 76
Un serveur Csound ... 77
Un Environnement de Scripting ... 78
TclCsound comme encapsuleur de langage ... 79
Référence des Commandes de TclCsound ... 79
Construire Csound ... 82
Liens Csound ... 88
II. Vue d'Ensemble des Opcodes ... 89
Générateurs de Signal ... 93
Synthèse/Resynthèse Additive ... 93
Oscillateurs Elémentaires ... 93
Oscillateurs à Spectre Dynamique ... 93
Synthèse FM ... 94
Synthèse Granulaire ... 94
Synthèse Hyper Vectorielle ... 95
Générateurs Linéaires et Exponentiels ... 95
Générateurs d'Enveloppe ... 96
Modèles et Emulations ... 96
Phaseurs ... 97
Générateurs de Nombres Aléatoires (de Bruit) ... 98
Reproduction de Sons Echantillonnés ... 99
Soundfonts ... 99
Synthèse par Balayage ... 100
Accès aux Tables ... 102
Synthèse par Terrain d'Ondes ... 103
Modèles Physiques par Guide d'Onde ... 103
Entrée et Sortie de Signal ... 104
Entrées et Sorties Fichier ... 104
Entrée de Signal ... 104
Sortie de Signal ... 104
Bus Logiciel ... 105
Impression et Affichage ... 105
Requêtes sur les Fichiers Sons ... 105
Modificateurs de Signal ... 107
Modificateurs d'Amplitude et Traitement des Dynamiques ... 107
Convolution et Morphing ... 107
Retard ... 107
Panning et Spatialisation ... 108
Réverbération ... 110
Opérateurs du Niveau Echantillon ... 110
Limiteurs de Signal ... 111
Effets Spéciaux ... 111
Filtres Standard ... 111
Filtres Spécialisés ... 113
Guides d'Onde ... 113
Distorsion Non-Linéaire et Distorsion de Phase ... 113
Contrôle d'Instrument ... 115
Contrôle d'Horloge ... 115
Valeurs Conditionnelles ... 115

- Instructions de Contrôle de Durée ... 115
- Contrôleurs Graphiques FLTK et GUI ... 115
  - Conteneurs FLTK ... 118
  - Valuateurs FLTK ... 118
  - Autres Contrôleurs Graphiques FLTK ... 119
  - Modifier l'Apparence des Contrôleurs Graphiques FLTK ... 120
  - Opcodes Généraux relatifs aux Contrôleurs Graphiques FLTK ... 120
- Appel d'Instrument ... 121
- Contrôle Séquentiel d'un Programme ... 121
- Controle de l'Exécution en Temps Réel ... 122
- Initialisation et Réinitialisation ... 122
- Détection et Contrôle ... 123
- Piles ... 124
- Contrôle de sous-instrument ... 125
- Lecture du Temps ... 125

Contrôle des Tables de Fonction ... 126
- Requêtes sur une Table ... 126
- Opérations de Lecture/Ecriture de Table ... 126
- Lecture de Table avec Sélection Dynamique ... 127

Opérations Mathématiques ... 128
- Conversion d'Amplitude ... 128
- Opérations Arithmétiques et Logiques ... 128
- Comparateurs et Accumulateurs ... 128
- Fonctions Mathématiques ... 129
- Opcodes Equivalents à des Fonctions ... 129
- Fonctions aléatoires ... 130
- Fonctions Trigonométriques ... 130
- Linear Algebra Opcodes ... 131

Conversion des Hauteurs ... 141
- Fonctions ... 141
- Opcodes de Hauteurs ... 141

Support MIDI en Temps-Réel ... 142
- Clavier Virtuel MIDI ... 143
- Entrée MIDI ... 146
- Sortie de Message MIDI ... 146
- Entrée et Sortie Génériques ... 147
- Convertisseurs ... 147
- Extension d'Evènements ... 147
- Sortie de Note-on/Note-off ... 147
- Opcodes pour l'Interopérabilité MIDI/Partition ... 148
- Messages System Realtime ... 149
- Banques de Réglettes ... 149

Traitement Spectral ... 151
- Resynthèse par Tranformée de Fourier à Court-Terme (STFT) ... 151
- Resynthèse par Codage Prédictif Linéaire (LPC) ... 152
- Traitement Spectral Non-standard ... 152
- Outils pour le Traitement Spectral en Temps Réel (opcodes pvs) ... 152
- Traitement Spectral avec ATS ... 154
- Opcodes Loris ... 154

Chaînes de Caractères ... 159
- Opcodes de Manipulation de Chaîne ... 160
- Opcodes de Conversion de Chaîne ... 160

Opcodes Vectoriels ... 162
- Opérateurs de Tableaux de Vecteurs ... 162
- Opérations Entre un Signal Vectoriel et un Signal Scalaire ... 162
- Opérations Entre deux Signaux Vectoriels ... 163
- Générateurs Vectoriels d'Enveloppe ... 163
- Limitation et Enroulement des Signaux Vectoriels de Contrôle ... 164

Chemins de Retard Vectoriel au Taux de Contrôle ... 164
Générateurs de Signal Aléatoire Vectoriel ... 164
Système de Patch Zak ... 166
Accueil de Plugin ... 167
DSSI et LADSPA pour Csound ... 167
VST pour Csound ... 167
OSC et Réseau ... 169
OSC ... 169
Réseau ... 169
Opcodes pour le Traitement à Distance ... 169
Opcodes Mixer ... 170
Opcodes Python ... 171
Introduction ... 171
Syntaxe de l'Orchestre ... 171
Opcodes pour le traitement d'image ... 173
Opcodes divers ... 174
III. Référence ... 175
Opcodes et Opérateurs de l'Orchestre ... 197
!= ... 198
#define ... 200
#include ... 204
#undef ... 206
#ifdef ... 207
#ifndef ... 209
$NOM ... 210
% ... 213
&& ... 215
> ... 216
>= ... 218
< ... 220
<= ... 222
* ... 224
+ ... 226
- ... 228
/ ... 230
= ... 232
== ... 234
^ ... 236
|| ... 238
0dbfs ... 239
<< ... 242
>> ... 244
& ... 245
| ... 246
¬ ... 247
... 248
a ... 249
abetarand ... 251
abexprnd ... 252
abs ... 253
acauchy ... 255
active ... 256
adsr ... 260
adsyn ... 263
adsynt ... 265
adsynt2 ... 268
aexprand ... 270
aftouch ... 271

agauss ... 273
agogobel ... 274
alinrand ... 275
alpass ... 276
ampdb ... 279
ampdbfs ... 281
ampmidi ... 283
apcauchy ... 285
apoisson ... 286
apow ... 287
areson ... 288
aresonk ... 290
atone ... 291
atonek ... 293
atonex ... 294
atrirand ... 295
ATSadd ... 296
ATSaddnz ... 298
ATSbufread ... 300
ATScross ... 302
ATSinfo ... 304
ATSinterpread ... 306
ATSread ... 307
ATSreadnz ... 309
ATSpartialtap ... 311
ATSsinnoi ... 312
aunirand ... 314
aweibull ... 315
babo ... 316
balance ... 320
bamboo ... 322
barmodel ... 324
bbcutm ... 326
bbcuts ... 331
betarand ... 333
bexprnd ... 335
bformenc ... 337
bformenc1 ... 339
bformdec ... 341
bformdec1 ... 343
binit ... 345
biquad ... 346
biquada ... 350
birnd ... 351
bqrez ... 353
butbp ... 355
butbr ... 356
buthp ... 357
butlp ... 358
butterbp ... 359
butterbr ... 361
butterhp ... 363
butterlp ... 365
button ... 367
buzz ... 368
cabasa ... 370
cauchy ... 372
ceil ... 374

cent 375
cggoto 377
chanctrl 379
changed 380
chani 382
chano 383
chebyshevpoly 384
checkbox 387
chn 389
chnclear 391
chnexport 392
chnget 394
chnmix 396
chnparams 397
chnset 398
chuap 400
cigoto 403
ckgoto 405
clear 407
clfilt 408
clip 411
clock 414
clockoff 415
clockon 416
cngoto 417
comb 419
compress 422
control 424
convle 425
convolve 426
cos 429
cosh 431
cosinv 433
cps2pch 435
cpsmidi 439
cpsmidib 441
cpsmidinn 443
cpsoct 446
cpspch 449
cpstmid 452
cpstun 455
cpstuni 458
cpsxpch 461
cpuprc 465
cross2 467
crunch 469
ctrl14 471
ctrl21 473
ctrl7 475
ctrlinit 477
cuserrnd 478
dam 480
date 483
dates 485
db 487
dbamp 489
dbfsamp 491
dcblock 493

dcblock2 495
dconv 496
delay 498
delay1 500
delayk 501
delayr 503
delayw 504
deltap 506
deltap3 508
deltapi 510
deltapn 512
deltapx 514
deltapxw 516
denorm 518
diff 519
diskgrain 521
diskin 524
diskin2 527
dispfft 530
display 532
distort 534
distort1 536
divz 538
downsamp 540
dripwater 542
dssiactivate 544
dssiaudio 545
dssictls 546
dssiinit 547
dssilist 549
dumpk 550
dumpk2 552
dumpk3 554
dumpk4 556
duserrnd 558
else 560
elseif 561
endif 562
endin 563
endop 565
envlpx 566
envlpxr 569
ephasor 571
eqfil 572
event 574
event_i 577
exitnow 578
exp 579
expcurve 581
expon 583
exprand 585
expseg 587
expsega 589
expsegr 591
ficlose 594
filelen 596
filenchnls 598
filepeak 600

filesr ... 602
filter2 ... 604
fin ... 606
fini ... 607
fink ... 609
fiopen ... 610
flanger ... 612
flashtxt ... 614
FLbox ... 616
FLbutBank ... 621
FLbutton ... 624
FLcloseButton ... 629
FLcolor ... 632
FLcolor2 ... 634
FLcount ... 635
FLexecButton ... 638
FLgetsnap ... 641
FLgroup ... 642
FLgroupEnd ... 644
FLgroupEnd ... 645
FLhide ... 646
FLhvsBox ... 647
FLhvsBoxSetValue ... 648
FLjoy ... 649
FLkeyIn ... 652
FLknob ... 654
FLlabel ... 659
FLloadsnap ... 661
FLmouse ... 662
flooper ... 664
flooper2 ... 666
floor ... 668
FLpack ... 669
FLpackEnd ... 672
FLpack_end ... 673
FLpanel ... 674
FLpanelEnd ... 677
FLpanel_end ... 678
FLprintk ... 679
FLprintk2 ... 680
FLroller ... 681
FLrun ... 684
FLsavesnap ... 685
FLscroll ... 690
FLscrollEnd ... 693
FLscroll_end ... 694
FLsetAlign ... 695
FLsetBox ... 696
FLsetColor ... 698
FLsetColor2 ... 700
FLsetFont ... 701
FLsetPosition ... 703
FLsetSize ... 704
FLsetsnap ... 705
FLsetSnapGroup ... 707
FLsetText ... 708
FLsetTextColor ... 710
FLsetTextSize ... 711

FLsetTextType ... 712
FLsetVal_i ... 715
FLsetVal ... 716
FLshow ... 717
FLslidBnk ... 718
FLslidBnk2 ... 722
FLslidBnkGetHandle ... 725
FLslidBnkSet ... 726
FLslidBnkSetk ... 727
FLslidBnk2Set ... 729
FLslidBnk2Setk ... 730
FLslider ... 733
FLtabs ... 739
FLtabsEnd ... 744
FLtabs_end ... 745
FLtext ... 746
FLupdate ... 749
fluidAllOut ... 750
fluidCCi ... 753
fluidCCk ... 754
fluidControl ... 755
fluidEngine ... 757
fluidLoad ... 761
fluidNote ... 763
fluidOut ... 765
fluidProgramSelect ... 767
fluidSetInterpMethod ... 769
FLvalue ... 770
FLvkeybd ... 772
FLvslidBnk ... 773
FLvslidBnk2 ... 777
FLxyin ... 779
fmb3 ... 782
fmbell ... 785
fmmetal ... 788
fmpercfl ... 791
fmrhode ... 794
fmvoice ... 797
fmwurlie ... 799
fof ... 802
fof2 ... 805
fofilter ... 811
fog ... 813
fold ... 815
follow ... 817
follow2 ... 819
foscil ... 821
foscili ... 823
fout ... 825
fouti ... 830
foutir ... 832
foutk ... 834
fprintks ... 836
fprints ... 842
frac ... 844
freeverb ... 846
ftchnls ... 848
ftconv ... 850

ftfree ... 853
ftgen ... 854
ftgentmp ... 857
ftlen ... 858
ftload ... 860
ftloadk ... 861
ftlptim ... 862
ftmorf ... 864
ftsave ... 866
ftsavek ... 868
ftsr ... 869
gain ... 871
gainslider ... 872
gauss ... 874
gbuzz ... 876
getcfg ... 879
gogobel ... 880
goto ... 882
grain ... 884
grain2 ... 886
grain3 ... 890
granule ... 895
guiro ... 899
harmon ... 901
harmon2 ... 904
hilbert ... 906
hrtfer ... 910
hrtfmove ... 912
hrtfmove2 ... 915
hrtfstat ... 918
hsboscil ... 921
hvs1 ... 924
hvs2 ... 928
hvs3 ... 934
i ... 937
ibetarand ... 938
ibexprnd ... 939
icauchy ... 940
ictrl14 ... 941
ictrl21 ... 942
ictrl7 ... 943
iexprand ... 944
if ... 945
igauss ... 949
igoto ... 950
ihold ... 952
ilinrand ... 954
imagecreate ... 955
imagefree ... 957
imagegetpixel ... 959
imageload ... 961
imagesave ... 963
imagesetpixel ... 965
imagesize ... 967
imidic14 ... 969
imidic21 ... 970
imidic7 ... 971
in ... 972

in32 973
inch 974
inh 975
init 976
initc14 977
initc21 978
initc7 979
ino 980
inq 981
inrg 982
ins 983
insremot 984
insglobal 986
instimek 987
instimes 988
instr 989
int 992
integ 994
interp 996
invalue 999
inx 1000
inz 1001
ioff 1002
ion 1003
iondur 1004
iondur2 1005
ioutat 1006
ioutc 1007
ioutc14 1008
ioutpat 1009
ioutpb 1010
ioutpc 1011
ipcauchy 1012
ipoisson 1013
ipow 1014
is16b14 1015
is32b14 1016
islider16 1017
islider32 1018
islider64 1019
islider8 1020
itablecopy 1021
itablegpw 1022
itablemix 1023
itablew 1024
itrirand 1025
iunirand 1026
iweibull 1027
jacktransport 1028
jitter 1030
jitter2 1032
jspline 1034
k 1035
kbetarand 1036
kbexprnd 1037
kcauchy 1038
kdump 1039
kdump2 1040

kdump3 1041
kdump4 1042
kexprand 1043
kfilter2 1044
kgauss 1045
kgoto 1046
klinrand 1048
kon 1049
koutat 1050
koutc 1051
koutc14 1052
koutpat 1053
koutpb 1054
koutpc 1055
kpcauchy 1056
kpoisson 1057
kpow 1058
kr 1059
kread 1060
kread2 1061
kread3 1062
kread4 1063
ksmps 1064
ktableseg 1065
ktrirand 1066
kunirand 1067
kweibull 1068
lfo 1069
limit 1071
line 1072
linen 1074
linenr 1076
lineto 1077
linrand 1078
linseg 1080
linsegr 1083
locsend 1086
locsig 1088
log 1091
log10 1093
logbtwo 1095
logcurve 1097
loop_ge 1099
loop_gt 1100
loop_le 1101
loop_lt 1102
loopseg 1103
loopsegp 1105
lorenz 1107
lorisread 1110
lorismorph 1112
lorisplay 1113
loscil 1114
loscil3 1117
loscilx 1120
lowpass2 1121
lowres 1123
lowresx 1125

lpf18 1127
lpfreson 1129
lphasor 1130
lpinterp 1132
lposcil 1133
lposcil3 1134
lposcila 1135
lposcilsa 1136
lposcilsa2 1137
lpread 1138
lpreson 1140
lpshold 1141
lpsholdp 1143
lpslot 1144
mac 1146
maca 1147
madsr 1148
mandel 1151
mandol 1152
marimba 1154
massign 1157
max 1159
maxabs 1160
maxabsaccum 1161
maxaccum 1162
maxalloc 1163
max_k 1165
mclock 1166
mdelay 1167
metro 1169
midic14 1171
midic21 1173
midic7 1175
midichannelaftertouch 1177
midichn 1179
midicontrolchange 1182
midictrl 1184
mididefault 1185
midiin 1186
midinoteoff 1189
midinoteoncps 1191
midinoteonkey 1193
midinoteonoct 1195
midinoteonpch 1197
midion 1199
midion2 1202
midiout 1203
midipitchbend 1205
midipolyaftertouch 1207
midiprogramchange 1209
miditempo 1210
midremot 1211
midglobal 1214
min 1215
minabs 1216
minabsaccum 1217
minaccum 1218
mirror 1219

MixerSetLevel ... 1220
MixerGetLevel ... 1222
MixerSend ... 1223
MixerReceive ... 1224
MixerClear ... 1226
mode ... 1227
monitor ... 1230
moog ... 1231
moogladder ... 1233
moogvcf ... 1235
moogvcf2 ... 1237
moscil ... 1239
mpulse ... 1241
mrtmsg ... 1243
multitap ... 1244
mute ... 1245
mxadsr ... 1247
nchnls ... 1249
nestedap ... 1250
nlfilt ... 1253
noise ... 1255
noteoff ... 1258
noteon ... 1259
noteondur ... 1260
noteondur2 ... 1262
notnum ... 1264
nreverb ... 1266
nrpn ... 1269
nsamp ... 1270
nstrnum ... 1272
ntrpol ... 1273
octave ... 1274
octcps ... 1276
octmidi ... 1279
octmidib ... 1281
octmidinn ... 1283
octpch ... 1286
opcode ... 1289
OSCsend ... 1294
OSCinit ... 1296
OSClisten ... 1297
oscbnk ... 1301
oscil ... 1306
oscil1 ... 1308
oscil1i ... 1309
oscil3 ... 1310
oscili ... 1312
oscilikt ... 1314
osciliktp ... 1316
oscilikts ... 1318
osciln ... 1320
oscils ... 1321
oscilx ... 1323
out ... 1324
out32 ... 1325
outc ... 1326
outch ... 1327
outh ... 1328

outiat 1329
outic 1330
outic14 1331
outipat 1333
outipb 1334
outipc 1335
outkat 1336
outkc 1337
outkc14 1338
outkpat 1339
outkpb 1340
outkpc 1341
outo 1344
outq 1345
outq1 1346
outq2 1347
outq3 1348
outq4 1349
outrg 1350
outs 1351
outs1 1352
outs2 1353
outvalue 1354
outx 1355
outz 1356
p 1357
pan 1359
pan2 1361
pareq 1362
partials 1365
partikkel 1367
partikkelsync 1376
pcauchy 1377
pchbend 1379
pchmidi 1381
pchmidib 1383
pchmidinn 1385
pchoct 1388
pconvolve 1391
pcount 1394
pdclip 1396
pdhalf 1399
pdhalfy 1402
peak 1405
peakk 1407
pgmassign 1408
phaser1 1412
phaser2 1415
phasor 1419
phasorbnk 1421
pindex 1423
pinkish 1425
pitch 1428
pitchamdf 1431
planet 1434
pluck 1436
poisson 1438
polyaft 1442

polynomial ... 1444
pop ... 1446
pop_f ... 1447
port ... 1448
portk ... 1449
poscil ... 1451
poscil3 ... 1453
pow ... 1456
powershape ... 1458
powoftwo ... 1460
prealloc ... 1462
prepiano ... 1464
print ... 1467
printf ... 1469
printk ... 1470
printk2 ... 1472
printks ... 1474
prints ... 1477
product ... 1479
pset ... 1480
ptrack ... 1481
puts ... 1483
push ... 1484
push_f ... 1485
pvadd ... 1486
pvbufread ... 1489
pvcross ... 1491
pvinterp ... 1493
pvoc ... 1495
pvread ... 1497
pvsadsyn ... 1499
pvsanal ... 1501
pvsarp ... 1504
pvsbandp ... 1506
pvsbandr ... 1508
pvsbin ... 1510
pvsblur ... 1512
pvsbuffer ... 1514
pvsbufread ... 1516
pvscale ... 1518
pvscent ... 1520
pvscross ... 1521
pvsdemix ... 1522
pvsdiskin ... 1524
pvsdisp ... 1525
pvsfilter ... 1527
pvsfread ... 1529
pvsfreeze ... 1530
pvsftr ... 1532
pvsftw ... 1534
pvsfwrite ... 1536
pvshift ... 1538
pvsifd ... 1540
pvsinfo ... 1542
pvsinit ... 1543
pvsin ... 1544
pvsmaska ... 1545
pvsmix ... 1547

pvsmorph 1548
pvsmooth 1549
pvsout 1551
pvsosc 1552
pvspitch 1555
pvstencil 1558
pvsvoc 1560
pvsynth 1562
pyassign Opcodes 1564
pycall Opcodes 1565
pyeval Opcodes 1568
pyexec Opcodes 1569
pyinit Opcodes 1572
pyrun Opcodes 1573
rand 1575
randh 1577
randi 1579
random 1581
randomh 1583
randomi 1585
rbjeq 1587
readclock 1590
readk 1592
readk2 1594
readk3 1596
readk4 1598
reinit 1600
release 1602
remoteport 1603
remove 1604
repluck 1605
reson 1607
resonk 1609
resonr 1610
resonx 1613
resonxk 1615
resony 1616
resonz 1618
resyn 1620
reverb 1622
reverb2 1624
reverbsc 1625
rewindscore 1627
rezzy 1628
rigoto 1630
rireturn 1631
rms 1633
rnd 1635
rnd31 1637
round 1642
rspline 1643
rtclock 1644
s16b14 1646
s32b14 1648
scale 1650
samphold 1652
sandpaper 1653
scanhammer 1655

scans ... 1656
scantable ... 1658
scanu ... 1660
scoreline ... 1662
scoreline_i ... 1664
schedkwhen ... 1665
schedkwhennamed ... 1668
schedule ... 1670
schedwhen ... 1672
seed ... 1675
sekere ... 1676
semitone ... 1678
sense ... 1680
sensekey ... 1681
seqtime ... 1685
seqtime2 ... 1688
setctrl ... 1690
setksmps ... 1692
setscorepos ... 1694
sfilist ... 1695
sfinstr ... 1696
sfinstr3 ... 1698
sfinstr3m ... 1700
sfinstrm ... 1702
sfload ... 1704
sflooper ... 1705
sfpassign ... 1707
sfplay ... 1708
sfplay3 ... 1710
sfplay3m ... 1712
sfplaym ... 1714
sfplist ... 1716
sfpreset ... 1717
shaker ... 1719
sin ... 1721
sinh ... 1723
sininv ... 1725
sinsyn ... 1727
sleighbells ... 1729
slider16 ... 1731
slider16f ... 1733
slider32 ... 1735
slider32f ... 1737
slider64 ... 1739
slider64f ... 1741
slider8 ... 1743
slider8f ... 1745
slider16table ... 1747
slider16tablef ... 1749
slider32table ... 1751
slider32tablef ... 1753
slider64table ... 1755
slider64tablef ... 1757
slider8table ... 1759
slider8tablef ... 1761
sliderKawai ... 1763
sndload ... 1764
sndloop ... 1766

sndwarp 1768
sndwarpst 1772
socksend 1775
sockrecv 1777
soundin 1779
soundout 1782
soundouts 1784
space 1786
spat3d 1790
spat3di 1798
spat3dt 1802
spdist 1806
specaddm 1810
specdiff 1811
specdisp 1812
specfilt 1813
spechist 1814
specptrk 1815
specscal 1817
specsum 1818
spectrum 1819
splitrig 1821
spsend 1823
sprintf 1826
sprintfk 1827
sqrt 1829
sr 1831
stack 1832
statevar 1833
stix 1835
strchar 1837
strchark 1838
strcpy 1839
strcpyk 1840
strcat 1841
strcatk 1842
strcmp 1843
strcmpk 1844
streson 1845
strget 1847
strindex 1848
strindexk 1849
strlen 1850
strlenk 1851
strlower 1852
strlowerk 1853
strrindex 1854
strrindexk 1855
strset 1856
strsub 1858
strsubk 1860
strtod 1861
strtodk 1862
strtol 1863
strtolk 1864
strupper 1865
strupperk 1866
subinstr 1867

subinstrinit 1870
sum 1871
svfilter 1872
syncgrain 1875
syncloop 1877
syncphasor 1879
system 1883
tb 1885
tab 1888
tabrec 1889
table 1890
table3 1892
tablecopy 1893
tablegpw 1894
tablei 1895
tableicopy 1896
tableigpw 1897
tableikt 1898
tableimix 1900
tableiw 1902
tablekt 1904
tablemix 1906
tableng 1908
tablera 1910
tableseg 1913
tablew 1914
tablewa 1917
tablewkt 1920
tablexkt 1923
tablexseg 1926
tabmorph 1927
tabmorpha 1929
tabmorphak 1931
tabmorphi 1933
tabplay 1935
tambourine 1936
tan 1938
tanh 1940
taninv 1942
taninv2 1944
tbvcf 1946
tempest 1949
tempo 1952
tempoval 1954
tigoto 1956
timedseq 1957
timeinstk 1959
timeinsts 1961
timek 1963
times 1965
timout 1967
tival 1968
tlineto 1969
tone 1970
tonek 1971
tonex 1972
trandom 1973
tradsyn 1974

transeg ... 1976
trcross ... 1978
trfilter ... 1980
trhighest ... 1982
trigger ... 1983
trigseq ... 1985
trirand ... 1987
trlowest ... 1989
trmix ... 1990
trscale ... 1991
trshift ... 1992
trsplit ... 1993
turnoff ... 1995
turnoff2 ... 1997
turnon ... 1998
unirand ... 1999
upsamp ... 2001
urd ... 2002
vadd ... 2003
vadd_i ... 2006
vaddv ... 2008
vaddv_i ... 2011
vaget ... 2013
valpass ... 2015
vaset ... 2016
vbap16 ... 2018
vbap16move ... 2020
vbap4 ... 2022
vbap4move ... 2024
vbap8 ... 2026
vbap8move ... 2028
vbaplsinit ... 2031
vbapz ... 2033
vbapzmove ... 2035
vcella ... 2037
vco ... 2040
vco2 ... 2043
vco2ft ... 2047
vco2ift ... 2049
vco2init ... 2051
vcomb ... 2054
vcopy ... 2057
vcopy_i ... 2060
vdelay ... 2062
vdelay3 ... 2064
vdelayx ... 2066
vdelayxq ... 2068
vdelayxs ... 2070
vdelayxw ... 2072
vdelayxwq ... 2074
vdelayxws ... 2076
vdivv ... 2078
vdivv_i ... 2081
vdelayk ... 2083
vecdelay ... 2084
veloc ... 2085
vexp ... 2087
vexp_i ... 2090

vexpseg ... 2092
vexpv ... 2094
vexpv_i ... 2097
vibes ... 2099
vibr ... 2101
vibrato ... 2103
vincr ... 2106
vlimit ... 2107
vlinseg ... 2108
vlowres ... 2110
vmap ... 2112
vmirror ... 2114
vmult ... 2115
vmult_i ... 2119
vmultv ... 2121
vmultv_i ... 2124
voice ... 2126
vosim ... 2129
vphaseseg ... 2134
vport ... 2136
vpow ... 2137
vpow_i ... 2140
vpowv ... 2142
vpowv_i ... 2145
vpvoc ... 2147
vrandh ... 2149
vrandi ... 2152
vstaudio, vstaudiog ... 2155
vstbankload ... 2156
vstedit ... 2157
vstinit ... 2158
vstinfo ... 2159
vstmidiout ... 2160
vstnote ... 2162
vstparamset,vstparamget ... 2164
vstprogset ... 2166
vsubv ... 2167
vsubv_i ... 2170
vtable1k ... 2172
vtablei ... 2174
vtablek ... 2176
vtablea ... 2178
vtablewi ... 2180
vtablewk ... 2181
vtablewa ... 2183
vtabi ... 2185
vtabk ... 2187
vtaba ... 2189
vtabwi ... 2191
vtabwk ... 2192
vtabwa ... 2193
vwrap ... 2194
waveset ... 2195
weibull ... 2197
wgbow ... 2199
wgbowedbar ... 2201
wgbrass ... 2203
wgclar ... 2205

wgflute ... 2207
wgpluck ... 2209
wgpluck2 ... 2212
wguide1 ... 2214
wguide2 ... 2216
wrap ... 2219
wterrain ... 2220
xadsr ... 2222
xin ... 2224
xout ... 2226
xscanmap ... 2228
xscansmap ... 2229
xscans ... 2230
xscanu ... 2232
xtratim ... 2235
xyin ... 2238
zacl ... 2240
zakinit ... 2242
zamod ... 2245
zar ... 2247
zarg ... 2249
zaw ... 2251
zawm ... 2253
zfilter2 ... 2256
zir ... 2258
ziw ... 2260
ziwm ... 2262
zkcl ... 2264
zkmod ... 2266
zkr ... 2268
zkw ... 2270
zkwm ... 2272
Instructions de Partition et Routines GEN ... 2275
Instructions de Partition ... 2275
Instruction a (ou Instruction Avancer) ... 2276
Instruction b ... 2277
Instruction e ... 2278
Instruction f (ou Instruction de Table de Fonction) ... 2279
Instruction i (Instruction d'Instrument ou de Note) ... 2282
Instruction m (Instruction de Marquage) ... 2286
Instruction n ... 2287
Instruction q ... 2288
Instruction r (Instruction Répéter) ... 2289
Instruction s ... 2291
Instruction t (Instruction de Tempo) ... 2292
Instruction v ... 2293
Instruction x ... 2295
{ Statement ... 2296
} Statement ... 2299
Routines GEN ... 2299
GEN01 ... 2303
GEN02 ... 2306
GEN03 ... 2308
GEN04 ... 2310
GEN05 ... 2312
GEN06 ... 2314
GEN07 ... 2316
GEN08 ... 2318

GEN09 ... 2320
GEN10 ... 2323
GEN11 ... 2325
GEN12 ... 2327
GEN13 ... 2329
GEN14 ... 2332
GEN15 ... 2335
GEN16 ... 2336
GEN17 ... 2339
GEN18 ... 2340
GEN19 ... 2341
GEN20 ... 2343
GEN21 ... 2345
GEN22 ... 2347
GEN23 ... 2348
GEN24 ... 2349
GEN25 ... 2350
GEN27 ... 2351
GEN28 ... 2352
GEN30 ... 2354
GEN31 ... 2355
GEN32 ... 2356
GEN33 ... 2358
GEN34 ... 2360
GEN40 ... 2362
GEN41 ... 2363
GEN42 ... 2364
GEN43 ... 2365
GEN51 ... 2366
GEN52 ... 2368
Les Programmes Utilitaires ... 2369
Répertoires. ... 2369
Formats des Fichiers Son. ... 2369
Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL) ... 2370
Requêtes sur un Fichier (SNDINFO) ... 2381
Conversion de Fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EXPORT, PV_IMPORT, SDIF2AD, SRCONV) ... 2383
Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MAKECSD, MIXER, SCALE) ... 2398
Cscore ... 2411
Evénements, Listes et Opérations ... 2411
Ecrire un Programme de Contrôle Cscore ... 2414
Compiler un Programme Cscore ... 2419
Exemples Plus Avancés ... 2422
Etendre Csound ... 2424
Ajouter des Générateurs Unitaires ... 2424
Créer un Générateur Unitaire Intégré ... 2424
Ajouter un Générateur Unitaire comme Plugin ... 2428
Référence de OENTRY ... 2428
IV. Référence Rapide des Opcodes ... 2431
Référence Rapide des Opcodes ... 2433
A. Conversion de Hauteur ... 2477
B. Valeurs d'Intensité du Son ... 2481
C. Valeurs de Formant ... 2482
D. Rapports de Fréquence Modale ... 2487
E. Fonctions Fenêtres ... 2489
F. Format de Fichier SoundFont2 ... 2494

G. Csound Double (64 bit) ou Float (32 bit) 2495
Glossaire 2497

Préface

Table des matières

Préface du Manuel de Csound xxix
Histoire du Manuel de Référence Canonique de Csound xxx
Mentions de copyright xxxi
Débuter avec Csound xxxiii
Les nouveautés de Csound 5.10 xxxv

Préface du Manuel de Csound

Barry Vercoe, MIT Media Lab

La réalisation de musique par ordinateur nécessite la synthèse de signaux audio avec des points discrets ou échantillons représentant des formes d'onde continues. Il y a de nombreuses façons de faire ceci, chacune offrant un type de contrôle différent. La synthèse directe génère des formes d'onde en échantillonnant une fonction enregistrée représentant une simple période ; la synthèse additive génère les nombreux partiels d'un son complexe, chacun ayant sa propre enveloppe d'intensité ; la synthèse soustractive démarre avec un son complexe pour le filtrer. La synthèse non-linéaire utilise la modulation de fréquence et la distorsion non-linéaire pour donner des caractéristiques complexes à des signaux simples, tandis que l'échantillonnage et l'enregistrement d'un son naturel permettent de l'utiliser à volonté.

Comme la spécification détaillée d'un son point par point est vite ennuyeuse, le contrôle est opéré de deux manières : 1) à partir d'instruments dans un orchestre, et 2) à partir d'évènements dans une partition. Un orchestre est en fait un programme d'ordinateur qui peut produire des sons, tandis qu'une partition est un ensemble de données auxquelles ce programme réagit. Qu'une durée d'attaque soit une constante fixée dans un instrument, ou une variable de chaque note dans la partition, dépend de la façon dont l'utilisateur veut la contrôler.

Les instruments d'un orchestre de Csound (voir *Syntaxe de l'Orchestre*) sont définis dans une syntaxe simple qui invoque des procédures de traitement audio complexe. Une partition (voir *La Partition Numérique Standard*) passée à cet orchestre contient des informations de hauteur et de contrôle codées dans un format numérique standard. Bien que la plupart des utilisateurs se contentent de ce format, des langages de traitement de partition de plus haut niveau sont souvent pratiques.

Les programmes constituant le système Csound ont une longue histoire de développement, qui a commencé avec le programme Music 4 écrit aux Bell Telephone Laboratories au début des années 1960 par Max Mathews. C'est là que fut conçu le concept de table d'onde ainsi qu'une grande partie de la terminologie qui a permis depuis aux chercheurs de l'informatique musicale de communiquer. D'importantes additions furent apportées à Princeton par feu Godfrey Winham dans Music 4B ; mon propre Music 360 (1968) doit beaucoup à ce travail. Avec Music 11 (1973) j'ai pris une voie différente : les deux structures distinctes des signaux de contrôle et des signaux audio sont issues de mon engagement intensif lors des années précédentes dans la conception et l'élaboration de synthétiseurs numériques. Cette division a été retenue dans Csound.

Parce qu'il est entièrement écrit en C, on peut installer facilement Csound sur n'importe quelle machine équipée de Unix ou du langage C. Au MIT il tourne sur des stations VAX/DEC sous Ultrix 4.2, sur des machines SUN sous OS 4.1, sur SGI sous 5.0, sur IBM PC sous DOS 6.2 et Windows 3.1, et sur le Macintosh d'Apple sous ThinkC 5.0. Avec ce seul langage de définition de traitement numérique du signal et des formats audio portables comme AIFF et WAV, les utilisateurs peuvent passer facilement d'une machine à l'autre.

La version de 1991 apporta le vocodeur de phase, FOF, et les types de données spectrales. 1992 vit l'arrivée des convertisseurs et des unités de contrôle MIDI, permettant de piloter Csound depuis des fichiers MIDI (midifiles) et des claviers externes. En 1994 les programmes d'analyse du son (lpc, pvoc) furent intégrés dans le module principal, permettant de lancer tous les traitements de Csound depuis un seul exécutable, et Cscore pouvait passer les partitions directement à l'orchestre pour une réalisation itérative. La version de 1995 introduisit un ensemble MIDI étendu avec linseg basé sur MIDI, les filtres de Butterworth, la synthèse granulaire, et un détecteur de hauteur amélioré, dans le domaine fréquentiel. L'addition d'outils de génération d'évènements en temps-réel (Cscore et MIDI) fut particulièrement importante, permettant des configurations excitation/réponse en temps-réel qui rendent possible la composition et l'expérimentation interactives. Il est apparu que la synthèse numérique par programme en temps-réel était désormais réellement prometteuse.

Histoire du Manuel de Référence Canonique de Csound

La version initiale de ce manuel pour les premières versions de Csound fut démarrée au MIT par Barry L. Vercoe et y fut maintenue durant les années 1980 et le début des années 1990. Une partie du manuel provient de documents pour des programmes des années 1970 comme *Music11*. Ce manuel original fut amélioré et développé par Richard Boulanger, John ffitch, Jean Piché et Rasmus Ekman.

Ce manuel évolua vers le Manuel de Référence Officiel de Csound que l'on trouve toujours à *http://www.lakewoodsound.com/csound* [http://www.lakewoodsound.com/csound/hypertext/manual.htm], pour la version 4.16 de Csound, de novembre 1999, qui était maintenu par David M. Boothe.

Une version parallèle du manuel appelée le Manuel de Référence Alternatif de Csound, fut développée par Kevin Conder en utilisant *DocBook/SGML* [http://www.docbook.org/]. Cette version devint plus tard la version Canonique.

Quand le MIT plaça Csound sous license LGPL en 2003, le manuel passa sous license GFDL et fut placé sur Sourceforge avec les sources de Csound.

Durant l'hiver 2004, le Manuel Canonique fut converti en DocBook/XML par Steven Yi afin de permettre à plus de gens d'assurer sa compilation et sa maintenance.

Le manuel est actuellement maintenu par Andrés Cabrera avec des contributions continues de la communauté de Csound.

Le manuel est toujours un projet communautaire qui dépend des contributions des développeurs et des utilisateurs afin d'aider à affiner l'étendue et la précision de son contenu. Toutes les contributions sont les bienvenues et sont appréciées.

Tableau 1. Autres Collaborateurs

Mike Berry
Eli Breder
Michael Casey
Michael Clark
Perry Cook
Sean Costello
Richard Dobson
Mark Dolson

Dan Ellis
Tom Erbe
Bill Gardner
Michael Gogins
Matt Ingalls
Richard Karpen
Anthony Kozar
Victor Lazzarini
Allan Lee
David Macintyre
Gabriel Maldonado
Max Mathews
Hans Mikelson
Peter Neubäcker
Peter Nix
Ville Pulkki
Maurizio Umberto Puxeddu
John Ramsdell
Marc Resibois
Rob Shaw
Paris Smaragdis
Greg Sullivan
Istvan Varga
Bill Verplank
Robin Whittle
Steven Yi
François Pinot

Cette liste n'est en aucune façon exhaustive. On peut obtenir plus d'information par le fichier Changelog dans l'entrepôt des sources du manuel.

Mentions de copyright

Cette version du Manuel de Csound ("Le Manuel Canonique de Csound") est délivrée sous la GNU Free Documentation Licence [http://www.gnu.org/licenses/fdl.txt]. Les mentions de copyright antérieures et le crédit de leurs auteurs sont donnés ci-dessous, pour des raisons historiques.

Mentions de copyright antérieures

Copyright © 1986, 1992 par le Massachusetts Institute of Technology. Tous droits réservés.

Développé par *Barry L. Vercoe* au Experimental Music Studio, Media Laboratory, M.I.T., Cambridge, Massachusetts, avec le support partiel de la System Development Foundation et du National Science Foundation Grant # IRI-8704665.

Manuel

Copyright © 2003 by Kevin Conder pour les modifications apportées au Manuel de Référence Publique de Csound.

Il est permis de copier, distribuer et/ou modifier ce document selon les termes de la GNU Free Documentation License, Version 1.2 ou toute version ultérieure publiée par la Free Software Foundation ; sans aucune partie non modifiable, aucun texte de première de couverture et aucun texte de quatrième de couverture. Une copie de cette licence est disponible dans le sous-répertoire des exemples [examples/fdl.txt] ou à : www.gnu.org/licenses/fdl.txt [http://www.gnu.org/licenses/fdl.txt].

La documentation du langage Csound de ce manuel est dérivée du *Manuel de Référence Alternatif de Csound* de Kevin Conder, qui est lui-même dérivé du *Manuel de Référence Public de Csound*.

Copyright © 2004-2005 par Michael Gogins pour les modifications faites au *Manuel de Référence Alternatif de Csound*.

Cette mention légale provient du *Manuel de Référence Public de Csound* : « L'Edition Hypertexte originale du Manuel de Csound du MIT fut préparée pour le World Wide Web par *Peter J. Nix* du Department of Music at the University of Leeds et *Jean Piché* de la Faculté de musique de l'Université de Montréal. Une Edition d'Impression, en format Adobe Acrobat, fut ensuite maintenue par *David M. Boothe*. Les éditeurs reconnaissent entièrement les droits des auteurs de la documentation et des programmes originaux, comme décrits ci-dessus, et demandent en conséquence que cette mention soit citée chaque fois que ce matériel est utilisé. »

La dernière adresse réseau connue du Manuel de Référence Public de Csound était http://www.lakewoodsound.com/csound/hypertext/manual.htm.

L'adresse réseau du Manuel de Référence Alternatif de Csound, pour les copies Transparentes et les copies Opaques, est http://kevindumpscore.com/download.html#csound-manual.

L'adresse réseau du manuel de Csound et de CsoundAC est http://sourceforge.net/projects/csound.

Traduction française du manuel par François Pinot.

La traduction française du manuel est placée sous GNU Free Documentation License, Version 1.2 ou ultérieure, comme la version anglaise originale.

Csound et CsoundAC

Csound est protégé par copyright de 1991 à 2008 par Barry Vercoe, John ffitch et les autres développeurs.

CsoundAC est protégé par copyright de 2001 à 2008 par Michael Gogins.

Csound et CsoundAC (anciennement CsoundVST) sont des logiciels libres ; vous pouvez les redistribuer et/ou les modifier selon les termes de la GNU Lesser General Public License tels que publiés par la Free Software Foundation ; soit la version 2.1 de la License, soit (à votre choix) n'importe quelle version ultérieure.

Csound et CsoundAC sont distribués dans l'espoir qu'il seront utiles, mais SANS AUCUNE GARANTIE ; sans même la garantie implicite de la VALEUR COMMERCIALE ou de l'ADEQUATION A UNE UTILISATION SPECIALE. Consultez la GNU Lesser General Public License pour plus de détails.

Vous devez avoir reçu une copie de la GNU Lesser General Public License en même temps que Csound et CsoundAC ; si ce n'est pas le cas, écrivez à la Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Virtual Synthesis Technology

Virtual Synthesis Technology (VST) PlugIn technologie d'interfaçage par Steinberg Soft- und Hardware GmbH.

Débuter avec Csound

Téléchargement

Si vous n'avez pas déjà installé Csound (ou si vous avez une ancienne version) téléchargez la version de Csound adaptée à votre plate-forme depuis la *Sourceforge Csound5 Download Page* [http://sourceforge.net/project/showfiles.php?group_id=81968&package_id=120482]. Les installeurs pour Windows ont un suffixe '.exe' et ceux pour le Mac '.dmg' ou '.tar.gz'. Si le nom de l'installeur se termine en '-d' cela veut dire que l'installeur a été construit avec la *double* précision (64-bit) qui produit une sortie de meilleure qualité que la *simple* précision (32-bit), qui produit plus rapidement la sortie. Vous pouvez aussi télécharger les sources et les compiler, mais cela réclame plus d'expertise (voir la section *Construire Csound*).

Il est aussi utile de télécharger la version la plus récente de ce manuel, que vous trouverez également sur ce site.

Exécution

Il y a différentes manières d'exécuter Csound. Comme Csound est un programme en ligne de commande (DOS dans la terminologie Windows), double-cliquer sur l'exécutable de Csound n'aura aucun effet. On doit appeler Csound soit depuis un terminal (ou invite DOS), soit depuis un frontal. Pour utiliser Csound en ligne de commande, vous devez ouvrir un *terminal* (une invite de commande DOS sous Windows). L'utilisation de Csound en ligne de commande pouvant sembler difficile si vous n'avez jamais utilisé de terminal, vous voudrez peut-être essayer un des frontaux inclus dans votre distribution. Un *frontal* est un programme graphique qui facilite l'exécution de Csound et peut souvent aider à éditer les fichiers csound.

Que ce soit avec un frontal ou en ligne de commande, l'exécution de Csound nécessite deux choses :

- Un fichier Csound ('.csd' ou bien un fichier '.orc' et un fichier '.sco')
- Une liste de drapeaux de ligne de commande (ou options de configuration) qui configurent l'exécution. Ils déterminent des éléments comme le nom et le format du fichier de sortie, si le temps-réel audio et le MIDI sont actifs, quelle carte son utiliser, la taille des tampons, la quantité de messages imprimés, etc. On peut inclure ces options dans le fichier '.csd' lui-même, aussi dans le cas des exemples inclus dans ce manuel, *vous ne devriez pas avoir en vous en soucier*. Vous pouvez trouver la liste complète et très longue des options de ligne de commande *ici*, mais vous voudrez peut-être la consulter plus tard...

Consultez la section *Configuration* si vous rencontrez des problèmes avec Csound.

Cette documentation comprend de nombreux fichiers '.csd' que vous pouvez tester, et qui devraient fonctionner directement depuis la ligne de commande ou depuis n'importe quel frontal. *oscil.csd* [examples/oscil.csd] est un exemple simple que l'on peut trouver dans le répertoire des *exemples* de cette documentation. Votre frontal devrait vous permettre de choisir le fichier, et il devrait avoir un bouton 'play' ou 'render'.

Note pour les utilisateurs de MacCsound

Il peut être nécessaire d'effacer toutes les lignes de la balise des options de commande afin

de faire fonctionner les exemples du manuel.

Vous pouvez aussi essayer les exemples à partir de la ligne de commande en vous déplaçant dans le répertoire des exemples du manuel avec ce type de commande sous Windows (en supposant que le manuel est situé en c:\Program Files\Csound\manual\) :

```
cd "c:\Program Files\Csound\manual\examples"
```

ou quelque chose comme :

```
cd /manualdirectory/manual/examples
```

pour les terminaux Mac ou linux et en tapant ensuite :

```
csound oscil.csd
```

Les fichiers exemples étant configurés pour fonctionner en temps-réel par défaut, vous devriez avoir entendu une onde sinusoïdale de 2 secondes.

Ecrire vos propres fichiers .csd

Un fichier *.csd* ressemble à ceci (ce fichier est *oscils.csd* [examples/oscils.csd]) :

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscils.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a fast sine oscillator.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Les fichiers *.csd* de Csound comprennent 3 sections principales entre les balises *<CsSynthesizer>* et *</CsSynthesizer>* :

- *CsOptions* - Contient les *options de ligne de commande* spécifiques à ce fichier particulier. Ces options peuvent aussi être définies dans le fichier *.csoundrc* ou directement dans la *ligne de commande*. Certains frontaux offrent également des moyens de spécifier les options globales ou locales.

- *CsInstruments* - Contient les instruments ou processus disponibles dans ce fichier. Les instruments sont définis en utilisant les codes d'opération *instr* et *endin*. La section *CsInstruments* contient aussi l'*En-tête de l'Orchestre* qui définit des choses comme le *taux d'échantillonnage*, le *nombre d'échantillons dans une période de contrôle* et le *nombre de canaux de sortie*.

- *CsScore* - Contient les 'notes' à jouer et optionnellement la définition de f-tables. Les notes sont crées en utilisant l'*instruction i*, et les f-tables sont crées en utilisant l'*instruction f*. Plusieurs autres *instructions de partition* sont disponibles.

Notez que tout ce qui suit un point-virgule (;) jusqu'à la fin de la ligne est un commentaire, et est ignoré par csound.

Vous pouvez écrire les fichiers csd dans n'importe quel éditeur de texte pur comme notepad ou textedit. Assurez vous de sauvegarder le fichier en texte pur (et non en texte enrichi).De nombreux *frontaux* proposent des capacités d'édition avancées avec coloration syntaxique et complétion.

Vous pouvez trouver ici [http://csound.sourceforge.net/tutorial.pdf] un tutoriel détaillé pour débuter avec Csound écrit par Michael Gogins.

Les nouveautés de Csound 5.10

Nouveautés dans la Version 5.10 (Décembre 2008)

- Nouvelles fonctionnalités :
  - Nouvelle option pour écouter tous les périphériques MIDI avec le module temps réel de portmidi. Pour activer l'écoute de tous les périphériques utiliser "-+rtmidi=portmidi -Ma".
  - Implémentation du dithering sur la sortie ; le dithering rectangulaire et triangulaire est disponible dans certains cas.
  - Le type 6 de *GEN20* a maintenant une option pour fixer la variance.
- Bogues corrigés et améliorations :
  - La variable d'environnement Locale a été fixée à "C numeric" pour éviter les problèmes de point décimal (, vs .).
  - Bogue corrigé dans *diskin*
  - *outo* était défectueux pour le canal 6
  - Bogue corrigé dans *pitchamdf*
  - L'initialisation de *zfilter2* a été corrigée
  - Bogue corrigé dans *s32b14*
  - D'autres bogues qui n'avaient pas été publiés ont été corrigés.
- Changements Internes :

- La version majeure de l'API de Csound est incrémentée à 2, ceci affectant aussi csound.so. Ceci signifie que Csound 5.10 est incompatible avec les applications (frontaux, clients ou hôtes) construites pour Csound 5.08 et pour les versions antérieures qui utilisent la version 1.x de l'API. Il faudra reconstruire ces applications pour qu'elles fonctionnent avec les versions courante ou futures de Csound. Les frontaux pour Csound écrits dans des langages interprétés tels que Python ou Java pourront continuer à fonctionner sans modification. Il est également possible de garder une version antérieure de la bibliothèque de Csound et une version de l'API 2.0 sur la même machine afin que les applications basées sur l'ancienne ou sur la nouvelle version de Csound puissent coexister. Ces changements n'affectent en rien la compatibilité des orchestres et des partitions de Csound : tous les anciens documents devraient continuer à fonctionner comme par le passé.
- Le temps est maintenant mesuré en interne en échantillons, ce qui résoud un ancien bogue concernant l'arrondi du temps au taux-k.
- Plusieurs changements internes concernant les branchements. Certains opcodes sont significativement plus rapides.
-

Nouveautés dans la Version 5.09 (Octobre 2008)

- Nouveaux opcodes :
  - Nouvel opcode *vosim* par Rasmus Ekman qui recrée la technique historique VOSIM (VOcal SIMulator).
  - Nouvel opcode *dcblock2* par Victor Lazzarini.
  - Nouveau modèle de l'oscillateur de Chua : *chuap* par Michael Gogins.
  - Nouveaux opcodes d'*Algèbre Linéaire* par Michael Gogins. Algèbre linéaire standard sur vecteurs et matrices réels et complexes : arithmétique composante à composante, normes, transposition et conjugaison, produits scalaires, matrice inverse, décomposition LU, décomposition QR et décomposition QR en valeurs propres. Inclut la copie de vecteur de et vers des signaux de taux-a, des tables de fonction et des signaux-f.
  - Nouveaux opcodes ambisonic : *bformdec1* et *bformenc1*. Ces opcodes rendent obsolètes les anciens *bformdec* et *bformenc*.
  - Nouveaux opcodes de contrôle de la partition par Victor Lazzarini : *rewindscore*et *setscorepos*.
- Nouvelles fonctionnalités :
  - Les opcodes de la famille *vbap* (*vbap4*, *vbap8*, *vbap16* et *vbapz*) acceptent maintenant des variables de taux-k pour tous leurs arguments en entrée.
  - Nouveau module d'E/S pulseaudio sous Linux.
  - Nouveau paramètre facultatif *ienv* pour générer des enveloppes pour les opcodes de soundfont : *sfplay*, *sfplay3*, *sfplaym* et *sfplay3m*.
  - Ajout d'un 'argument de non-normalisation' à la routine GEN nommée "tanh". (Voir *Routines GEN Nommées*)
  - Ajout d'une option d'ordonnanceur de priorité sur alsa.

- Bogues corrigés et améliorations :
  - Notation scientifique permise dans *GEN23* (comme c'était le cas dans csound4 !).
  - Bogue fixé dans l'initialisation de FLTK. L'utilisation de FLTK devrait être plus stable.
  - L'erreur sur les commentaires /* */ dans l'orchestre a été corrigée.
  - *poscil* n'écrase plus la fréquence si la variable est partagée.
  - *printk* et *printks* vérifient que l'opcode est initialisé.
  - *soundout* et *soundouts* sont déclarés obsolètes en faveur de *fout*.
  - L'opcode *space* a été modifié pour accepter des tables dont la taille n'est pas une puissance de 2 (taille différée).
  - Un bogue de *pvsmorph* a été corrigé.
- Changements Internes :
  - Le nouveau parseur reconnaît #include et les macros sans argument.
  - Moins de forçage de type entre floats et doubles dans la version float.
  - Un support expérimental multicore a été inclut.
  - L'opcode *buzz* a été réécrit.
  - Plusieurs autres changements internes et quelques corrections de bogues.

Nouveautés dans la Version 5.08 (Février 2008)

- Nouveaux opcodes :
  - *imagecreate*, *imagesize*, *imagegetpixel*, *imagesetpixel*, *imagesave*, *imageload* et *imagefree*: nouveaux opcodes de traitement d'image par Cesare Marilungo pour lire/écrire des images png depuis Csound.
  - *pvsbandp* et *pvsbandr* par John ffitch, qui réalisent le filtrage passe-bande et réjection de bande dans le domaine spectral sur un signal pvs.
  - Nouveaux opcodes HRTF par Brian Carty : *hrtfmove*, *hrtfmove2* et *hrtfstat*.
  - Nouveau opcodes de distorsion non-linéaire : *powershape*, *polynomial*, *chebyshevpoly*, *pdclip*, *pdhalf*, *pdhalfy*, et *syncphasor*
  - Nouvel opcode de contrôle de transport jack : *jacktransport*
- Nouvelles fonctionnalités :
  - Ajout de l'option de ligne de commande *--csd-line-nums=* pour sélectionner la façon de rapporter la ligne d'une erreur.
  - Nouvel opérateur de "non-report" (*!*) du langage de partition qui empêche le report implicite des p-champs dans les instructions i.

  - Ajout de l'option de ligne de commande *--syntax-check-only* (mutuellement exclusive avec *--i-only*)
  - Balise *<CsLicence>* pour les *CSDs*. *<CsLicense>* est acceptée comme une alternative à *<CsLicence>*.

- Bogues corrigés et améliorations :
  - L'ordre des sorties pour *hilbert* a été changé. Ce changement brise la compatibilité avec les versions précédentes, mais il fixe l'opcode qui travaille maintenant comme c'est décrit dans la documentation.
  - Les messages sur le chargement de plugins d'opcode ont été modifiés pour pouvoir être supprimés avec une option de niveau de message.
  - Changements majeurs sur les rapport d'erreur de partition ; les numéros de ligne dans la chaîne des entrées sont rapportés précisement pour la plupart des erreurs.
  - *pan2* a été corrigé afin d'être conforme à la documentation.
  - La balise *<CsVersion>* fonctionne à nouveau en conformité avec le manuel.
  - Les instructions de boucle { et } ont été fixées. Leur documentation ainsi que celles des opérateurs des expressions de partition ~, &, |, et # ont été ajoutées.
  - *hilbert* avait ses sorties permutées, c'est corrigé. L'exemple de manual a été mis à jour.
- Changements Internes :
  - Changement de la localisation pour gettext ; les traductions en français et en espagnol (Colombie) sont disponibles.
  - Changements internes dans *partikkel*, interpolation de la lecture de forme d'onde et du fenêtrage, ce qui permet une synthèse granulaire synchrone de hauteur plus précise. Exemples mis à jour pour *partikkel*.
  - *pvscale* : algorithme amélioré pour la SDFT, supprimant la variation d'amplitude.

Nouveautés dans la Version 5.07 (Octobre 2007)

- Nouveaux opcodes :
  - *pan2* : un opcode de spatialisation stéréo
  - *cpsmidinn*, *pchmidinn*, *octmidinn* : des convertisseurs de numéros de note MIDI
  - *fluidSetInterpMethod* : interpolation dans les SoundFonts de fluid
  - *sflooper* : une version SoundFont de *flooper2*
  - *pvsbuffer* et *pvsbufread* : mise en tampon/lecture de fsigs pour des changements de retards/échelle temporelle.
- Nouvelles fonctionnalités :
  - SDFT - la Transformée de Fourier Discrète à fenêtre Glissante -- intégrée aux opcodes *pvsanal*, etc

si le recouvrement est inférieur à *ksmps* ou inférieur à 10. Certains opcodes pvsXXX sont étendus pour prendre des paramètres de taux-a dans cette situation.

- Nouvelle option (*-O null* / *--logfile=null*) qui désactive tous les messages et toutes les impressions sur la console.

- Bogues corrigés et améliorations :
  - *partikkel* -- la synthèse par particule avait un bogue accidentel, corrigé.
  - La fermeture de l'entrée MIDI sur Windows(MM) échouait ; corrigé.
  - L'opcode *fluidEngine* prend maintenant comme paramètres facultatifs le nombre de canaux (compris entre 16 et 256) et le nombre de voix à jouer en polyphonie (compris entre 16 et 4096, la valeur par défaut étant 4096).
  - L'utilitaire *atsa* se comporte de manière plus sûre lorsqu'il reçoit du silence.
  - *ATSaddnz* : vérifications améliorées.
  - Les ambisonics (*bformdec*, *bformenc*) ont plus d'options pour le contrôle des opposés.
  - Le bogue dans *turnoff2* est corrigé.
  - *het_export* : une vérification erronée plantait l'export.
- Changements Internes :
  - L'installeur sous Windows a été amélioré.
  - CsoundVST a été remplacé par CsoundAC, qui ne dépend pas des en-têtes du SDK de VST.
  - Moins de messages lors du démarrage sous Windows(MM).
  - Le type d'argument p (le taux-k vaut alors 1 par défaut) a été ajouté dans les types des paramètres d'entrée et de sortie des opcodes.

Nouveautés dans la Version 5.06 (Juin 2007)

- Nouveaux opcodes granulaires : *partikkel*, *partikkelsync* et *diskgrain*.
- Nouvel opcode pour distribuer des évènements : *scoreline*.
- Plusieurs nouveaux opcodes en provenance de CsoundAV de Grabriel Maldonado : *hvs1*, *hvs2*, *hvs3*, *vphaseseg*, *inrg*, *outrg*, *lposcila*, *lposcilsa*, *lposcilsa2*, *tabmorph*, *tabmorpha*, *tabmorphi*, *tabmorphak*, *trandom*, *vtable1k*, *slider8table*, *slider16table*, *slider32table*, *slider64table*, *slider8tablef*, *slider16tablef*, *slider32tablef*, *slider64tablef*, *sliderKawai* et la version au taux-a de *ctrl7*.
- Egalement depuis CsoundAV, plusieurs nouveaux contrôleurs graphiques FLTK : *FLkeyIn*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLmouse*, *FLxyin*, *FLhvsBox*, *FLslidBnkSet*, *FLslidBnkSetk*, *FLslidBnk2Set*, *FLslidBnk2Setk*, *FLslidBnkGetHandle*,
- De nouveaux opcodes pvs : *pvsdiskin*, *pvsmorph*,
- *eqfil*

- De nouvelles options de ligne de commande (*--m-warnings*) pour contrôler les messages
- csladspa : un kit de plugin CSD vers LADSPA.
- Et plusieurs corrections de bogues parmi lesquelles : version au taux-k de *system* ; problèmes de changement d'échelle de *vrandh* et de *vrandi* ; plantage occasionnel de *turnoff* ; bogue OS X ; *ATScross* et *mod*.

  Csound5GUI fonctionne maintenant correctement sur toutes les plates-formes et csoundapi~ (objet pd) a été mis à jour.

Partie I. Vue d'Ensemble

Table des matières

Introduction 4
Développements Récents 5
Caractéristiques de Csound 5 5
Caractéristiques de CsoundAC 6
La commande Csound 8
Ordre de priorité 8
Description de la syntaxe de la commande 8
Csound command line 10
Options de Ligne de Commande (par Catégorie) 20
Variables d'Environnement de Csound 30
Format de Fichier Unifié pour les Orchestres et les Partitions 33
Description 33
Exemple 35
Fichier de Paramètres de Ligne de Commande (.csoundrc) 36
Prétraitement du Fichier Partition 36
La Fonction Extract 36
Prétraitement Indépendant avec Scsort 36
Utiliser Csound 38
Comment Csound5 fonctionne 38
Valeurs d'amplitude dans Csound 39
Audio en temps-réel 41
Entrées/Sorties en temps-réel sur Linux 42
Windows 47
Mac 48
Optimisation de la Latence Audio en E/S 48
Configuration 50
Syntaxe de l'Orchestre 51
Instructions de l'En-tête de l'Orchestre 52
Instructions de Bloc d'Instrument et d'Opcode 52
Instructions Ordinaires 53
Constantes et Variables 53
Initialisation de Variable 54
Expressions 55
Répertoires et Fichiers 55
Nomenclature 56
Macros 56
Instruments Nommés 57
Opcodes Définis par l'Utilisateur (UDO) 59
La Partition Numérique Standard 61
Prétraitement des Partitions Standard 61
Carry 61
Tempo 62
Sort 62
Instructions de Partition 63
Symboles Next-P et Previous-P 63
Ramping 64
Macros de Partition 65
Partition dans Plusieurs Fichiers 67
Evaluation des Expressions 68
Chaînes de caractères dans les p-champs 69
Frontaux 71
CsoundAC 71
CsoundVST 73

TclCsound 76
L'interpréteur Tcl : cstclsh 76
Cswish: le shell de fenêtrage 76
Un serveur Csound 77
Un Environnement de Scripting 78
TclCsound comme encapsuleur de langage 79
Référence des Commandes de TclCsound 79
Construire Csound 82
Liens Csound 88

Introduction

Csound est un système de musique par ordinateur basé sur des générateurs unitaires et programmable par l'utilisateur. Il fut écrit à l'origine par Barry Vercoe au Massachusetts Institute of Technology en 1984 comme la première version en langage C de ce type de logiciel. Depuis, Csound a reçu de nombreuses contributions de la part de chercheurs, de programmeurs et de musiciens du monde entier.

Vers 1991, John ffitch porta Csound sur Microsoft DOS. De nos jours, Csound tourne sur plusieurs variétés de UNIX et de Linux, sur Microsoft DOS et Windows, sur toutes les versions du système d'exploitation du Macintosh y compris Mac OS X, et sur d'autres systèmes.

Il y a des systèmes de musique par ordinateur plus récents qui ont des éditeurs graphiques de patch (par exemple Max/MSP, PD, jMax, ou Open Sound World), ou qui utilisent des techniques d'ingénieurie logicielle plus avancées (par exemple Nyquist ou SuperCollider). Cependant Csound possède toujours l'ensemble le plus important et le plus varié de générateurs unitaires, est le mieux documenté, s'exécute sur le plus grand nombre de plates-formes, et il est très facilement extensible. Il est possible de compiler Csound en utilisant l'arithmétique double précision pour obtenir une qualité sonore supérieure. Bref, on peut considérer Csound comme l'un des instruments de musique les plus puissants jamais créé.

En plus de cette version "canonique" de Csound et de CsoundAC, il existe d'autres versions de Csound et d'autres frontaux pour Csound, dont la plupart se trouvent sur http://www.csounds.com.

Développements Récents

Depuis l'époque à laquelle Barry Vercoe écrivit la Préface originale de ce manuel, imprimée ci-dessus, de nombreuses nouvelles contributions ont été apportées à Csound. CsoundAC est une version étendue de Csound5.

Caractéristiques de Csound 5

Csound 5 débute une nouvelle version majeure de Csound qui inclut les nouvelles caractéristiques suivantes :

- Autorisé maintenant sous la GNU Lesser General Public License, une licence de code source libre (open source).
- Un nouveau système de construction, plus facile à mettre en œuvre, utilisant SCons.
- L'utilisation de bibliothèques de code source libre largement acceptées :
  - libsndfile pour les entrées et les sorties dans les fichiers son.
  - PortAudio avec les pilotes ASIO pour les entrées et les sorties en temps-réel à faible latence.
  - FLTK pour les contrôles graphiques que l'on peut programmer dans le code de l'orchestre.
  - PortMidi pour les entrées et les sorties MIDI en temps-réel.

  De plus, Istvan Varga a écrit des pilotes natifs MIDI et audio pour Windows et Linux.
- Un système simplifié de tampons audio.
- Des valeurs d'état retournées par toutes les fonctions internes, y compris les fonctions des opérateurs.
- Des opérateurs MIDI interopérables, ce qui permet d'utiliser les mêmes définitions d'instrument de façon interchangeable pour une exécution MIDI live ou une exécution différée commandée par une partition.
- Les opérateurs en plugin (module externe) sont opérationnels et sont acceptés plus largement. De nombreux opérateurs ont été déplacés dans des plugins. La plupart des nouveaux opérateurs sont des plugins, notamment :
  - Les opérateurs SoundFont basés sur FluidSynth.
  - Les opérateurs Python qui permettent d'exécuter du code Python dans l'en-tête d'un orchestre ou dans le code d'un instrument à cadence-`i` ou à cadence-`k`.
  - Les opérateurs Loris pour l'analyse temps/fréquence et la resynthèse.
  - Les opérateurs du bus de contrôle.
  - Les opérateur de mélangeur audio.
  - Les opérateurs de conversion de chaîne de caractères.
  - Les opérateurs Open Sound Control (OSC) améliorés.
  - Les opérateurs vectoriels.

  - Les opérateurs pvs pour le traitement fréquentiel du signal en temps-réel, un portage du code du vocodeur de phase de Mark Dolson.
  - Les opérateurs ATS pour l'analyse spectrale, la transformation, et la synthèse du son basée sur un modèle sinusoïdal avec bruit de bande critique. Un son dans ATS est un objet symbolique représentant un modèle spectral qu'on peut sculpter au moyen de diverses fonctions de transformation. Ces opérateurs peuvent lire, transformer et resynthétiser des fichiers d'analyse ATS. Il faut noter que l'application ATS est nécessaire pour produire les fichiers d'analyse.
  - Les opérateurs STK, constitués par les instruments du Synthesis Toolkit original de Perry Cook en C++, adaptés en opérateurs.
  - Les opérateurs d'adaptation DSSI et LADSPA pour accueillir des modules externes DSSI et LADSPA dans Csound.
  - Les opérateurs d'adaptation vst4csVST pour accueillir des modules externes VST dans Csound. (Distribués seulement sous la forme de sources à cause des restrictions de la licence du SDK de VST.)
- Le fichier d'en-tête `OpcodeBase.hpp` pour écrire des modules externes en C++. C'est basé sur la technique du polymorphisme statique via l'héritage de template.
- le frontal csound5gui d'Istvan Varga pour Csound, qui simplifie l'édition de Csound et son utilisation spécialement pour les exécutions en direct, et le suivi de contrôle des exécutions.
- Les frontaux en Tcl/Tk de Victor Lazzarini pour Csound, cstclsh et cswish.
- L'API de Csound devient plus normalisée et est plus largement utilisée. Il existe des interfaces encapsulant l'API dans les langages suivants :
  - C (include `csound.h`).
  - C++ (include `csound.hpp)`). Cette API contient les fonctions conteneur des fichiers de partition et d'orchestre de Csound.
  - Python (`import csnd`).
  - Java (`import csnd.*;`).
  - Lua (`require "csnd";`).
  - Lisp (utiliser le fichier CFFI `csound5.lisp`).
- Csound est maintenant totalement ré-entrant, ce qui veut dire que l'on peut exécuter plusieurs instances de Csound en même temps, dans le même processus.

John ffitch projette de remplacer l'analyseur syntaxique écrit à la main par un analyseur syntaxique produit à l'aide d'un générateur d'analyseur syntaxique, ce qui le rendrait moins sensible aux bogues et sans doute plus efficace.

Caractéristiques de CsoundAC

CsoundAC est un module d'extension Python pour écrire de la musique en programmant en Python. CsoundAC est basé sur le concept de graphes de musique par Michael Gogins, dans lequel une partition est représentée par un arbre hiérarchisé de nœuds, qui peuvent contenir des notes, des générateurs de partition, des transformations de partition, et d'autres nœuds.

CsoundAC fournit aussi une interface Python vers l'API de Csound. Grâce à celle-ci, il est très facile d'utiliser Csound pour exécuter les compositions en CsoundAC. Avec les triples guillemets de Python, il est même possible d'inclure le code de l'orchestre de Csound pour une pièce directement dans le code Python de cette pièce, si bien que toute la programmation pour une pièce peut être maintenue dans un seul fichier.

Le système de coordonnées dans CsoundAC est basé sur un espace musical euclidien ayant pour dimensions {temps, durée, type d'évènement, numéro d'instrument, hauteur comme numéro de touche MIDI, intensité comme vélocité MIDI, phase, coordonnée spatiale X, coordonnée spatiale Y, coordonnée spatiale Z, ensemble de classes de hauteur, 1}. Un point dans cet espace musical peut être une note, une inflection de note, ou même un grain sonore.

Un graphe de musique est un graphe orienté acyclique, ou un arbre, de nœuds dans l'espace musical. Ces nœuds sont associés avec des transformations locales du système de coordonnées. Il y a des nœuds pour contenir des partitions ou des fragments de partition, pour générer des partitions et pour transformer des partitions. De plus, chaque nœud peut contenir des nœuds enfants qui héritent du système de coordonnées du nœud parent.

Il est ainsi possible de composer une partition musicale en incluant ou en générant des notes dans les nœuds de niveau inférieur, puis en les assemblant dans une partition en utilisant des nœuds de niveau supérieur, et finalement en exécutant la partition avec Csound. Le procédé est strictement analogue à la construction d'une scène en 3 dimensions en synthèse graphique lorsque l'on génère des objets primitifs tels que sphères, cônes et cubes, puis qu'on les déplace dans l'espace pour assembler la scène.

Les classes de nœud dans CsoundAC sont :

- ScoreNode : contient simplement une séquence de notes ou d'autres points dans l'espace musical, peut-être importés d'un fichier MIDI.
- Rescale : met à l'échelle des points enfants pour les recadrer dans un intervalle donné de temps, durée, hauteur, et/ou d'autres dimensions.
- Cell : répète des points enfants en séquence à intervalles réguliers ; l'intervalle peut avoir une durée plus courte ou plus longue que celle des points enfants.
- Hocket : hoquet produit par des nœuds enfants.
- Lindenmayer : génère des partitions au moyen de systèmes de Lindenmayer O-L.
- StrangeAttractor : génère des partitions à partir de divers systèmes dynamiques chaotiques ajustables.
- MCRM : génère des partitions au moyen de l'algorithme de machine de copies multiples par réduction.
- ImageToScore : génère des partitions en transposant des fichiers image en points dans l'espace musical.
- Random : disperse des points enfants au hasard sur une ou plusieurs dimensions de l'espace musical, en utilisant diverses variables aléatoires.
- VoiceleadingNode : génère des progressions d'accords et des voix conductrices pour des notes enfants, au moyen d'opérations basées sur la théorie mathématique de la musique de Dmitri Tymoczko.

Enfin, le processus de composition peut inclure la dérivation d'une nouvelle classe Node en Python à partir d'un Node existant, afin de créer de nouveaux générateurs de partition et des transformations.

La commande Csound

Csound est une commande pour générer une sortie son à partir d'un fichier *orchestre* et d'un fichier *partition* (ou d'un *fichier csd* unifié). Il a été conçu pour être appelé depuis un terminal ou une fenêtre DOS, mais on peut l'appeler depuis un *frontal* plus facile à utiliser. Le fichier partition peut être codé dans un des différents formats, au choix de l'utilisateur. La traduction, le tri et le formatage de la partition dans un texte numérique lisible par l'orchestre sont effectués par différents préprocesseurs ; tout ou partie de la partition est ensuite envoyé à l'orchestre. L'exécution de l'orchestre est influencée par des *options de commande*, qui fixent le niveau des comptes-rendus graphiques et de console, spécifient les noms des fichiers d'E/S et les formats d'échantillonnage, et déclarent la nature de la détection et du contrôle en temps-réel.

Ordre de priorité

On peut fixer les options d'exécution de Csound en cinq endroits. Elles sont traitées dans l'ordre suivant :

1. Les valeurs par défaut de Csound
2. Le fichier défini par la *variable d'environnement* CSOUNDRC, ou le fichier .csoundrc dans le répertoire HOME
3. Le fichier *.csoundrc* dans le répertoire courant
4. La balise <CsOptions> dans un fichier *.csd*
5. En les passant sur la ligne de *commande* de Csound

Les dernières options dans la liste vont écraser les éventuelles options précédentes. A partir de la version 5.01 de Csound, les taux d'échantillonnage et de contrôle (options *-r* et *-k*) spécifiés n'importe où prévalent sur les valeurs sr, kr et ksmps définis dans l'en-tête de l'orchestre.

Description de la syntaxe de la commande

La commande csound est suivie par un ensemble d'*Options de Ligne de Commande* et par les noms des fichiers de l'orchestre (*.orc*) et de la partition (*.sco*) ou du *Fichier Unifié csd* (contenant à la fois l'orchestre et la partition) à traiter. Les *Options de Ligne de Commande* pour contrôler la configuration d'entrée et de sortie peuvent apparaître n'importe où dans la ligne de commande, séparées ou collées ensemble. Un drapeau nécessitant un Nom ou un Nombre le trouvera dans l'argument lui-même ou dans celui qui le suit immédiatement. Les commandes suivantes sont donc équivalentes :

```
csound -nm3 nomorchestre -Sxxnomfichier nompartition
csound -n -m 3 nomorchestre -x xnomfichier -S nompartition
```

Tous les drapeaux et les noms sont optionnels. Les valeurs par défaut sont :

```
csound -s -otest -b1024 -B1024 -m7 -P128 nomorchestre nompartition
```

où *nomorchestre* est un fichier contenant le code de l'orchestre Csound, et *nompartition* est un fichier de

données de partition en format de partition numérique standard, facultativement pré-trié et réajusté en temps. Si *nompartition* est omis, il y a deux options par défaut :

1. si l'on attend une entrée en temps-réel (par exemple *-L*, *-M*, *-iadc* ou *-F*), un fichier partition factice est utilisé, constitué de la seule instruction 'f 0 3600' (c'est-à-dire écouter sur l'entrée TR pendant une heure)
2. sinon Csound utilise le dernier *score.srt* produit dans le répertoire courant.

Csound rend compte des différentes étapes de traitement de la partition et de l'orchestre lors de l'exécution, effectuant différents tests de syntaxe et d'erreurs. Une fois l'exécution commencée, les messages d'erreur proviennent soit du chargeur d'instrument soit des générateurs unitaires eux-mêmes. Une commande Csound peut inclure toute combinaison d'options bien formée.

Exécuter les exemples du manuel à partir de la ligne de commande

La plupart des exemples du manuel sont prêts à l'emploi sans avoir besoin d'ajouter des options de ligne de commande, car ces options sont fixées dans la balise <CsOptions> du fichier csd, si bien qu'il suffit de taper une commande telle que :

```
csound oscil.csd
```

depuis le répertoire des exemples, et une sortie audio en temps-réel sera générée.

Csound command line

csound — Csound command.

Description

La commande *csound* exécute Csound.

Syntaxe

```
csound  [options]  [nomorch]  [nompartition]
```

```
csound  [options]  [nomfichiercsd]
```

Options de la ligne de commande de Csound

Ci-dessous la liste par ordre alphabétique des options de ligne de commande disponibles dans Csound 5. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options ! On peut consulter les options de ligne de commande par catégorie dans la section *Options de la Ligne de Commande (par Catégorie).*

Les arguments de la ligne de commande sont de 2 types : arguments *options* (commençant par « - », « -- » ou « -+ »), et arguments *nom* (tels que noms de fichier). Certains arguments option sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « -+ » prennent habituellement elles-mêmes un argument précédé du signe « = ».

Options de la Ligne de Commande

-@ FICHIER	Une ligne de commande étendue est fournie par le fichier « FICHIER »
-3, --format=24bit	Utiliser des échantillons audio de 24 bit.
-8, --format=uchar	Utiliser des échantillons audio en caractères non-signés sur 8 bit.
--format=*type*	Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf, flac, htk, ircam, mat4, mat5, nis, paf, pvf, raw, sd2, sds, svx, voc, w64, wav, wavex et xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.
-A, --aiff, --format=aiff	Ecrire un fichier son au format AIFF. A utiliser avec les options -*c*, -*s*, -*l*, ou -*f*.
-a, --format=alaw	Utiliser des échantillons audio a-law.
-B NUM, --hardwarebufsamps=NUM	Nombre de trames d'échantillonnage audio maintenues dans le tampon du *circuit* CNA. C'est une limite au-dessus de laquelle l'E/S audio *logicielle* va attendre avant de retourner. Une faible valeur réduit le délai audio d'E/S ; mais la valeur est souvent limi-

tée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps-réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 sur Linux, 4096 sur Mac OS X et 16384 sur Windows.

-b NUM, --iobufsamps=NUM

Nombre de trames d'échantillonnage audio dans chaque tampon *logiciel* d'E/S. De grandes valeurs conviennent, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des évènements en temps-réel. La valeur par défaut est 256 sur Linux, 1024 sur Mac OS X, et 4096 sur Windows. Lors d'une exécution en temps-réel, Csound attend les E/S audio toutes les *NUM* divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les *ksmps* divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est *ksmps * -NUM* (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.

Note : si l'on utilise en même temps -iadc et -odac (audio temps-réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de ksmps.

-C, --cscore

Utiliser le traitement par Cscore du fichier partition.

-c, --format=schar

Utiliser des échantillons audio en caractères signés sur 8 bit.

--csd-line-nums=NUM

Détermine comment les numéro de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fichiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures).

- 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.
- 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.

-D, --defer-gen1

Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.

-d, --nodisplays

Supprimer tous les affichages.

--displays

Autoriser les affichages, inversant l'effet d'une éventuelle option -d précédente.

--default-paths

Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option *--no-default-paths* précédente (par exemple dans *.csoundrc*).

--env:NOM=VALEUR

Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne avec INCDIR,

SADIR, SFDIR, et SSDIR.

--env:NOM+=VALEUR

Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, SFDIR, ou SSDIR). Si un fichier est trouvé dans plusieurs répertoires, c'est le dernier qui est utilisé.

--expression-opt

A partir de Csound 5. Activer certaines optimisations dans les expressions :

- Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne a1 = a2 + a3 sera compilée en a1 Add a2, a3 au lieu de #a0 Add a2, a3 a1 = #a0 évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec --expression-opt, mais cela dépend aussi largement du nombre d'expressions utilisées, du taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).

- le nombre de variables temporaires de taux a et de taux k est réduit significativement. L'expression

```
(a1 + a2 + a3 + a4)
```

  sera compilée en

```
#a0 Add a1, a2
#a0 Add #a0, a3
#a0 Add #a0, a4       ; (le résultat se trouve dans #a0)
```

  au lieu de

```
#a0 Add a1, a2
#a1 Add #a0, a3
#a2 Add #a1, a4       ; (le résultat se trouve dans #a2)
```

  Les avantages d'avoir moins de variables temporaires sont :

  - moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple ksmps = 100)

  - les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents

  - les erreurs de dépassement d'indice (par exemple quand des messages comme Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c) sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

  Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.

Avertissement

Lorsque --expression-opt est activé, il est interdit d'utiliser la fonction i() avec un argument expression, et il n'est pas prudent de compter au temps i sur la valeur de k-expressions.

-F FICHIER, --midifile=FICHIER
: Lire les évènements MIDI à partir du fichier *FICHIER*. Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.

-f, --format=float
: Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec *-i*, *soundin* et *GEN01*).

-G, --postscriptdisplay
: Supprimer les graphiques, une sortie graphique PostScript est produite à la place.

-g, --asciidisplay
: Supprimer les graphiques, une sortie pseudo-graphique ASCII est produite à la place.

-H#, --heartbeat=NUM
: Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son :

- pas de NUM, une barre tournante.
- NUM = 1, une barre tournante.
- NUM = 2, un point (.)
- NUM = 3, la taille du fichier en secondes.
- NUM = 4, un beep sonore.

-h, --noheader
: Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.

--help
: Afficher un message d'aide en ligne.

-I, --i-only
: *seulement au temps i.* Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps p (pas de k-signaux ni de a-signaux, et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des p-champs de la partition et des i-variables de l'orchestre. Cette option est mutuellement exclusive avec l'option --syntax-check-only.

-i FICHIER, --input=FICHIER
: Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle définie), enfin par SFDIR. Si le nom est *stdin*, la lecture audio se fera à partie de l'entrée standard.

Les noms *devaudio* ou *adc* provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : (par

	exemple -iadc3, -iadc:hw:1,1). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.
-+id_artist=chaîne	(longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)
-+id_comment=chaîne	(longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)
-+id_copyright=chaîne	(longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)
-+id_date=chaîne	(longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)
-+id_software=chaîne	(longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)
-+id_title=chaîne	(longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)
-+ignore_csopts=entier	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section CsOptions du fichier csd. Voir *Format de Fichier Unifié pour les Orchestres et les Partitions*.
-+input_stream=chaîne	Nom du flot d'entrée pulseaudio.
-J, --ircam, --format=ircam	Ecrire un fichier son dans le format de l'IRCAM.
-+jack_client=[nom_client]	Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)
-+jack_inportname=[préfixe du nom du port d'entrée], -+jack_outportname=[préfixe du nom du port de sortie]	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement) Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :

```
csound5:input1              (enregistrement gauche)
csound5:input2              (enregistrement droite)
csound5:output1             (reproduction gauche)
csound5:output2             (reproduction droite)
```

-K, --nopeaks	Ne générer aucun bloc PEAK.
-k NUM, --control-rate=NUM	Remplacer le taux de contrôle (*kr*) fourni par l'orchestre.
-L PERIPHERIQUE, --score-in=PERIPHERIQUE	Lire en temps-réel des évènements de partition en ligne de texte à partir du périphérique *PERIPHERIQUE*. Le nom *stdin* permettra de recevoir les évènements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les évène-

ments sont codés de la même manière que ceux de la *partition numérique standard*, sauf qu'un évènement avec p2=0 sera exécuté immédiatement, et qu'un évènement avec p2=T sera exécuté T secondes après son arrivée. Les évènements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction *carry (report de valeur)* de la partition est autorisée ici, ainsi que les notes liées (p3 négatif) at les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.

-l, --format=long
: Utiliser des échantillons audio codés en entiers longs.

-M PERIPHERIQUE, --midi-device=PERIPHERIQUE
: Lire les évènements MIDI à partir du périphérique *PERIPHERIQUE*. Si l'on utilise ALSA MIDI (-+rtmidi=alsa), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est en-dehors de l'intervalle permis, une erreur est levée et les numéros de périphérique valides sont imprimés.

-m NUM, --messagelevel=NUM
: Niveau des messages pour la sortie standard (terminal). Prend la *somme* de n'importe lesquelles de ces valeurs :

- 1 = messages d'amplitude de note
- 2 = message d'échantillons hors intervalle
- 4 = messages d'avertissement
- 128 = impression d'information de tests de référence

Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :

- 0 = amplitudes brutes, pas de couleur
- 32 = dB, pas de couleur
- 64 = dB, hors intervalle colorées en rouge
- 96 = dB, toutes colorées
- 256 = brutes, hors intervalle colorées en rouge
- 512 = brutes, toutes colorées

La valeur par défaut est 135 (128+4+2+1), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La mise en couleur des amplitudes brutes fut introduite dans la version 5.04.

--m-amps=NUM
: Niveau des messages d'amplitudes sur la sortie standard (terminal).

- 0 = pas de messages d'amplitude de note
- 1 = messages d'amplitude de note

--m-range=NUM
: Niveau des messages de dépassement de limite sur la sortie stan-

dard (terminal).

- 0 = aucun message d'échantillon hors limites
- 1 = messages d'échantillons hors limites

--m-warnings=NUM

Niveau des messages d'avertissement sur la sortie standard (terminal).

- 0 = pas de messages d'avertissement
- 1 = messages d'avertissement

--m-dB=NUM

Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).

- 0 = messages d'amplitude absolue
- 1 = messages d'amplitude en dB

--m-colours=NUM

Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).

- 0 = pas de coloration des messages d'amplitude
- 1 = coloration des messages d'amplitude

--m-benchmarks=NUM

Niveau des messages d'information de test de performance sur la sortie standard (terminal).

- 0 = pas de nombres de test de performance
- 1 = nombres de test de performance

-+max_str_len=entier

(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.

--midi-key=N

Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].

--midi-key-cps=N

Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.

--midi-key-oct=N

Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.

--midi-key-pch=N

Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).

--midi-velocity=N

Transmettre la vélocité d'un message MIDI note on au p-champ N en valeur MIDI [0-127].

--midi-velocity-amp=N

Transmettre la vélocité d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].

--midioutfile=NOMFICHIER

Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).

-+msg_color=booléen	Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.
-+mute_tracks=chaîne	(longueur max. = 255 caractères) Ignorer les évènements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par example, -+mute_tracks=00101 désactivera la troisième et la cinquième pistes).
-N, --notify	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
-n, --nosound	Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.
--no-default-paths	Désactiver l'addition de répertoire de CSD/ORC/SCO au chemin de recherche.
--no-expression-opt	Désactiver l'optimisation des expressions.
-O FICHIER, --logfile=FICHIER	Compte-rendu dans le fichier *FICHIER*. Si *FICHIER* est null (c-à-d *-O null* ou *--logfile=null*) toutes les impressions de message sur la console sont désactivées.
-o FICHIER, --output=FICHIER	Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement SFDIR (si elle est définie), sinon dans le répertoire courant. Le nom *stdout* provoque l'écriture audio sur la sortie standard, tandis qu'avec *null* il n'y a aucun son en sortie comme pour l'option -n. Si aucun nom n'est donné, le nom par défaut sera *test*. Les noms *devaudio* ou *dac* (on peut utiliser *-odac* ou *-o dac*) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple -odac3, -odac:hw:1,1). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.
--omacro:XXX=YYY	Donner la valeur YYY à la macro d'orchestre XXX
-+output_stream=chaîne	Nom du flot de sortie pulseaudio.
-Q PERIPHERIQUE	Activer les opérations MIDI OUT vers le périphérique d'id *PERIPHERIQUE*. Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquencement temps-réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option *-b*. Si l'on utilise ALSA MIDI (-+rtmidi=alsa), les périphériques sont sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme -Q hw:CARTE,PERIPHERIQUE où

	CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -Q hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.
-R, --rewrite	Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
-r NUM, --sample-rate=NUM	Remplacer le taux d'échantillonage (*sr*) fourni par l'orchestre.
-+raw_controller_mode=booléen	Désactiver le traitement spécial des contrôleurs MIDI tels que sustain, pédale, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
-+rtaudio=chaîne	(longueur max. = 20 caractères) Nom du module audio temps-réel. La valeur par défaut est PortAudio. Sont disponibles selon la plate-forme et les options de construction : Linux : alsa, jack; Windows : mme; Mac OS X : CoreAudio. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin audio temps-réel.
-+rtmidi=chaîne	(longueur max. = 20 caractères) Nom du module MIDI temps-réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin MIDI temps-réel. Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0).
-s, --format=short	Utiliser des échantillons audio codés par des entiers courts.
--sched	*Seulement sur linux.* Utiliser pour le temps-réel le temps-partagé et le verrouillage de la mémoire. (nécessite également *-d* et *-o dac* ou *-o devaudio*). Voir aussi --sched=N ci-dessous.
--sched=N	*Seulement sur linux.* Identique à --sched, mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de temps-partagé SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" (nice) à N. On peut aussi l'utiliser avec le format --sched=N,MAXCPU,TEMPS pour autoriser l'utilisation d'un processus léger (thread) de contrôle qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).
-+server=chaîne	Nom du serveur pulseaudio.
-+skip_seconds=float	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les évènements antérieurs de la partition ou du fichier MIDI.

--smacro:XXX=YYY	Donner la valeur YYY à la macro de partition XXX
--strset	*Csound 5.* L'option --strset permet de passer des chaînes à strset pour les lier à des valeurs numériques depuis la ligne de commande, dans le format '--strsetN=VALEUR'. Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
--syntax-check-only	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option --i-only. (Csound 5.08 et versions ultérieures).
-T, --terminate-on-midi	Terminer l'exécution quand la fin du fichier MIDI est atteinte.
-t0, --keep-sorted-score	Empêcher Csound d'effacer le fichier de la partition triée, *score.srt*, lors de la sortie.
-t NUM, --tempo=NUM	Utiliser les pulsations non interprétées de *score.srt* pour cette exécution, et fixer le tempo initial à *NUM* pulsations par minute. Quand ce drapeau est positionné, le tempo de l'exécution de la partition est aussi contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.
-U UTILITE, --utility=UTILITE	Invoquer le programme utilitaire *UTILITE*. En donnant un nom invalide on obtient une liste des utilitaires.
-u, --format=ulaw	Utiliser des échantillons audio u-law.
-v, --verbose	Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.
-W, --wave, --format=wave	Ecrire un fichier son au format WAV.
-x FICHIER, --extract-score=FICHIER	Extraire un morceau de la partition triée, *score.srt*, en utilisant le fichier d'extraction *FICHIER* (voir *Extract*).
-Z, --dither	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.
-Z, --dither--triangular, --dither--uniform	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de -Z le chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.
-z NUM, --list-opcodesNUM	Lister les opcodes de cette version : • pas de NUM, montrer seulement les noms • NUM = 0, montrer seulement les noms • NUM = 1, montrer les arguments de chaque opcode dans le format <nomop> <argssortie> <argsentrée>

Options de Ligne de Commande (par Catégo-

rie)

Ci-dessous la liste par catégorie des options de ligne de commande disponibles dans Csound 5. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options !

On peut consulter les options de ligne de commande par ordre alphabétique dans la section *Options de Ligne de Commande (par Ordre Alphabétique).*

Le format d'une commande est soit :

`csound` [`options`] [nomorchestre] [nompartition]
soit

`csound` [`options`] [nomfichiercsd]

où les arguments sont de 2 sortes : arguments *options* (commençant par « - », « -- » ou « -+ »), et arguments *nom* (tels que noms de fichier). Certains arguments options sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « -+ » prennent habituellement un argument précédé du signe « = ».

Sortie dans un Fichier Audio

-3, --format=24bit	Utiliser des échantillons audio de 24 bit.
-8, --format=uchar	Utiliser des échantillons audio en caractères non-signés sur 8 bit.
-A, --aiff, --format=aiff	Ecrire un fichier son au format AIFF. A utiliser avec les options -*c*, *-s*, *-l*, ou *-f*.
-a, --format=alaw	Utiliser des échantillons audio a-law.
-c, --format=schar	Utiliser des échantillons audio en caractères signés sur 8 bit.
-f, --format=float	Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec *-i*, *soundin* et *GEN01*).
--format=*type*	Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf, flac, htk, ircam, mat4, mat5, nis, paf, pvf, raw, sd2, sds, svx, voc, w64, wav, wavex et xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.
-h, --noheader	Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.
-i FICHIER, --input=FICHIER	Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle définie), enfin par SFDIR. Si le nom est *stdin*, la lecture audio se fera à partie de l'entrée standard. Les noms *devaudio* ou *adc* provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et

1023, ou un nom de périphérique séparé par un caractère : . L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.

-J, --ircam, --format=ircam — Ecrire un fichier son dans le format de l'IRCAM.

-K, --nopeaks — Ne générer aucun bloc PEAK.

-l, --format=long — Utiliser des échantillons audio codés en entiers longs.

-n, --nosound — Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.

-o FICHIER, --output=FICHIER — Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement SFDIR (si elle est définie), sinon dans le répertoire courant. Le nom *stdout* provoque l'écriture audio sur la sortie standard, tandis qu'avec *null* il n'y a aucun son en sortie comme pour l'option -n. Si aucun nom n'est donné, le nom par défaut sera *test*.

Les noms *dac* ou *devaudio* (on peut utiliser *-odac* ou *-o dac*) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : . Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.

-R, --rewrite — Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).

-s, --format=short — Utiliser des échantillons audio codés par des entiers courts.

-u, --format=ulaw — Utiliser des échantillons audio u-law.

-W, --wave, --format=wave — Ecrire un fichier son au format WAV.

-Z, --dither — Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.

-Z, --dither--triangular, --dither--uniform — Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de -Z le chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.

Champs du Fichier de Sortie

-+id_artist=chaîne — (longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)

-+id_comment=chaîne — (longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)

-+id_copyright=chaîne — (longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)

-+id_date=chaîne — (longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)

-+id_software=chaîne — (longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)

-+id_title=chaîne — (longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)

Entrée/Sortie Audio en Temps-Réel

-i adc[PERIPHERIQUE], --input=adc[PERIPHERIQUE] — Les noms *devaudio* ou *adc* provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : (par exemple -iadc3, -iadc:hw:1,1). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.

-o dac[PERIPHERIQUE], --output=dac[PERIPHERIQUE] — Les noms *dac* ou *devaudio* (on peut utiliser *-odac* ou *-o dac*) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple -odac3, -odac:hw:1,1). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.

-+rtaudio=chaîne — (longueur max. = 20 caractères) Nom du module audio temps-réel. La valeur par défaut est PortAudio (sur toutes les plates-formes). Sont également disponibles selon la plate-forme et les options de construction : Linux : alsa, jack; Windows : mme; Mac OS X : CoreAudio. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin audio temps-réel.

-+server=chaîne — Nom du serveur pulseaudio.

-+output_stream=chaîne — Nom du flot de sortie pulseaudio.

-+input_stream=chaîne — Nom du flot d'entrée pulseaudio.

-+jack_client=[nom_client] — Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)

-+jack_inportname=[préfixe du nom du port d'entrée], -+jack_outportname=[préfixe du nom du port de sortie]	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement) Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :

```
csound5:input1              (enregistrement gauche)
csound5:input2              (enregistrement droite)
csound5:output1             (reproduction gauche)
csound5:output2             (reproduction droite)
```

Entrée/Sortie par fichier MIDI

-F FICHIER, --midifile=FICHIER	Lire les évènements MIDI à partir du fichier *FICHIER*. Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.
--midioutfile=NOMFICHIER	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
-+mute_tracks=chaîne	(longueur max. = 255 caractères) Ignorer les évènements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par example, -+mute_tracks=00101 désactivera la troisième et la cinquième pistes).
-+raw_controller_mode=booléen	Désactiver le traitement spécial des contrôleurs MIDI tels que pédale de sustain, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
-+skip_seconds=float	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les évènements antérieurs de la partition ou du fichier MIDI.
-T, --terminate-on-midi	Terminer l'exécution quand la fin du fichier MIDI est atteinte.

Entrée/Sortie MIDI en Temps-Réel

-M PERIPHERIQUE, --midi-device=PERIPHERIQUE	Lire les évènements MIDI à partir du périphérique *PERIPHERIQUE*. Si l'on utilise ALSA MIDI (*-+rtmidi*=alsa), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est en-dehors de l'intervalle permis, une erreur est levée et les numéros de périphérique valides sont imprimés.

--midi-key=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
--midi-key-cps=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.
--midi-key-oct=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.
--midi-key-pch=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).
--midi-velocity=N	Transmettre la vélocité d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
--midi-velocity-amp=N	Transmettre la vélocité d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].
--midioutfile=NOMFICHIER	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
-+rtmidi=chaîne	(longueur max. = 20 caractères) Nom du module MIDI temps-réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin MIDI temps-réel. Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0).
-Q PERIPHERIQUE	Activer les opérations MIDI OUT vers le périphérique d'id *PERIPHERIQUE*. Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquencement temps-réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option *-b*. Si l'on utilise ALSA MIDI (*-+rtmidi*=alsa), les périphériques sont sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme -Q hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -Q hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.

Affichage

--csd-line-nums=NUM	Détermine comment les numéro de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fi-

chiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures).

- 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.
- 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.

-d, --nodisplays
: Supprimer tous les affichages.

--displays
: Autoriser les affichages, inversant l'effet d'une éventuelle option -d précédente.

-G, --postscriptdisplay
: Supprimer les graphiques, une sortie graphique PostScript est produite à la place.

-g, --asciidisplay
: Supprimer les graphiques, une sortie pseudo-graphique ASCII étant produite à la place.

-H#, --heartbeat=NUM
: Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son :

- pas de NUM, une barre tournante.
- NUM = 1, une barre tournante.
- NUM = 2, un point (.)
- NUM = 3, la taille du fichier en secondes.
- NUM = 4, un beep sonore.

-m NUM, --messagelevel=NUM
: Niveau des messages pour la sortie standard (terminal). Prend la *somme* de n'importe lesquelles de ces valeurs :

- 1 = messages d'amplitude de note
- 2 = message d'échantillons hors intervalle
- 4 = messages d'avertissement
- 128 = impression d'information de tests de référence

Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :

- 0 = amplitudes brutes, pas de couleur
- 32 = dB, pas de couleur
- 64 = dB, hors intervalle colorées en rouge
- 96 = dB, toutes colorées
- 256 = brutes, hors intervalle colorées en rouge
- 512 = brutes, toutes colorées

La valeur par défaut est 135 (128+4+2+1), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La coloration des amplitudes brutes

fut introduite dans la version 5.04.

--m-amps=NUM
: Niveau des messages d'amplitudes sur la sortie standard (terminal).

- 0 = pas de messages d'amplitude de note
- 1 = messages d'amplitude de note

--m-range=NUM
: Niveau des messages de dépassement de limite sur la sortie standard (terminal).

- 0 = aucun message d'échantillon hors limites
- 1 = messages d'échantillons hors limites

--m-warnings=NUM
: Niveau des messages d'avertissement sur la sortie standard (terminal).

- 0 = pas de messages d'avertissement
- 1 = messages d'avertissement

--m-dB=NUM
: Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).

- 0 = messages d'amplitude absolue
- 1 = messages d'amplitude en dB

--m-colours=NUM
: Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).

- 0 = pas de coloration des messages d'amplitude
- 1 = coloration des messages d'amplitude

--m-benchmarks=NUM
: Niveau des messages d'information de test de performance sur la sortie standard (terminal).

- 0 = pas de nombres de test de performance
- 1 = nombres de test de performance

-+msg_color=booléen
: Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.

-v, --verbose
: Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.

-z NUM, --list-opcodesNUM
: Lister les opcodes de cette version :

- pas de NUM, montrer seulement les noms
- NUM = 0, montrer seulement les noms
- NUM = 1, montrer les arguments de chaque opcode dans le for-

mat <nomop> <argssortie> <argsentrée>

Configuration et Contrôle de l'Exécution

-B NUM, --hardwarebufsamps=NUM

Nombre de trames d'échantillonnage audio maintenues dans le tampon du *circuit* CNA. C'est une limite au-dessus de laquelle l'E/S audio *logicielle* va attendre avant de retourner. Une faible valeur réduit le délai audio d'E/S ; mais la valeur est souvent limitée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps-réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 dans Linux, 4096 dans Mac OS X et 16384 dans Windows.

-b NUM, --iobufsamps=NUM

Nombre de trames d'échantillonnage audio dans chaque tampon *logiciel* d'E/S. De grandes valeurs fonctionnent bien, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des évènements en temps-réel. La valeur par défaut est 256 dans Linux, 1024 dans Mac OS X, et 4096 dans Windows. Lors d'une exécution en temps-réel, Csound attend les E/S audio toutes les *NUM* divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les *ksmps* divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est *ksmps * -NUM* (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.

Note : si l'on utilise en même temps -iadc et -odac (audio temps-réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de ksmps.

-k NUM, --control-rate=NUM

Remplacer le taux de contrôle (*kr*) fourni par l'orchestre.

-L PERIPHERIQUE, --score-in=PERIPHERIQUE

Lire en temps-réel des évènements de partition en ligne de texte à partir du périphérique *PERIPHERIQUE*. Le nom *stdin* permettra de recevoir les évènements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les évènements sont codés de la même manière que ceux de la *partition numérique standard*, sauf qu'un évènement avec p2=0 sera exécuté immédiatement, et qu'un évènement avec p2=T sera exécuté T secondes après son arrivée. Les évènements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction *carry (report de valeur)* de la partition est autorisée ici, ainsi que les notes liées (p3 négatif) at les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.

--omacro:XXX=YYY

Donner la valeur YYY à la macro d'orchestre XXX

-r NUM, --sample-rate=NUM

Remplacer le taux d'échantillonage (*sr*) fourni par l'orchestre.

--sched	*Seulement dans linux.* Utiliser la planification du temps-réel et le verrouillage de la mémoire. (nécessite également *-d* et *-o dac* ou *-o devaudio*). Voir aussi --sched=N ci-dessous.
--sched=N	*Seulement sur linux.* Identique à --sched, mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de planification SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" (nice) à N. On peut aussi l'utiliser avec le format --sched=N,MAXCPU,TEMPS pour autoriser l'utilisation d'un processus léger (thread) de surveillance qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).
--smacro:XXX=YYY	Donner la valeur YYY à la macro de partition XXX
--strset	*Csound 5.* L'option --strset permet de passer des chaînes de caractères à strset depuis la ligne de commande, dans le format '--strsetN=VALEUR'. Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
-+skip_seconds=float	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les évènements antérieurs de la partition ou du fichier MIDI.
-t NUM, --tempo=NUM	Utiliser les pulsations non interprétées de *score.srt* pour cette exécution, et fixer le tempo initial à *NUM* pulsations par minute. Quand cette options est positionnée, le tempo de l'exécution de la partition est également contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.

Divers

-@ FICHIER	Une ligne de commande étendue est fournie dans le fichier « FICHIER »
-C, --cscore	Utiliser le traitement par Cscore du fichier partition.
--default-paths	Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option *--no-default-paths* précédente (par exemple dans *.csoundrc*).
-D, --defer-gen1	Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.
--env:NOM=VALEUR	Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne avec INCDIR, SADIR, SFDIR, et SSDIR.
--env:NOM+=VALEUR	Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, SFDIR, ou SSDIR). Si un fichier est trouvé

dans plusieurs répertoires, c'est le dernier qui est utilisé.

--expression-opt

A partir de Csound 5. Activer certaines optimisations dans les expressions :

- Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne a1 = a2 + a3 sera compilée en a1 Add a2, a3 au lieu de #a0 Add a2, a3 a1 = #a0 évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec --expression-opt, mais cela dépend aussi largement du nombre d'expressions utilisées, du taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).

- le nombre de variables temporaires de taux a et de taux k est réduit significativement. L'expression

  ```
  (a1 + a2 + a3 + a4)
  ```

  sera compilée en

  ```
  #a0 Add a1, a2
  #a0 Add #a0, a3
  #a0 Add #a0, a4       ; (le résultat se trouve dans #a0)
  ```

  au lieu de

  ```
  #a0 Add a1, a2
  #a1 Add #a0, a3
  #a2 Add #a1, a4       ; (le résultat se trouve dans #a2)
  ```

  Les avantages d'avoir moins de variables temporaires sont :

  - moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple ksmps = 100)

  - les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents

  - les erreurs de dépassement d'indice (par exemple quand des messages comme Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c) sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

  Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.

Avertissement

Lorsque --expression-opt est activé, il est interdit d'utiliser la fonction i() avec un argument expres-

sion, et il n'est pas prudent de compter au temps i sur la valeur de k-expressions.

--help	Afficher un message d'aide en ligne.
-I, --i-only	*seulement au temps i.* Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps p (pas de k-signaux ni de a-signaux, et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des p-champs de la partition et des i-variables de l'orchestre. Cette option est mutuellement exclusive avec l'option --syntax-check-only.
-+ignore_csopts=entier	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section CsOptions du fichier csd. Voir *Format de Fichier Unifié pour les Orchestres et les Partitions*.
-+max_str_len=entier	(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.
-N, --notify	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
--no-default-paths	Désactiver l'addition de répertoire de CSD/ORC/SCO au chemin de recherche.
--no-expression-opt	Désactiver l'optimisation des expressions.
-O FICHIER, --logfile=FICHIER	Compte-rendu dans le fichier *FICHIER*. Si *FICHIER* est null (c-à-d. *-O null* ou *--logfile=null*) toutes les impressions de message sur la console sont désactivées.
--syntax-check-only	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option --i-only. (Csound 5.08 et versions ultérieures).
-t0, --keep-sorted-score	Empêche Csound d'effacer le fichier de la partition triée, *score.srt*, lors de la sortie.
-U UTILITE, --utility=UTILITE	Invoquer le programme utilitaire *UTILITE*. En donnant un nom invalide on obtient une liste des utilitaires.
-x FICHIER, --extract-score=FICHIER	Extraire un morceau de la partition triée, *score.srt*, en utilisant le fichier d'extraction *FICHIER* (voir *Extract*).

Variables d'Environnement de Csound

Csound peut utiliser les variables d'environnement suivantes :

- SFDIR : Répertoire par défaut pour les fichiers son. Utilisé si aucun chemin complet n'est fourni pour

les fichiers son.

- SSDIR : Répertoire par défaut pour les fichiers audio et MIDI en entrée (source). Utilisé si aucun chemin complet n'est fourni pour les fichiers son. On peut l'utiliser conjointement avec SFDIR pour fixer des répertoire d'entrée et de sortie séparés. Prière de noter qu'aussi bien les fichiers MIDI que les fichiers audio sont recherchés aussi dans SSDIR.

- SADIR : Répertoire par défaut pour les fichier d'analyse. Utilisé si aucun chemin complet n'est donné pour les fichiers d'analyse.

- SFOUTYP : Fixe le type par défaut du fichier de sortie. Actuellement ne sont valides que 'WAV', 'AIFF' et 'IRCAM'. Cette variable est testée par l'exécutable de csound et par les utilitaires et elle est utilisée si aucun type de fichier de sortie n'a été spécifié.

- INCDIR : Répertoire des fichiers à inclure. Spécifie l'endroit où se trouvent les fichiers utilisés par les instructions *#include*.

- OPCODEDIR : Définit l'endroit où se trouvent les plugins d'opcode en version simple précision (32 bit).

- OPCODEDIR64 : Définit l'endroit où se trouvent les plugins d'opcode en version double précision (64 bit).

- SNAPDIR : Utilisée par les opcodes de contrôle graphique FLTK pour charger et sauvegarder les instantanés.

- CSOUNDRC : Définit le fichier de ressource (ou de configuration) de csound. Un chemin complet avec le nom d'un fichier contenant des options de csound doit être donné. Cette variable vaut .csoundrc par défaut.

- CSSTRNGS : A partir de Csound 5.00, la localisation des messages est contrôlée par les deux variables d'environnement CSSTRNGS et CS_LANG, qui sont toutes deux optionnelles. CSSTRNGS pointe vers un répertoire contenant des fichiers .xmg.

- CS_LANG : Sélectionne une langue pour les messages de csound.

- RAWWAVE_PATH : Utilisée par les opcodes STK pour trouver les fichiers son bruts. Ne sert que si vous utilisez des opcodes de sur-couche STK comme STKBowed ou STKBrass.

- CSNOSTOP : Si cette variable d'environnement a pour valeur "yes", alors tous les affichages graphiques sont fermés à la fin de l'exécution (ce qui veut dire que vous n'en verrez peut-être pas grand chose dans le cas d'une exécution courte en temps différé). Dans le cas contraire, il faut cliquer sur "Quit" dans la fenêtre d'affichage FLTK pour sortir, ce qui permet de voir les graphiques même après que la fin de la partition soit atteinte.

- MFDIR : Répertoire par défaut pour les fichiers MIDI. Utilisé si aucun chemin complet n'est donné pour les fichiers MIDI. Prière de noter que les fichiers MIDI sont également recherchés dans SSDIR et SFDIR.

Pour plus d'information sur SFDIR, SSDIR, SADIR, MFDIR et INCDIR voir *Répertoires et Fichiers*.

Les seules variables d'environnement obligatoires sont OPCODEDIR et OPCODEDIR64. Il est très important de les remplir correctement, sinon la plupart des opcodes ne seront pas disponibles. Assurez-vous de fixer le chemin correctement en fonction de la précision de votre exécutable. Si vous lancez csound en ligne de commande sans aucun argument vous devriez voir un texte ressemblant à : Csound version 5.01.0 beta (float samples) Mar 23 2006. Ce texte fait référence à la version simple précision.

CSSTRNGS et CS_LANG sont actuellement peu utiles car Csound n'a pas encore été complètement traduit dans d'autres langues.

Voici d'autres variables d'environnement qui ne sont pas propres à Csound mais qui peuvent être importantes :

- *PATH* : Le répertoire contenant les exécutables de csound devrait être listé dans cette variable.
- *PYTHONPATH* : Si vous avez l'intention d'utiliser CsoundVST et python, le répertoire contenant la bibliothèque partagée _CsoundVST et le fichier CsoundVST.py doit être dans votre variable d'environnement *PYTHONPATH* (ou le chemin de recherche par défaut de python), afin que l'interpréteur Python sache comment charger ces fichiers.
- *LADSPA_PATH et DSSI_PATH* : Ces variables d'environnement sont nécessaires si vous utilisez les opcodes du plugin *dssi4cs* (hôtes LADSPA et DSSI).
- *CSDOCDIR* : Spécifie le répertoire dans lequel se trouvent les fichiers d'aide html. Bien qu'elle ne soit pas utilisée directement par Csound, cette variable d'environnement peut aider les frontaux et les éditeurs (qui la mettent en œuvre) à trouver le manuel de csound.

Fixer les variables d'environnement

Sur la ligne de commande

On peut fixer les variables d'environnement sur la ligne de commande ou dans le fichier de configuration .csoundrc en utilisant l'option de ligne de commande --env:NOM=VALEUR ou --env:NOM+=VALEUR, où NOM est le nom de la variable d'environnement, et VALEUR est sa valeur. Voir *Options de Ligne de Commande*.

Note

Prière de noter que cette méthode ne fonctionnera pas pour les variables d'environnement qui sont lues avant les arguments de la ligne de commande. Pour SADIR, SSDIR, SFDIR, INCDIR, SNAPDIR, RAWWAVE_PATH, CSNOSTOP, SFOUTYP cela devrait marcher, mais les variables d'environnement suivantes doivent être fixées dans le système avant de lancer csound : OPCODEDIR, OPCODEDIR64, CSSTRINGS, et CS_LANG. Actuellement (v. 5.02) CSOUNDRC peut être fixée par --env, mais cette possibilité n'est pas garantie dans les versions futures.

Windows

Pour fixer une variable d'environnement dans Windows XP et 2000 aller dans Panneau de Contrôle->Système->Avancé et cliquer sur le bouton 'Variables d'environnement'. Dans les autres versions de Windows antérieures à Windows XP et Windows 2000 on fixe les variables d'environnement dans le fichier autoexec.bat. Aller dans 'Poste de travail', choisir le lecteur C:, cliquer avec le bouton droit sur autoexec.bat, et choisir 'Edition'. Le format de l'instruction est : SET NOM=VALEUR.

Linux

Il y a plusieurs manières de fixer les variables d'environnement sur linux. On peut les initialiser avec la commande de shell *export*, dans le fichier .bashrc ou des fichiers similaires ou en les ajoutant au fichier /etc/profile.

Mac

Si l'on a un Mac avec une version d'OS X inférieure à la 10.3 (y compris 10.2 et 10.1) il est alors pos-

sible que le shell par défaut soit le Tenex C-shell (tcsh). Si c'est le cas, il faut alors taper :

```
~% setenv OPCODEDIR "/Users/you/your/Csound5/build"
```

ou changer votre fichier /etc/profile et/ou modifier votre fichier .tcshrc.

Si l'on a un Mac avec OS X 10.3 ou 10.4 alors il y a certainement un shell C "Bourne-again" (bash) par défaut. Si c'est le cas, alors il faut taper quelque chose comme ça :

```
~$ export OPCODEDIR=/Users/you/your/Csound5/build
```

De plus si l'on a un bash shell par défaut, alors il est plus facile de modifier le fichier .bashrc ou le fichier /etc/profile.

A noter que si l'on choisit l'une des méthodes ci-dessus, par exemple modifier le fichier .bashrc, alors les variables d'environnement sont allouées quand un nouveau shell est créé. Ceci peut poser un problème lorque votre application implémente une interface Quartz ou Aqua et n'utilise pas la ligne de commande.

Si c'est le cas, la solution standard (jusqu'à OS 10.3.9 et à moins que l'application utilise l'API de csound et fixe directement les variables d'environnement) consiste à créer un fichier contenant une liste de propriétés XML (un fichier nommé .plist par l'OS). Ce fichier devrait se trouver dans ~/.MacOSX/Environment.plist. Cette solution a été utilisée spécifiquement pour l'objet [csoundapi~] pour Pd sur OS X. Comme Pd utilise un style de paquetage .app natif OS X, et s'exécute en dehors de l'interface Aqua, les moyens standard de fournir les variables d'environnement à Csound ne fonctionnent pas. La solution consiste à fixer les variables d'environnement de Csound pour l'environnement Aqua.

Il est probable que la plupart des utilisateurs n'auront pas de répertoire caché .MaxOSX dans leur répertoire $HOME (alias ~/). Il faut d'abord créer ce répertoire et y ajouter Environment.plist. Le contenu du fichier Environment.plist ressemblera à ceci :

```
<?xml version="1.0" encoding='UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>OPCODEDIR</key>
<string>/Library/Frameworks/CsoundLib.framework/Versions/5.1/Resources/Opcodes</string>
<key>OPCODEDIR64</key>
<string>/Volumes/ExternalHD/devel/csound5/lib64</string>
<key>INCDIR</key>
<string>/Volumes/ExternalHD/CSOUND/include</string>
<key>SFDIR</key>
<string>/Volumes/ExternalHD/iTunes/csoundaudio</string>
</dict>
</plist>
```

et ainsi de suite, en utilisant la balise XML <key> pour chaque variable d'environnement requise par l'API et la balise <string> pour le chemin correspondant dans le système.

Prière de noter qu'il faut se déconnecter et se reconnecter (login) pour que ces changements prennent effet.

Format de Fichier Unifié pour les Orchestres et les Partitions

Description

Le Format de Fichier Unifié, introduit à partir de la version 3.50 de Csound, permet de combiner dans le

même fichier l'orchestre et la partition, ainsi que les options de ligne de commande. Le fichier a pour extension *.csd*. Ce format fut introduit à l'origine par Michael Gogins dans AXCsound.

Le fichier est un fichier de données structurées qui utilise un langage de balises, de la famille SGML comme HTML. Une balise ouvrante (<*balise*>) et une balise fermante (</*balise*>) servent à délimiter les différents éléments. Ce fichier est sauvegardé comme un fichier texte.

Format du Fichier de Données Structurées

Eléments Obligatoires

la première balise du fichier doit être la balise ouvrante <*CsoundSynthesizer*>. La dernière balise du fichier doit être la balise fermante </*CsoundSynthesizer*>. Cet élément sert à avertir le compilateur csound du format *.csd*. Tout texte situé avant la balise de début et après la balise de fin est ignoré par Csound. Cette balise peut aussi s'écrire <*CsoundSynthesiser*>.

Options (<CsOptions>)

Les *options de ligne de commande* de Csound sont insérées dans l'Elément Options. La section est délimitée par la balise ouvrante <*CsOptions*> et par la balise fermante </*CsOptions*> . Les lignes commençant par # ou par *;* sont traitées comme des commentaires.

Orchestre (<CsInstruments>)

Les définitions d'instruments (orchestre) sont mises dans l'Elément Instruments. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier orchestre* de Csound, et répondent aux mêmes besoins, y compris les instructions d'en-tête (*sr*, *kr*, etc). Cet Elément Instruments est délimité par la balise ouvrante <*CsInstruments*> et par la balise fermante </*CsInstruments*>.

Partition (<CsScore>)

Les instructions de la partition Csound sont mises dans l'Elément Score. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier partition* de Csound, et répondent aux mêmes besoins. L'Elément Score est délimité par la balise ouvrante <*CsScore*> et par la balise fermante </*CsScore*>.

Eléments Optionnels

Inclusion de Fichiers Base64 (<CsFileB>)

On peut inclure des fichiers encodés en Base64 avec la balise <*CsFileB filename= nomfichier*>, où *nomfichier* est le nom du fichier à inclure. Les données encodées en Base64 doivent se terminer par une balise </*CsFileB*>. Pour encoder les fichiers, on peut se servir des utilitaires *csb64enc* et *makecsd* (inclus dans Csound à partir de la version 5.00). Le fichier sera extrait dans le répertoire courant, et effacé à la fin de l'exécution. S'il existe déjà un fichier du même nom, il n'est pas écrasé, mais au contraire, une erreur est levée.

On peut inclure des fichiers MIDI encodés en Base64 avec la balise <*CsMidifileB filename= nomfichier*>, où *nomfichier* est le nom du fichier qui contient l'information MIDI. Il n'y a pas de balise fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser <*CsFileB*>.

On peut inclure des fichiers d'échantillons encodés en Base64 avec la balise <*CsSampleB filename= nomfichier*>, où *nomfichier* est le nom du fichier qui contient les échantillons. Il n'y a pas de balise fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser <*CsFileB*>.

Limitation de Version (<CsVersion>)

On peut se limiter à certaines versions de Csound en plaçant l'une de ces instructions entre la balise ouvrante <CsVersion> et la balise fermante </CsVersion> :

```
Before #.#
```

ou

```
After #.#
```

où #.# est le numéro de version de Csound requis. La deuxième instruction peut s'écrire simplement comme :

```
#.#
```

Ceci a été ajouté dans le version 4.09 de Csound.

Information de Licence (<CsLicence> or <CsLicense>)

Des détails de licence peuvent être inclus entre la balise ouvrante *<CsLicence>* et la balise fermante *</CsLicence>*. Il n'y a pas de format pour cette information, n'importe quel texte est acceptable. Ce texte sera imprimé par Csound sur la console lorsque le CSD sera exécuté.

Exemple

Ci-dessous un fichier exemple, test.csd, qui produit un fichier .wav échantillonné à 44,1 kHz contenant une seconde d'une onde sinus à 1 kHz. L'affichage est supprimé. test.csd a été créé à partir de deux fichiers, tone.orc et tone.sco, avec l'addition des options de ligne de commande.

```
<CsoundSynthesizer>;
  ; test.csd - un fichier Csound de données structurées

<CsOptions>
  -W -d -o tone.wav
</CsOptions>

<CsVersion>    ; section facultative
  Before 4.10  ; ces deux instructions testent si
  After 4.08   ; la version de Csound est la 4.09
</CsVersion>

<CsInstruments>
  ; à l'origine, tone.orc
  sr = 44100
  kr = 4410
  ksmps = 10
  nchnls = 1
  instr   1
      a1 oscil p4, p5, 1 ; simple oscillateur
         out a1
  endin
</CsInstruments>

<CsScore>
  ; à l'origine, tone.sco
  f1 0 8192 10 1
  i1 0 1 20000 1000 ; joue un son pur à un kHz pendant une seconde
  e
</CsScore>
```

```
</CsoundSynthesizer>
```

Fichier de Paramètres de Ligne de Commande (.csoundrc)

Si le fichier *.csoundrc* existe, il sera utilisé pour fixer les paramètres de la ligne de commande. Ceux-ci peuvent être redéfinis. Csound 5.00 et les versions ultérieures lisent ce fichier d'abord depuis le répertoire HOME (ou le chemin complet défini par la *variable d'environnement* CSOUNDRC), et ensuite depuis le répertoire courant. Si les deux existent, les options de .csoundrc du répertoire courant seront prioritaires. Ce fichier a la même forme qu'un fichier *.csd*, mais sans les balises. Les lignes commençant par # ou *;* sont traitées comme des commentaires.

Un fichier *.csoundrc* peut contenir des éléments comme ceux-ci :

-+rtaudio=portaudio -odac2 -iadc2 -+rtmidi=winmme -M1 -Q1 -m0

Dans ce cas, csound générera sa sortie en temps réel et recevra son entrée en temps réel depuis le périphérique 2, en utilisant l'interface du pilote portaudio. Il utilisera les entrées et les sorties MIDI en temps réel sur l'interface 1. Il imprimera très peu de messages (-m0). Ces options seront utilisées par défaut en l'absence d'autres options spécifiées dans la balise <CsOptions> du fichier .csd ou dans la ligne de commande (voir *Ordre de priorité*).

Prétraitement du Fichier Partition

La Fonction Extract

Cette fonction va extraire une partie d'un fichier de partition numérique triée en suivant les instructions venant d'un fichier de contrôle. Le fichier de contrôle contient une liste d'instruments et deux points dans le temps depuis (from) et à (to), de la forme :

```
instruments 1  2  from  1:27.5  to  2:2
```

Les étiquettes des composants peuvent être abrégés en i, f et t. Les points dans le temps marquent le début et la fin de l'extraction en termes de :

```
[no de section] : [no de pulsation].
```

chacune des trois parties de l'argument est optionnelle. Les valeurs par défaut lorsque i, f ou t sont manquants sont :

```
tous les instruments, début de la partition, fin de la partition.
```

Prétraitement Indépendant avec Scsort

Bien que le résultat de tout le prétraitement de la partition se trouvent dans le fichier *score.srt* après

l'exécution de l'orchestre (il existe dès que le prétraitement de la partition est fini), l'utilisateur peut vouloir parfois lancer ces phases indépendamment. La commande

```
scot nomfichier
```

va traiter le fichier au format Scot *nomfichier*, et produira comme résultat une *partition numérique standard* dans un fichier appelé *score* pour consultation ou traitement ultérieur.

La commande

```
scscort < fichierentrée > fichiersortie
```

effectuera les prétraitements de Report de Valeur (Carry), Tempo et Tri sur une partition numérique dans fichierentrée, déposant le résultat dans fichiersortie.

De même *extract*, lui aussi invoqué normalement comme élément de la *commande Csound*, peut être invoqué comme programme autonome :

```
extract xfile < partition.triée > extrait.partition
```

Cette commande attend une partition déjà triée. Une partition non triée doit d'abord passer par Scsort pour ensuite enchaîner avec le programme extract :

```
scsort < fichierpartition | extract xfile > extrait.partition
```

Utiliser Csound

On peut faire fonctionner Csound dans divers modes et configurations. La méthode originale pour lancer Csound était un programme de console (invite DOS pour Windows, Terminal pour Mac OS X). Bien sûr, ceci fonctionne toujours. Lancer csound sans argument retourne une liste d'options de commande en ligne, qui sont expliquées plus en détail dans la section *Options de Ligne de Commande (par Catégorie)*. Normallement, l'utilisateur exécute quelque chose comme :

```
csound monfichier.csd
```

ou si l'on utilise des fichiers d'orchestre (orc) et de partition (sco) séparés :

```
csound monorchestre.orc mapartition.sco
```

On peut trouver plusieurs fichiers .csd dans le répertoire des exemples. La plupart des articles de ce manuel sur les opcodes incluent également des fichiers .csd simples montrant l'utilisation de l'opcode.

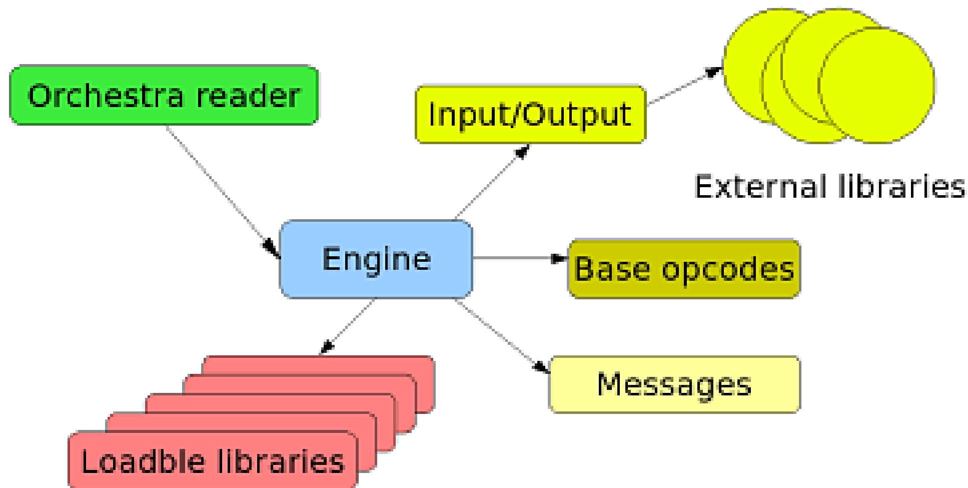
Il y a aussi plusieurs *Frontaux* que l'on peut utiliser pour lancer csound. Un *Frontal* est un programme graphique qui facilite la tâche de lancer csound, et qui fournit parfois des fonctionnalités d'édition et de composition.

Csound a aussi plusieurs moyens de produire une sortie. Il peut :

- Lire et écrire dans des fichiers son (restitution différée) - En utilisant les options *-o* et *-i* pour spécifier un fichier de sortie.
- Lire et écrire des données audio-numériques en utilisant une carte son (restitution en temps-réel) - En utilisant les options *-odac* et *-iadc*
- Lire et écrire dans des fichiers MIDI (temps différé) - En utilisant les options *-F* et *--midioutfile*.
- Lire et écrire des données MIDI en utilisant une interface et un contrôleur MIDI (contrôle en temps réel) - En utilisant les options *-M* et *-Q*.

Comment Csound5 fonctionne

Csound calcule et génère sa sortie en utilisant des "générateurs unitaires" (ugens) appelés *opcodes*. Ces opcodes sont utilisés pour définir des *instruments* dans l'*orchestre*. Quand vous lancez Csound, le moteur charge les Opcodes de base, et les opcodes contenus dans des "bibliothèques d'opcodes" séparées et chargeable. Il interprète ensuite l'orchestre (au moyen du chargeur d'orchestre). Le moteur met en place une chaîne de traitement des instruments, qui reçoit ensuite des évènements depuis la partition ou en temps réel. La chaîne de traitement utilise les modules d'entrée/sortie pour générer la sortie. Il y a des modules qui peuvent écrire dans un fichier, ou générer une *sortie audio en temps réel*.



La Structure Modulaire de Csound5.

Les tampons de traitement de Csound

Csound traite les données audio par blocs d'échantillons appelés tampons. Il y a trois couches de tampons séparées :

1. *spout* = tampon logiciel de bas niveau de Csound, contient *ksmps* trames d'échantillon. Csound traite les évènements de contrôle en temps réel toutes les *ksmps* trames d'échantillon.

2. *-b* = Tampon logiciel intermédiare de Csound (le tampon "logiciel"), en trames d'échantillon. Devrait être (mais ce n'est pas nécessaire) un multiple entier de *ksmps* (peut également être égal à ksmps). Une fois toutes les ksmps trames d'échantillon, Csound copie *spout* dans le tampon *-b*. Une fois toutes les *-b* trames d'échantillon, Csound copie le tampon *-b* dans le tampon "matériel" *-B*.

3. *-B* = tampon interne de la carte son (le tampon "matériel"), en trames d'échantillon. Devrait être (et cela peut être nécessaire) un multiple entier de *-b*. Si Csound n'arrive pas à délivrer un des *-b*, les trames d'échantillon *-b* en plus dans *-b* sont toujours là pour que la carte son continue de jouer tandis que Csound se rattrape. Mais ils peuvent être de la même taille si vous escomptez que Csound sera toujours en continuité avec la carte son.

Valeurs d'amplitude dans Csound

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "*0dbfs*" représentant l'amplitude de crête avant écrêtement, soit dans un codec AN/NA, soit dans un fichier son avec une étendue définie (ce qui est le cas de WAVE et de AIFF). A l'origine, dans Csound, cette valeur était toujours 32767, correspondant à l'étendue bipolaire d'un fichier son 16 bit ou d'un codec AN/NA 16 bit, les seules sorties possibles de Csound à l'époque. Ceci reste l'amplitude de crête *par défaut* dans Csound, pour une compatibilité descendante et vous verrez que la plupart des exemples de ce manuel utilisent toujours cette valeur (c'est pourquoi l'on trouve de grandes valeurs d'amplitude comme 10000).

La valeur *0dbfs* permet à Csound de produire des valeurs convenablement calibrées quelque soit le format utilisé, entiers sur 24 bit, nombres en virgule flottante sur 32 bit, ou même entiers sur 32 bit. Autrement dit, les valeurs d'amplitude littérales écrites dans un instrument de Csound ne concordent avec celles qui sont écrites *littéralement* dans le fichier que si la valeur *0dbfs* dans Csound correspond exactement à celle du format d'échantillonnage de la sortie. La conséquence de cette approche est que l'on peut écrire une pièce avec une certaine amplitude et en avoir une restitution correcte et identique (sans tenir compte bien sûr de la gamme dynamique meilleure des formats en haute résolution) qu'elle soit écrite dans un fichier de nombres entiers ou en virgule flottante, ou rendue en temps réel.

Note

La seule exception à ceci se produit si l'on choisit d'écrire dans un format de fichier "brut" (sans en-tête). Dans de tels cas la valeur interne *0dbfs* est sans signification, et quelques soient les valeurs utilisées, elles sont écrites inchangées. Cela permet de faire générer ou traiter par Csound des données arbitraires. C'est une chose relativement exotique à faire, mais certains utilisateurs en ont besoin.

Vous pouvez choisir de redéfinir la valeur *0dbfs* dans l'en-tête de l'orchestre, par pure commodité ou selon vos préférences. Beaucoup de personnes choisiront 1,0 (le standard pour SAOL, d'autres logiciels comme Pure Date, et pour beaucoup de plugins standard comme VST, LADSPA, CoreAudio AudioUnits, etc), mais n'importe quelle valeur est possible.

Le facteur commun dans la définition des amplitudes est l'échelle en décibel (dB), avec $0dB_{FS}$ toujours compris comme la crête numérique ; ainsi "0dbfs" veut dire valeur de "0dB Full-Scale" (sur l'étendue complète). Cette mesure est différentes des valeurs d'amplitude réelles, puisque celles-ci sont sur une échelle linéaire qui montre l'oscillation réelle autour de 0, et peuvent ainsi être positives ou négatives. Les valeurs en décibel forment une échelle logarithmique absolue, mais peuvent être également utiles pour la plupart des opcodes. On peut convertir les amplitudes de et en décibel en utilisant les fonctions *ampdb*, *ampdbfs*, *dbamp* et *dbfsamp*. De cette manière, Csound permet au programmeur d'exprimer toutes les amplitudes en dB - les amplitudes plus faibles seront alors représentées par des valeurs de décibel négatives. Cela reflète les pratiques de l'industrie (par exemple sur les indicateurs de niveau des tables de mixage, etc).

Par exemple le même niveau de -6dB (la moitié de l'amplitude) ou de -20dB représentent une amplitude linéaire par rapport à 0dbfs comme ceci :

Tableau 2. dB_{FS} en relation avec l'amplitude

dB_{FS}	0dbfs = 32767 (par défaut)	0dbfs = 1	0dbfs = 1000 (inhabituel)
0 dB	32767	1	1000
-6 dB	16384	0.5	500
-20 dB	3276.7	0.1	100

Certains utilisateurs de Csound peuvent ainsi avoir l'intention d'exprimer tous les niveaux en dB_{FS}, et éviter toute confusion ou toute ambiguïté de niveau qui pourrait autrement se produire lorsque des valeurs explicites d'amplitude sont utilisées. L'échelle en décibel reflète la réponse de l'oreille assez fidèlement, et si vous voulez exprimer un niveau vraiment doux, il peut être plus facile et plus expressif d'écrire "-46dB" que "0.005" ou "163.8".

La raison d'utiliser 0dbfs est très simple : la crête numérique est égale au niveau maximum quelque soit la résolution de l'échantillonnage. Si vous définissez un signal à -110dB, il disparaîtra s'il est restitué dans un fichier 16 bit, mais il restera (audible ou non) s'il est restitué en 24 bit ou mieux. Autrement dit,

il y a un plafond fixe mais un plancher mobile - vous pouvez définir des sons aussi doux que vous le voulez (par exemple des queues d'enveloppe), de manière prévisible, et les préserver ou non (sans changer le code de l'orchestre), selon la résolution de leur restitution (dans un fichier ou sur une e/s audio).

Une note sur l'amplitude numérique, les décibels et l'étendue dynamique

Une approche commode de l'étendue dynamique pour une certaine précision numérique est de calculer l'intervalle en décibels entre la valeur minimale et la valeur maximale pour un échantillon. En général, 1 bit (doublement du niveau) vaut 6dB, donc 16 bit = 96dB.

Ceci n'est pas entièrement exact car les valeurs des échantillons audio sont représentées sur une échelle bipolaire avec des valeurs positives et négatives, et un bit est utilisé pour le signe. Ainsi, puisque les échantillons en entiers sur 16 bit utilisent 1 bit pour le signe et 15 bit pour la valeur, l'intervalle dynamique est de 90dB.

Audio en temps-réel

L'information suivante concerne en premier lieu l'utilisation de csound à partir de la ligne de commande. Les frontaux implémentent ces caractéristiques de différentes manières, mais leur connaissance est nécessaire dans certains d'entre eux.

Les options *-i* et *-o* sont utilisées pour spécifier une sortie en temps-réel à la place de l'habituelle sortie différée dans un fichier. On utilise `-o dac` pour la sortie en temps-réel et `-i adc` pour l'entrée en temps-réel. Naturellement, on peut utiliser l'un ou les deux selon les possibilités matérielles. On peut aussi spécifier le matériel à utiliser en ajoutant un numéro ou un nom de périphérique au drapeau (voir *-i* et *-o*).

Il peut aussi être nécessaire d'utiliser l'option *-+rtaudio* pour spécifier le pilote d'interface à utiliser. Csound utilise Portaudio par défaut, qui est multi plates-formes et fiable, mais, pour obtenir de meilleures performances, on peut utiliser ALSA et JACK sur linux, et CoreAudio sur Mac. On peut utiliser ASIO sur Windows si la version de Portaudio a été compilée avec le support ASIO.

On peut voir une liste des périphériques disponibles en donnant un numéro de périphérique trop grand, par example `-o dac99`. Si vous utilisez Portaudio, ceci indiquera également si ASIO est disponible.

Tailles de Période & de Tampon

Les tailles de période et de tampon varient beaucoup d'une machine à l'autre. Plus la taille du tampon est petite et plus la latence est courte, mais cela peut causer des interruptions et des clics dans le flux audio. Les options Csound qui contrôlent les tailles de période et de tampon sont respectivement *-b* et *-B*. La taille de tampon dépend du matériel, et des essais peuvent être nécessaires pour trouver l'équilibre optimal entre une faible latence et un flux audio continu. Les valeurs données à *-b* et *-B* doivent être des puissances de deux, et la valeur de *-B* doit surpasser celle de *-b* d'au moins une puissance de deux.

Actuellement, avec `-B` fixé à 512, la latence de la sortie audio est d'environ 12 millisecondes, suffisamment rapide pour un jeu au clavier raisonnablement réactif. On peut même obtenir des latences plus courtes sur certains systèmes.

Cadence de Contrôle

De faibles valeurs de ksmps donneront en général une synthèse de meilleure qualité, mais consommeront plus de ressources système. Il n'y a pas de règle absolue pour fixer ksmps - différents orchestres nécessiteront différentes cadences de contrôle. Un instrument à guide d'onde nécessitera une valeur de ksmps de 1 (et pourra ne pas convenir au temps-réel), alors qu'une simple synthèse FM pourra fonction-

ner avec de plus grandes valeurs de ksmps sans dégradation notable du son. Si cette synthèse FM doit jouer une ligne de basse monodique, on peut utiliser une très faible valeur de ksmps, cependant des clusters de notes plus complexes nécessiteront une valeur de ksmps plus grande. Un système linux bien réglé devrait même être capable de produire des synthèses polyphoniques complexes avec des valeurs de ksmps aussi faibles que 4 ou 8. Si l'on a besoin de capacités audio duplex complètes, *-b* doit être un multiple entier de ksmps. En gardant cela à l'esprit, on peut poser comme règle empirique de n'utiliser que des puissances de deux pour ksmps.

Certains réglages diffèrent selon la plate-forme. Voir la suite pour de l'informations sur chaque plate-forme.

Entrées/Sorties en temps-réel sur Linux

Sous linux, les réglages PortAudio/PortMidi par défaut provoquent une latence plus longue que celle que l'on obtiendrait avec ALSA et/ou JACK. Les plugins PortMusic sont des serveurs audio et MIDI, qui fournissent une interface aux pilotes ALSA, tout comme le fait JACK, mais d'une manière fondamentalement différente. Pour une comparaison plus détaillée prière de se référer à :

http://jackaudio.org/faq

Utilisation d'ALSA

Le plus haut niveau de contrôle et la plus faible latence possible sont atteints en utilisant les plugins ALSA en combinaison avec l'option *--sched*. L'utilisation de *--sched* nécessite que Csound soit lancé par l'utilisateur root, ce qui peut être impossible ou indésirable dans certaines circonstances.

Les plugins ALSA nécessitent le nom ("name") d'une carte ("card") et d'un périphérique ("device"). A moins d'avoir nommé vos cartes dans ~/.asoundrc (ou /etc/asound.conf), les noms seront en fait des nombres. Pour obtenir une liste des configurations possibles, utilisez les utilitaires en ligne de commande "aplay", "arecord" et "amidi". Ces utilitaires sont inclus dans la plupart des distributions Linux, ou peuvent être téléchargés et construits à partir de ces sources :

ftp://ftp.alsa-project.org/pub/utils/

Sortie Audio

En tapant la commande suivante :

```
aplay -l
```

vous obtiendrez une liste des périphériques de reproduction audio disponibles sur votre système. Cette liste ressemble à ceci :

[....]
**** List of PLAYBACK Hardware Devices ****
card 0: A5451 [ALI 5451], device 0: ALI 5451 [ALI 5451]
[....]

Si vous avez plus d'une carte sur votre système, ou s'il y a plus d'un périphérique sur votre carte, la liste sera naturellement plus compliquée, cependant, dans tous les cas, l'information pertinente est le numéro/nom de la carte/périphérique. Afin d'utiliser la carte son ci-dessus pour la sortie audio, il faut ajouter l'option suivante à la ligne de commande Csound, dans ~/.csoundrc, ou dans la section <CsOptions> d'un CSD :

```
-+rtaudio=alsa -o dac
```

Sortie avec dmix

Si vous désirez utiliser Csound avec dmix et que votre carte son ne supporte pas le mixage matériel des flux audio, il faut régler les tampons logiciel (-b) et matériel (-B) avec un soin particulier. Si vous recevez un message du pilote ALSA de Csound qui ressemble à ceci :

```
ALSA: -B 8192 not allowed on this device; use 7526 instead
```

il y a de bonnes chances que vous puissiez utiliser dmix. Si vous utilisez dmix, les réglages de -b et de -B dans Csound doivent être synchronisés avec le taille de période (period_size) et la taille de tampon (buffer_size) de dmix respectivenemt, en utilisant le rapport du taux d'échantillonnage du projet Csound sur le taux d'échantillonnage sur lequel dmix est réglé. Les formules suivantes déterminent les réglages à utiliser pour Csound en fonction des réglages de dmix :

```
-b = (csound_sr/dmix_sample_rate) * dmix_period_size
-B = (csound_sr/dmix_sample_rate) * dmix_buffer_size
```

Par exemple, si dmix est fixé à 48000 échantillons par seconde, un period_size de 1024, et un buffer_size de 8192, si l'on exécute un projet Csound avec sr=48000, les réglages des tampons seront "-b 1024 -B8192". Si sr=24000, les réglages des tampons seront "-b 512 -B4096".

A cause de cette relation, si le taux d'échantillonnage du projet Csound ne divise pas exactement le taux d'échantillonnage utilisé par dmix, il pourra être difficile, voire impossible, de régler correctement -b et -B à cause des erreurs d'arrondi. En conséquence, si vous utilisez des taux d'échantillonnage différents que ceux que vous fixez pour dmix, nous vous suggérons de configurer dmix avec un period_size et un buffer_size divisibles par le rapport entre le taux d'échantillonnage de csound et celui de dmix. Par exemple, pour exécuter un projet avec sr=16000, les réglages suivants de dmix :

```
pcm.amix {
   type dmix
   ipc_key 50557
   slave {
       pcm "hw:0,0"
       period_time 0
       #period_size 1024
       #buffer_size 8192
       period_size 1536
       buffer_size 12288
   }
   bindings {
       0 0
       1 1
   }
}

# route ALSA software through pcm.amix
pcm.!default {
   type plug
   slave.pcm "amix"
}
```

avec period_size=1536 et buffer_size=12288 seront divisibles par 3 (le rapport du taux d'échantillonnage de csound par celui de dmix) pour obtenir "-b 512 -B4096" ((16000/48000) * 1536 = 512, (16000/48000) * 12288 = 4096).

Note

Pour la plupart des cartes son qui sont affectées par ceci, le taux d'échantillonnage par défaut de la carte sera 48000 et ceux de dmix seront 1024 et 8192.

Entrée Audio

Normalement, la même carte étant utilisée pour les entrées et les sorties, en continuant l'exemple précédant, l'option :

```
-i adc:hw:0,0
```

sera ajouté pour l'entrée audio à partie du périphérique 0 de la carte 0. Pour utiliser la carte par défaut, on emploie l'option suivante, mais attention, ça peut ne pas fonctionner :

```
-i adc
```

Si l'on désire utiliser une autre carte ou un autre périphérique pour l'entrée, la commande suivante fournira une liste de périphériques en entrée :

```
arecord -l
```

Si, par exemple, vous désirez utiliser en sortie une interface audio USB, qui est la deuxième "carte" dans votre système, alors que vous désirez utiliser en entrée votre carte son interne, la première carte de votre installation, positionnez les options suivantes à l'endroit adéquat :

```
-+rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,0
```

Si vous désirez utiliser le second périphérique sur votre interface USB, pour envoyer un flux audio à un canal particulier, vous utiliserez les options suivantes :

```
-+rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,1
```

Entrée MIDI

Csound ne crée pas automatiquement son propre port de séquenceur ALSA. Il crée un port midi direct ALSA à chaque lancement. Afin de permettre à votre orchestre de recevoir une entrée MIDI vous pouvez utiliser VirMIDI ou MIDIThru, selon vos préférences. La configuration de ces ports MIDI virtuels a été largement couverte ailleurs, voir le Linux MIDI how-to [http://www.midi-howto.com/] ou parcourez la documentation de votre distribution ou la documentation ALSA à la recherche d'instructions pour intaller et configurer VirMidi ou MIDIThru. Une fois ceci réalisé, la commande :

```
amidi -l
```

retourne une liste des périphériques disponibles. Cette liste ressemble à ceci :

```
   [....]
Device    Name
hw:1,0    Virtual Raw MIDI (16 subdevices)
hw:1,1    Virtual Raw MIDI (16 subdevices)
hw:1,2    Virtual Raw MIDI (16 subdevices)
hw:1,3    Virtual Raw MIDI (16 subdevices)
hw:2,0,0  PCR MIDI
hw:2,0,1  PCR 1
```

Dans cet exemple, Csound peut se connecter à n'importe lequel des quatre ports virtuels MIDI directs,

pour y écouter l'entrée MIDI. L'option suivante indique à Csound d'écouter sur le premier de ces ports :

```
-+rtmidi=alsa -Mhw:1,0
```

Il faudra ensuite connecter votre matériel ou votre contrôleur logiciel au port qui accueille votre synthétiseur Csound. La manière la plus simple de le faire est d'employer l'utilitaire "aconnect". Tapez :

```
aconnect -li
```

pour une liste des périphériques d'entrée disponibles, et :

```
aconnect -lo
```

pour une liste des périphériques de sortie disponibles (y compris le port auquel Csound a été connecté). Cette liste ressemble à ceci :

```
#aconnect -li
client 0: 'System' [type=kernel]
    0 'Timer           '
    1 'Announce        '
        Connecting To: 15:0
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]
    0 'VirMIDI 1-0     '
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
    0 'VirMIDI 1-1     '
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
    0 'VirMIDI 1-2     '
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
    0 'VirMIDI 1-3     '
client 24: 'PCR' [type=kernel]
    0 'PCR MIDI        '
    1 'PCR 1           '
    2 'PCR 2           '



#aconnect -lo
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]
    0 'VirMIDI 1-0     '
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
    0 'VirMIDI 1-1     '
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
    0 'VirMIDI 1-2     '
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
    0 'VirMIDI 1-3     '
client 24: 'PCR' [type=kernel]
    0 'PCR MIDI        '
    1 'PCR 1           '
```

Dans l'exemple suivant, le clavier USB qui est listé ci-dessus comme le client 24 sera connecté au synthétiseur Csound qui est à l'écoute sur le premier port VirMIDI. Le clavier a trois ports de sortie. Le premier (24:0) transmet les messages reçus sur le port d'entrée MIDI, le second (24:1) transmet les messages de touches et de contrôleurs, et le troisième (24:2) transmet les messages système exclusif. La

commande suivante connecte le second port du clavier au synthétiseur Csound :

```
aconnect 24:1 20:0
```

Il faut garder à l'esprit que Csound agit comme un périphérique MIDI direct et non comme un client du séquenceur ALSA. Cela signifie que Csound n'apparaitra pas dans la liste des périphériques MIDI et ne sera pas disponible pour un usage direct avec *aconnect*, ainsi, il faut se connecter à un périphérique virtuel (comme 'virtual raw MIDI' ou 'MIDI through') pour des connexions persistantes, ou se connecter directement à la destination en utilisant les options de ligne de commande.

Sortie MIDI

On peut connecter Csound à n'importe quel périphérique qui apparaît dans la liste des ports de sortie du séquenceur ALSA, que l'on obtient par "amidi -l" comme ci-dessus. Afin de connecter un synthétiseur Csound au port MIDI out du clavier listé ci-dessus, on utilise l'option suivante :

```
-Qhw:2,0,0
```

Temps-partagé

Si vous avez la possibilité d'exécuter Csound en tant qu'utilisateur root, l'option "--sched" permet d'améliorer spectaculairement les performances temps-réel avec ALSA, cependant vous pouvez bloquer le système si vous faites quelque chose de stupide. N'UTILISEZ PAS "--sched" si vous choisissez JACK pour la sortie audio. JACK contrôle le temps-partagé pour les applications audio qui l'utilisent, et il essaie également de fonctionner avec la priorité maximale. Si l'option "--sched" est utilisée, Csound et JACK vont entrer en compétition au lieu de coopérer, ce qui aura pour résultat de piètres performances.

Utiliser JACK

La manière la plus simple d'activer les entrées-sorties avec le plugin JACK est :

```
-+rtaudio=jack -i adc -o dac
```

En outre, il y a quelques options de ligne de commande spécifiques à JACK :

Options de ligne de commande de JACK

-+jack_client=[nom_de_client]
: Le nom de client par défaut de Csound est 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser des noms de client différents pour éviter les conflits de noms.

-+jack_inportname=[préfixe du nom de port d'entrée], -+jack_outportname=[préfixe du nom de port de sortie]
: Le préfixe du nom des ports d'entrée/sortie JACK de Csound ; la valeur par défaut est 'input' et 'ouput'. Le nom complet d'un port est obtenu en ajoutant le numéro du canal au préfixe. Exemple : avec les réglages par défaut, un orchestre stéréo créera les ports suivants en mode d'opération full duplex :

```
csound5:input1             (enregistrement gauche)
csound5:input2             (enregistrement droite)
csound5:output1            (reproduction gauche)
csound5:output2            (reproduction droite)
```

en microsecondes]	Depuis Csound version 5.01, cette option est dépréciée et ignorée.

Connecter Csound à d'autres clients JACK

Il n'y a par défaut aucune connection (on doit utiliser jack_connect, qjackctl, ou un utilitaire semblable) ; cependant, on peut connecter le plugin à des ports spécifiés par '-iadx:portname_prefix' ou '-odac:portname_prefix', où portname_prefix est le nom d'un port sans le numéro de canal, tel que 'alsa_pcm:capture_' (pour -i adc), ou 'alsa_pcm:playback_' (pour -o dac).

Notes sur les tailles de tampon

Les données audio sont reçues de et envoyées vers le serveur JACK par Csound au moyen d'un tampon circulaire qui est contrôlé par les options -b et -B. -B est la taille totale du tampon, tandis que -b est la taille d'une période. Ces valeurs sont arrondies de façon à ce que la taille totale soit un multiple entier de la taille de la période et supérieure à cette dernière. La différence de taille entre le tampon de Csound et la période doit être supérieure ou égale à la taille de la période de JACK.

Si l'on utilise en même temps -iadc et -odac, l'option -b doit être fixée à une valeur multiple de ksmps.

Exemple de réglage de tampon pour obtenir une faible latence sur un système rapide :

```
jackd -d alsa -P -r 48000 -p 64 -n 4 -zt &
csound -+rtaudio=jack -b 64 -B 256 [...]
```

avec temps-partagé pour le temps-réel (en tant que root) :

```
jackd -R -P 90 -d alsa -P -r 48000 -p 64 -n 2 -zt &
csound --sched=80,90,10 -d -+rtaudio=jack -b 64 -B 192 [...]
```

Pour améliorer les performances, utiliser des valeurs de ksmps comme 32 ou 64.

Le taux d'échantillonnage de l'orchestre doit être le même que celui du serveur JACK.

Utilisation de Pulseaudio

Le support de Pulseaudio [http://www.pulseaudio.org/] a été ajouté dans Csound 5.09. Vous pouvez spécifier les réglages suivants :

1. Noms de sortie : il est possible d'utiliser un nombre à la place du nom complet, ainsi -odac:1 sélectionne votre second périphérique (le compte commence à 0).
2. Nom du serveur : il est possible de se connecter à un serveur spécifique en utilisant -+server=<server_string> où server_string est le nom d'un serveur ou une chaîne plus complexe de sélection de serveur (voir pulseaudio.org [http://www.pulseaudio.org/] sur les chaînes de serveur). Ceci est transparent sur un réseau et permet les connexions à des machines distantes.
3. Noms de flot : il est possible d'étiqueter les flots générés par csound, en utilisant -+output_stream=<stream-name> et -+input_stream=<stream-name>

Voici un exemple de ligne de commande :

csound -odac:1 examples/trapped.csd -+rtaudio=pulse -+server=unix:/tmp/pulse-victor/native -+output_stream=trapped

Windows

Audio en temps-réel

Les utilisateurs de Windows peuvent utiliser soit le module temps-réel par défaut *PortAudio*, soit le module temps-réel *winmm*. Le module winmm est un module natif de Windows qui fournit une grande stabilité, mais une latence qui sera en général trop grande pour une interaction en temps réel. Pour activer un module temps-réel on peut utiliser l'option *-+rtaudio* avec la valeur *portaudio* ou *winmme*. La valeur par défaut est *portaudio*, qui est active sans avoir à être spécifiée.

On doit aussi spécifier le périphérique son que l'on veut utiliser, et indiquer que l'on veut générer une sortie audio en temps-réel plutôt qu'un fichier son vers une sortie disque. Pour cela, on doit utiliser l'option *-odac* ou *-o dac*, qui indique comme sortie de csound les convertisseurs Numérique-Analogique plutôt qu'un fichier. En ajoutant un numéro après l'option (par exemple -odac2), on peut choisir le numéro du périphérique désiré. Pour trouver les périphériques disponibles dans le système, on peut utiliser un numéro trop grand (par exemple -odac99), et csound rapportera une erreur ainsi que la liste des périphériques disponibles.

Lorsque l'on choisit le numéro de périphérique sous Portaudio, on choisit également l'interface du pilote, car Portaudio supporte WinMME, DirectX et ASIO. Si vous avez une interface compatible ASIO ou un émulateur de pilote ASIO comme ASIO4ALL [http://www.asio4all.com], le périphérique affichera plusieurs durées, une pour chaque interface de pilote. Comme ASIO fournit la meilleure latence pour un système, il devrait être choisi pour une sortie audio en temps-réel s'il est disponible.

On active l'entrée audio en temps-réel par *-iadc*, ce qui règle csound sur l'écoute de l'entrée audio temps-réel. On peut également choisir le périphérique par son numéro, et tester les périphériques disponibles avec un numéro trop grand. Notez que pour les entrées on utilise 'adc' au lieu de 'dac'. Assurez-vous que la bonne entrée soit sélectionnée dans le panneau de contrôle de votre carte son.

MIDI en temps-réel

Pour activer le MIDI en temps-réel dans Windows on peut utiliser l'option *-M* pour l'entrée MIDI et l'option *-Q* pour la sortie MIDI. On peut spécifier le numéro du périphérique après le drapeau (par exemple -M2), et aussi trouver les périphériques disponibles en donnant un numéro trop grand.

Csound utilise par défaut le module MIDI PortMidi, mais il y a aussi un module natif winmme, que l'on peut activer avec l'option :

```
-+rtmidi=winmme
```

Un ensemble d'options typique pour activer l'Audio et les E/S MIDI en temps-réel ressemblera à ceci:

```
-+rtmidi=winmme -M1 -Q1 -+rtaudio=portaudio -odac3 -iadc3
```

Mac

Prochainement...

Optimisation de la Latence Audio en E/S

Pour atteindre la latence la plus basse possible sans interruptions audio, il faut régler une combinaison de variables. Les valeurs retenues dépendront de la plate-forme et du système, et aussi de la complexité

des calculs audio mis en œuvre. Il faut ajuster *ksmps* dans l'orchestre, ainsi que la taille du tampon logiciel (*-b*) et celle du tampon matériel (*-B*).

Habituellement la solution la plus simple est la suivante :

1. Fixer *ksmps* à une valeur de compromis entre qualité et performance, sans ajuster *-B* du tout.

2. Fixer *-b* à une puissance de deux négative.

   Pour obtenir les valeurs optimales, commencer avec une valeur qui vous semble trop petite, c'est-à-dire -1, et continuer ensuite en "augmentant", -2, -4, etc., jusqu'à ne plus avoir de défauts dans le son. La valeur réelle de *-b* sera la valeur absolue de *-b * ksmps*.

3. Réduire le tampon matériel (*-B*). Partir de la valeur par défaut (1024 sur Linux, 4096 sur Max OS X, 16384 sur Windows), et la réduire de moitié à chaque fois, jusqu'à entendre à nouveau des défauts. La remonter alors jusqu'à ce que l'exécution soit continue.

Cette procédure s'applique aux cartes 16 bit. Si vous avez une carte son 24 bit, alors *-B* doit valoir 3/2, ou 3 fois *-b*, plutôt que 2 ou 4 fois. Csound travaille avec des nombres en virgule flottante en 32 bit ou 64 bit alors que la plupart des cartes son utilisent des entiers en 16 ou 24 bit. *-b* est le tampon interne, c'est pourquoi il traite de la partie 32 ou 64 bit, tandis que *-B* est le tampon matériel, et il traite ainsi de la partie 16 ou 24 bit. Le réglage par défaut de csound pour les réels est *-B = 4 * -b*. C'est une valeur sûre pour une carte 16 bit. On peut s'en sortie avec *-B = 2 * -b*, mais c'est le minimum absolu. Par exemple, si votre réglage est *-b1024 -B2048*, csound vous dira ceci :

```
audio buffered in 1024 sample-frame blocks
writing 4096-byte blocks to dac
```

4096 octets font 32768 bits. 32768/32 = 1024, notre taille de bloc de trames d'échantillons, 1024 * 32/16 = 2048, notre taille de tampon. Si nous réduisons la valeur de *-B*, il faudra réduire la valeur de *-b* d'un montant proportionnel afin de continuer à écrire des entiers en 16 bit sur le CNA. La taille minimale de *-b* est *(-B * bitrate)/32*. Cela veut dire que le rapport minimum de *-b* à *-B* doit être :

- 1/2 en 16 bit
- 2/3 en 24 bit
- 1/1 en 32 bit

Bien qu'il n'y ait théoriquement pas de rapport maximum, il n'y a aucun sens à avoir un rapport très élévé ici, car le tampon logiciel doit remplir le tampon matériel avant de retourner. Si le rapport est élevé, cela prendra plus de temps, annulant le bénéfice de mettre une petite valeur pour *-b*.

Il faudra varier la valeur de *-b* en fonction de la complexité de l'instrument sur lequel vous travaillez, mais comme elle est intimement liée à celle de ksmps, il vaut mieux la synchroniser avec *ksmps* et partir de là. Une manière de faire est de décider quelle sera la longueur optimale de la chute de vos enveloppes (pour l'effet désiré), de fixer toutes les enveloppes au maximum, de donner vous-même une valeur généreuse à *-b*, et de jouer. S'il y a des interruptions, doubler ksmps, et répéter le processus jusqu'à obtenir la fluidité, descendre ensuite la valeur de *-b* aussi bas que possible.

La valeur de *-B* est d'abord déterminée par le système d'explotation et la carte son. Essayez de trouver (par la méthode ci-dessus) jusqu'où vous pouvez descendre, et utilisez cette valeur (ou une valeur supérieure par sécurité). Si vous rencontrez des problèmes ce sera probablement à cause d'une valeur de ksmps inappropriée, d'une valeur de *-b* trop faible, ou de nombres hors-norme (voir *denorm*).

Configuration

Après avoir installé une distribution binaire ou bien avoir construit Csound à partir des sources, il faut configurer Csound afin de l'adapter à votre système. Les installeurs réalisent habituellement ces étapes automatiquement pour vous.

Sur toutes les plates-formes il faut s'assurer que le ou les répertoires contenant les bibliothèques des plugins de Csound sont indiqués dans une variable d'environnement `OPCODEDIR` ou `OPCODEDIR64` en fonction de la précision utilisée par les binaires compilés.

Les opérateurs Python nécessitent actuellement au moins Python 2.4 que l'on peut télécharger à www.python.org [http://www.python.org] s'il n'est pas déjà installé sur votre système. On peut tester s'il est installé en tapant 'python' depuis une invite de commande ou une fenêtre DOS.

Windows

Sur Windows, assurez-vous que le ou les répertoires (normalement le répertoire `C:\Program Files\Csound`) contenant le répertoire des exécutables de Csound est dans votre variable `PATH`, ou bien copiez tous les fichiers exécutables dans le répertoire `system32` de Windows. En fonction de votre méthode d'installation, il peut être aussi nécessaire de fixer les variables d'environnement `OPCODEDIR` et `OPCODEDIR64`. En supposant que Csound est installé dans le répertoire par défaut `C:\Program Files\Csound` vous pouvez utiliser (sinon fixez les chemins en conséquence) :

```
set OPCODEDIR=C:\Program Files\Csound\plugins
set OPCODEDIR64=C:\Program Files\Csound\plugins64
set PATH=%PATH%;C:\Program Files\Csound\bin
```

python24.dll ou python25.dll manquante

S'il apparaît une fenêtre pop-up au sujet de la bibliothèque Python manquante (python24.dll ou python25.dll) et que vous n'avez pas besoin des opérateurs python, effacez simplement `C:\Program Files\Csound\plugins\py.dll` et `C:\Program Files\Csound\plugins64\py.dll`, et la fenêtre pop-up au sujet de la bibliothèque Python manquante ne devrait plus réapparaitre

Unix et Linux

Sur Unix et Linux, installez le programme Csound dans l'un des répertoires `bin` du système, normalement `/usr/local/bin`, et les bibliothèques partagées de Csound et des plugins dans des endroits comme `/usr/local/lib/csound/plugins` ou `/usr/local/lib/csound/plugins64` et assurez-vous que les variables d'environnement `OPCODEDIR` et `OPCODEDIR64` sont remplies correctement.

CsoundAC

CSoundAC nécessite quelques configurations supplémentaires. Sur toutes les plates-formes, CsoundAC nécessite que vous ayez installé Python sur votre ordinateur. Le répertoire contenant la bibliothèque partagée `_CsoundAC` et le fichier `CsoundAC.py` doit être dans votre variable d'environnement `PYTHONPATH`, afin que le runtime Python sache comment charger ces fichiers.

Syntaxe de l'Orchestre

L'orchestre Csound (.orc) ou la section *<CsInstruments>* d'un fichier csd, contient :

- Une *section d'en-tête*, qui spécifie les options globales pour l'exécution des instruments.
- Une liste d'*opcodes définis par l'utilisateur (UDO)* et de *blocs d'instrument* contenant les définitions des UDO et des instruments.

L'en-tête de l'orchestre, les blocs d'instrument, et les UDOs contiennent des *instructions d'Orchestre*. Dans Csound une *instruction d'orchestre* a le format :

```
étiquette:   résultat opcode argument1, argument2, ... ;commentaires
```

L'étiquette est facultative et indentifie l'instruction de base qui suit comme cible potentielle d'une opération goto (voir *Contrôle du Déroulement du Programme*). Une étiquette n'a aucun effet sur l'instruction en soi.

Selon leur fonction, certains opcodes ne produisent pas de sortie et n'ont donc pas de valeur de retour. D'autres ne prennent pas d'argument et produisent seulement un résultat.

Chaque instruction d'orchestre doit tenir sur une seule ligne, cependant les longues lignes peuvent être continuées sur la ligne suivante grâce au caractère '\'. Ce caractère indique que la ligne suivante fait partie de la ligne courante, de façon à pouvoir couper une ligne pour en faciliter la lecture, comme ceci :

```
a2  oscbnk kcps, 1.0, kfmd1, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148,  \
           0, 0, 0, 0, 0, 0, -1,                                          \
           kfnum, 3, 4
```

Les commentaires sont facultatifs et ils ont pour but de permettre à l'utilisateur de commenter le code de son orchestre. Les commentaires commencent par un point-virgule (;) et s'étendent jusqu'à la fin de la ligne. Les commentaires peuvent optionnellement être écrits en style C, s'étendant sur plusieurs lignes comme ceci :

```
/* Tout ce qui se trouve ici --------
   est un commentaire qui peut couvrir
   plusieurs lignes ---------  */
```

Le reste (résultat, opcode, et arguments) forme l'instruction de base. C'est également facultatif, ce qui veut dire qu'une ligne peut n'avoir qu'une étiquette ou un commentaire ou bien être complètement blanche. Si elle est présente, l'instruction de base doit être entièrement contenue dans une ligne, et elle est terminée par un retour chariot et un linefeed.

L'opcode détermine l'opération à effectuer ; habituellement, il prend un certain nombre de valeurs en entrée (ou arguments, au maximum environ 800) ; et il a normalement un champ résultat variable dans lequel il envoie les valeurs de sortie à un certain taux de cadencement fixe. Il y a quatre taux de cadencement possibles :

1. une seule fois, au moment de l'initialisation de l'orchestre (en fait une affectation permanente)
2. une fois au début de chaque note (à la date (init) de l'initialisation : taux-i)

3. à chaque passage dans la boucle de contrôle de l'exécution (taux de contrôle, ou taux-k)

4. à chaque échantillon sonore de chaque boucle de contrôle (taux d'exécution audio, ou taux-a)

Instructions de l'En-tête de l'Orchestre

L'*En-tête de l'Orchestre* contient l'information globale qui s'applique à tous les instruments et qui définit les aspects de la sortie de Csound. On y fait parfois référence comme *instr 0*, parce qu'il se comporte comme un instrument, mais sans traitement de taux-k ou de taux-a (seuls les opcodes et les instructions qui fonctionnent au taux-i y sont autorisés).

Une *instruction d'en-tête d'orchestre* n'opère qu'une fois, à l'initialisation de l'orchestre. La plupart du temps il s'agit de l'affectation d'une valeur à un *symbole global réservé*, par exemple sr = 20000. Toutes les instructions d'en-tête d'orchestre appartiennent au pseudo instrument 0, dont un passage *init* est effectué avant tout autre instrument au temps 0 de la partition. Toute *instruction ordinaire* peut servir d'instruction d'en-tête d'orchestre, par exemple gifreq = cpspch(8.09) à condition d'être seulement une opération du moment d'initialisation. Les instructions placées normallement dans un en-tête d'orchestre sont :

- *0dbfs*
- *ctrlinit*
- *ftgen*
- *kr*
- *ksmps*
- *massign*
- *nchnls*
- *pgmassign*
- *pset*
- *seed*
- *sr*
- *strset*

Par exemple, un en-tête de Csound peut ressembler à ceci :

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

massign 1, 10
```

Instructions de Bloc d'Instrument et d'Opcode

Un bloc d'instrument comprend des *instructions ordinaires* qui fixent des valeurs, contrôlent le déroule-

ment logique, ou appellent les différents sous-programmes de traitement du signal qui mènent à la sortie audio. Les instructions qui définissent un bloc d'instrument sont :

- *instr*
- *endin*

Un bloc d'instrument ressemble à ceci :

```
    instr 1   ; Un simple oscillateur sinusoïdal
aout  oscils  10000, 440, 0
out   aout
    endin
```

Les instructions qui définissent un bloc d'opcode défini par l'utilisateur (UDO) sont :

- *opcode*
- *endop*

voir la section *UDO* pour plus d'information.

Instructions Ordinaires

On utilise une *instruction ordinaire* soit lors de l'initialisation soit pendant l'exécution soit durant les deux. Les opérations qui produisent un résultat fonctionnent formellement au taux de ce résultat (c'est-à-dire, pendant l'initialisation pour les résultats de taux-i ; pendant l'exécution pour les résultats de taux-k et de taux-a), avec pour seule exception l'opcode *init*. Cependant, la plupart des générateurs et des modificateurs produisent des signaux que ne dépendent pas seulement de la valeur instantanée de leurs arguments mais aussi d'un état interne conservé. Ainsi, ces unités de la période d'exécution ont un composant implicite de la période d'initialisation pour créer cet état. Le type temporel d'une opération qui ne produit pas de résultat est apparent dans l'opcode.

Les arguments sont des valeurs qui sont envoyées à une opération. La plupart des arguments accepteront des expressions arithmétiques composées de constantes, de variables, de symboles réservés, de convertisseurs de valeur, d'opérations arithmétiques, et de valeurs conditionnelles.

Constantes et Variables

Les *constantes* sont des nombres en virgule flottante tels que 1, 3.14159 ou -73.45. Elles sont constamment disponibles et leur valeur ne change pas.

Les *variables* sont des cellules nommées contenant des nombres. Elle sont constamment disponibles et peuvent être mises à jour à l'un des quatre taux de mise à jour (initialisation seulement, taux-i, taux-k, taux-a). Les variables de taux-i et de taux-k sont scalaires (c'est-à-dire qu'elle ne peuvent prendre qu'une valeur à la fois) et sont utilisées principalement pour stocker et rappeler des données de contrôle, données qui changent au rythme des notes (pour les variables de taux-i) ou au taux de contrôle (pour les variables de taux-k). Les i- et les k-variables sont ainsi utiles pour stocker les valeurs des paramètres de note, hauteurs, durées, fréquences variant lentement, vibratos, etc. D'un autre côté, les variables de taux-a sont des tableaux ou vecteurs d'information. Bien que rafraichies pendant le même passage de contrôle de la période d'exécution que les variables de taux-k, ces cellules de tableau représentent une résolution temporelle plus fine en divisant la période de contrôle en durées d'échantillons (voir *ksmps*). Les variables de taux-a sont utilisées pour stocker et rappeler des données qui changent au taux d'échantillonnage audio (par exemple les signaux de sortie des oscillateurs, des filtres, etc.).

On distingue également les variables locales des variables globales. Les variables *locales* sont privées dans un instrument, et un autre instrument ne peut y accéder ni en lecture ni en écriture. Leurs valeurs sont conservées, et leur information est reportée de passage en passage (par exemple de la période d'initialisation à la période d'exécution) à l'intérieur d'un instrument. Les noms de variable locale commencent par la lettre *p, i, k*, ou *a*. Le même nom de variable locale peut apparaître dans plusieurs blocs d'instrument différents sans conflit.

Les variables *globales* sont des cellules qui sont accessibles par tous les instruments. Leurs noms sont formés soit comme les noms locaux précédés de la lettre *g*, soit de symboles réservés spéciaux. Les variables globales sont utilisées pour diffuser des valeurs générales, pour la communication entre instruments (sémaphores), ou pour envoyer un son d'un instrument à l'autre (par exemple un mixage avant une réverbération).

Etant données ces distinctions, il y a huit formes de variables locales et globales :

Tableau 3. Types de Variables

Type	Moment de Renouvellement	Local	Global
symboles réservés	permanent	--	rsymbole
p-champs de partition	temps-i	p nombre	--
variables d'initialisation	temps-i	i nom	gi nom
signaux de contrôle	temps-p, taux-k	k nom	gk nom
signaux audio	temps-p, taux-k (tous les échantillons audio dans une passe-k)	a nom	ga nom
types de données spectrales	taux-k	w nom	--
flots de données spectrales	taux-k	f nom	gf nom
variables chaînes	temps-i et optionnellement temps-k	S nom	gS nom

où *rsymbole* est un symbole réservé spécial (par exemple *sr, kr*), *nombre* est un entier positif faisant référence à un p-champ de partition ou à un numéro de séquence, et *nom* est une chaîne composée de lettres, du caractère de soulignement, et/ou de chiffres, avec une signification locale ou globale. Comme on peut le voir, les paramètres de partition sont des variables de taux-i dont les valeurs sont copiées à partir de l'instruction de partition appelante juste avant la passe d'initialisation d'un instrument, tandis que les contrôleurs MIDI sont des variables que l'on peut mettre à jour de manière asynchrone depuis un fichier MIDI ou un périphérique MIDI.

Initialisation de Variable

Les opcodes qui permettent l'initialisation de variable sont :

- *assign*
- *divz*
- *init*

- *tival*

Macros de Constantes Mathématiques Prédéfinies

Csound définit plusieurs constantes mathématiques importantes par des *Macros*. On peut consulter la liste complète *ici*.

Expressions

On peut composer des expressions de n'importe quelle profondeur. Chaque partie d'une expression est évaluée à son propre taux. Par exemple, si tous les termes d'une sous-expression changent au taux de contrôle ou plus lentement, cette sous-expression ne sera évaluée qu'au taux de contrôle ; le résultat peut alors être utilisé dans une évaluation au taux audio. Par exemple, dans

```
k1 + abs(int(p5) + frac(p5) * 100/12 + sqrt(k1))
```

100/12 sera évalué à l'initialisation de l'orchestre, les expressions en p5 seront évaluées à l'initialisation de la note, et le reste de l'expression à chaque période-k. Le tout pourrait apparaître en position d'argument dans un générateur unitaire, ou bien faire partie d'une instruction d'affectation.

Répertoires et Fichiers

Plusieurs générateurs et la commande Csound elle-même spécifient des noms de fichier pour l'écriture ou la lecture. Ceux-ci peuvent parfois être des chemins complets, dont le répertoire cible est complètement spécifié. Lorsque le chemin n'est pas complet, les noms de fichiers sont recherchés dans plusieurs répertoires dans un ordre dépendant de leur type et de la valeur de certaines variables d'environnement. Ces dernières sont facultatives, mais elles peuvent servir à partitionner et à organiser les répertoires de façon à partager les fichiers plutôt que de les dupliquer dans plusieurs répertoires de l'utilisateur. Les variables d'environnement peuvent définir des répertoires pour les fichiers son (SFDIR), les sons échantillonés (SSDIR), les analyses de son (SADIR), et les fichiers à inclure pour l'orchestre et la partition (INCDIR).

A partir de la version 5.00 de Csound, ces variables d'environnement peuvent spécifier plusieurs répertoires dans une liste dont le séparateur est le point-virgule (;). Si un fichier est trouvé à plusieurs endroits, c'est le premier de ceux-ci qui à la priorité.

L'ordre de recherche est :

1. Les fichiers son en écriture sont placés dans SFDIR (s'il existe), sinon dans le répertoire courant.

2. Les fichiers son en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans SSDIR puis dans SFDIR.

3. Les fichiers de contrôle d'analyse en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans SADIR.

4. Les fichiers MIDI en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans MFDIR, SSDIR et SFDIR.

5. Les fichiers de code à inclure dans les fichiers d'orchestre et de partition (avec *#include*) sont recherchés d'abord dans le répertoire courant, ensuite dans le même répertoire que le fichier d'orchestre ou de partition (respectivement), enfin dans INCDIR.

Nomenclature

Tout au long de ce document, les opcodes sont indiqués en **caractères gras** et les mnémoniques de leurs arguments et de leur résultat, lorsqu'ils sont mentionnés dans le texte, sont écrits en *italique*. Les noms d'arguments sont généralement des mnémoniques (*amp*, *phs*), et le résultat est souvent dénoté par la lettre *r*. Tous commencent par une qualification de type *i, k, a,* ou *x* (par exemple *kamp, iphs, ar*). Le préfixe *i* dénote des valeurs scalaires au temps de l'initialisation de note ; les préfixes *k* ou *a* dénotent des valeurs de contrôle (scalaires) et audio (vectorielles), modifiées et référencées en continu tout au long de l'exécution (c'est-à-dire à chaque période de contrôle tant que l'instrument est actif). Les arguments sont utilisés aux temps indiqués par leur préfixe ; les résultats sont créés aux temps de leur préfixe, et restent disponibles ensuite pour être utilisés comme entrées ailleurs. A part quelques exceptions, les taux des arguments ne peuvent pas dépasser le taux du résultat. La validité des entrées est définie comme suit :

- arguments avec préfixe *i* doivent être valides à l'initialisation ;
- arguments avec préfixe *k* peuvent être des valeurs de contrôle ou d'initialisation (qui restent valides) ;
- arguments avec préfixe *a* doivent être des entrées vectorielles ;
- arguments avec préfixe *x* peuvent être soit des vecteurs soit des scalaires (le compilateur distinguera).

Tous les arguments, sauf précision contraire, peuvent être des expressions dont les résultats sont conformes à la liste ci-dessus. La plupart des opcodes (tels que **linen** et **oscil**) peuvent être utilisés dans plusieurs modes, le choix étant déterminé par le préfixe ou le symbole du résultat.

Tout au long de ce manuel, le terme "opcode" est utilisé pour indiquer une commande qui produit habituellement une sortie au taux-a, -k ou -i, et qui forme toujours la base d'une instruction complète d'un orchestre Csound. Des éléments comme "+" ou "*sin*(x)" ou, "(a >= b **?** c **:** d)" sont appelés "opérateurs."

Macros

Les macros de l'orchestre fonctionnent comme les macros du préprocesseur C, et remplacent le contenu de la macro dans l'orchestre avant sa compilation. Les opcodes qui servent à créer, appeler, ou annuler les macros de l'orchestre sont :

- *#define*
- *$NAME*
- *#ifdef*
- *#ifndef*
- *#end*
- *#else*
- *#include*

- *#undef*

On peut aussi définir des macros de l'orchestre au moyen de l'option de la ligne de commande *--omacro:*.

On peut trouver plus d'information et des exemples sur l'utilisation des macros de l'orchestre à *#define*.

Ces opcodes font référence aux macros de l'orchestre ; pour les macros de la partition, voir *Macros de Partition*.

Instruments Nommés

La syntaxe de l'orchestre a été modifiée récemment pour permettre de définir des instruments avec des noms en chaîne de caractères. On peut appeler les instruments ainsi nommés depuis la partition et ils sont supportés par un certain nombre d'opcodes.

Syntaxe

Un instrument nommé est déclaré comme suit :

```
instr Nom[, Nom2[, Nom3[, ...]]]
[...]
endin
```

Un instrument seul peut avoir autant de noms que l'on veut, et chacun de ces noms peut être utilisé pour appeler l'instrument. De plus, il est possible d'utiliser des nombres comme des noms, dénotant un instrument numéroté de façon standard, ce qui fait que la déclaration suivante est également valide :

```
instr 100, Nom1, 99, Nom2, 1, 2, 3
```

Un nom d'instrument est constitué de lettres, de chiffres, et du caractère de soulignement (_), sans limite de taille, cependant, le premier caractère ne doit pas être un chiffre. Optionnellement, le nom de l'instrument peut-être préfixé par un caractère '+' (voir ci-dessous), par exemple :

```
instr +Reverb
```

Pour tous les noms d'instrument, un numéro est affecté automatiquement (note : si le niveau des messages (-m) n'est pas nul, ces numéros sont imprimés sur la console pendant la compilation de l'orchestre), en suivant ces règles :

- le nombre est choisi parmi les numéros d'instrument non affectés en ordre ascendant, en commençant par 1
- les numéros sont affectés dans l'ordre de définition des noms d'instrument, si bien que les derniers instruments nommés auront toujours un numéro plus élevé (sauf si le modificateur '+' est utilisé)
- si le nom de l'instrument est préfixé par un '+', le numéro affecté sera plus grand que tous ceux des autres instruments sans le '+' (numérotés et nommés). S'il y a plusieurs instruments '+', la numérotation de ceux-ci suivra l'ordre de leur définition, selon la règle ci-dessus.

  L'utilisation de '+' est surtout utile pour la sortie globale ou les instruments d'effets, qui doivent être

exécutés après les autres instruments.

Exemple de numérotation d'instruments :

```
instr 1, 2
endin

instr Instr1
endin

instr +Effet1, Instr2
endin

instr 100, Instr3, +Effet2, Instr4, 5
endin
```

Dans cet exemple, les numéros d'instrument sont affectés comme suit :

```
Instr1: 3
Effet1: 101
Instr2: 4
Instr3: 6
Effet2: 102
Instr4: 7
```

Utilisation des Instruments Nommés

On peut appeler les instruments nommés en utilisant le nom entre guillements à la place du numéro d'instrument (note : le caractère '+' doit être omis). Actuellement (depuis Csound 4.22.4), les instruments nommés sont supportés par :

- les évènements de partition 'i' et 'q'

Notes

1. dans les fichiers de partition, il faut éviter les guillemets non appariés, et les espaces et autres caractères illégaux dans les chaînes, sinon (au moins dans la version actuelle) un comportement imprévisible peut apparaître (ce problème n'existe pas pour les évènements en ligne -L). Cependant, il y a un test pour détecter les instruments non définis, et dans ce cas, l'évènement est simplement ignoré avec un avertissement.

2. Les utilitaires autonomes (scsort et extract) ne supportent pas les instruments nommés. Il est toujours possible de trier de telles partitions en utilisant l'option -t0 de l'exécutable Csound.

- les évènement temps-réel en ligne (-L)
- les opcodes event, schedkwhen, subinstr, et subinstrinit
- les opcodes massign, pgmassign, prealloc, et mute

De plus, il y a un nouvel opcode (nstrnum) qui retourne le numéro d'un instrument nommé :

```
insno nstrnum "nom"
```

Dans l'exemple ci-dessus, nstrnum "Effet1" retournerait 101. S'il n'existe aucun instrument avec le nom spécifié, une erreur d'initialisation est levée et -1 est retourné.

Exemple

```
; ---- orchestre ----

sr      =  44100
ksmps   =  10
nchnls  =  1

        prealloc "SineWave", 20
        prealloc "MIDISineWave", 20

        massign 1, "MIDISineWave"

gaOutSend       init 0

        instr +OutputInstr

        out gaOutSend
        clear gaOutSend

        endin

        instr SineWave

a1      oscils p4, p5, 0
        vincr gaOutSend, a1

        endin

        instr MIDISineWave

iamp    veloc
inote   notnum
icps    =  cpsoct(inote / 12 + 3)
a1      oscils iamp * 100, icps, 0
        vincr gaOutSend, a1

        endin

; ---- partition ----

i "SineWave" 0 2 12000 440
i "OutputInstr" 0 3
e
```

Auteur

Istvan Varga

2002

Opcodes Définis par l'Utilisateur (UDO)

Csound permet la définition d'opcodes dans l'en-tête de l'orchestre au moyen des opcodes *opcode* et *endop*. L'opcode défini peut fonctionner avec un nombre d'échantillons par période de contrôle (*ksmps*) différent en utilisant *setksmps*.

Pour connecter les entrées et les sorties de l'UDO, on utilise *xin* et *xout*.

Un UDO ressemble à ceci :

```
        opcode Lowpass, a, akk

        setksmps 1              ; nécessite sr=kr
```

```
ain, ka1, ka2   xin              ; lire les paramètres d'entrée
aout    init 0                   ; initialiser la sortie
aout    =  ain*ka1 + aout*ka2    ; filtre simple comme tone
        xout aout                ; écrire la sortie

        endop
```

Cet UDO appelé *Lowpass* reçoit trois entrées (la première au taux-a, et les deux autres au taux-k), et délivre une sortie au taux-a. Noter l'utilisation de *xin* pour recevoir les entrées et de *xout* pour délivrer les sorties. Noter aussi l'utilisation de *setksmps*, qui est nécessaire pour que le filtre fonctionne correctement.

Pour utiliser cet UDO depuis un instrument, on écrirait quelque chose comme :

```
afiltre Lowpass asource, kvaleur1, kvaleur2
```

voir l'entrée *opcode* pour des informations détaillées sur la définition d'UDO.

Vous pouvez trouver plusieurs UDO déjà rédigés (ou apporter votre propre contribution) à *User Defined Opcode Database* [http://www.csounds.com/udo/] sur *Csounds.com* [http://www.csounds.com/].

La Partition Numérique Standard

La section de la partition contient des évènements qui démarrent des instances d'instruments de l'orchestre. La partition propose diverses instructions qui permettent l'élaboration de partitions complexes avec le langage de csound.

Actuellement, la longueur maximale de la partition est de 2^{31}-1 périodes de contrôle. Par exemple, avec kr=1500, on peut exécuter une partition pendant une période maximale de 16,5 jours avant l'apparition de problèmes provoqués par un dépassement des variables entières signées sur 32 bit.

Il faut noter également que lorsque l'on utilise des nombres flottants en simple précision (c-à-d les installeurs 'f' plutôt que les 'd'), la précision temporelle se détériore après une longue durée d'exécution.

Prétraitement des Partitions Standard

Une *Partition* (un ensemble d'instructions de partition) se divise en sections ordonnées dans le temps par l'*instruction s*. Avant sa lecture par l'orchestre, une partition est prétraitée section par section. Chaque section est normalement traitée par trois routines : *Carry* (report de valeur), *Tempo*, et *Sort* (tri).

Carry

Dans un groupe d'*instructions i* consécutives dont les nombres entiers p1 sont indentiques, tout p-champ non rempli prendra la même valeur que celle du p-champ correspondant dans l'instruction précédente. Un p-champ vide peut-être marqué par un point (.) entouré d'espaces. Il n'y a pas besoin de point après le dernier p-champ non vide. La sortie du prétraitement Carry montre explicitement les valeurs reportées. La Fonction Carry n'est pas affectée par les commentaires rencontrés ou les lignes blanches ; elle s'arrête seulement lorsqu'elle rencontre une instruction autre que l'*instruction i* ou une *instruction i* avec un nombre entier p1 différent.

Il y a trois fonctions supplémentaires, pour p2 seulement : +, ^+*x*, et ^-*x*. Le symbole + en p2 recevra la valeur de p2 + p3 de l'instruction i précédente. Cela permet de déterminer automatiquement l'instant du début d'une note à partir de la somme des durées précédentes. Le symbole + peut lui-même être reporté. Il n'est autorisé que dans p2. Par exemple : les instructions

```
i1   0    .5    100
i .  +
i
```

se transformeront en

```
i1   0    .5    100
i1   .5   .5    100
i1   1    .5    100
```

Les symboles ^+*x* et ^-*x* determinent la valeur de p2 en additionnant ou en soustrayant respectivement la valeur *x* du p2 précédent. Ils ne peuvent être utilisés qu'en p2 et ne sont *pas* reportés comme le symbole +. Noter aussi qu'il ne doit pas y avoir d'espaces après la partie ^, + ou - de ces symboles -- le nombre doit suivre directement comme dans ^+2.3. Si l'exemple ci-dessus avait été

```
i1   0    .5        100
i .  ^+1
```

```
i .  ^+1
```

le résultat aurait été

```
i1   0    .5        100
i1   1    .5        100
i1   2    .5        100
```

On peut se servir largement de la fonction Carry. Son utilisation, spécialement dans les grandes partitions, peut réduire grandement la frappe au clavier et elle simplifiera les modifications ultérieures.

Il y a des circonstances où l'on ne veut pas que les p-champs "manquants" après le dernier qui a été entré soient implicitement reportés. Par exemple dans un instrument prévu pour prendre un nombre variable de p-champs. A partir de Csound 5.08, on peut empêcher le report implicite des p-champs à la fin d'une instruction i en utilisant le symbole ! (appelé le "symbol de non-report"). Le ! doit apparaître à la fin d'une instruction i et il ne peut pas être utilisé en p1, p2 ou p3, car ces p-champs sont obligatoires. Voici un exemple :

```
i1   0    .5        100
i .  +
i .  .    .         !
i
```

Cette partition sera interprétée comme ceci

```
i1   0    .5        100
i1   .5   .5        100
i1   1    .5              ; no p4
i1   1.5  .5              ; only p1 to p3 are carried here
```

Tempo

Cette opération modifie l'information temporelle d'une section de partition selon les directives de l'*instruction t*. L'opération tempo convertit p2 (et pour les *instructions i*, p3) de la valeur originale en pulsations vers des secondes réelles, celles-ci étant les unités temporelles requises par l'orchestre. Après la modification temporelle, les fichiers partitions apparaîtront dans un format lisible par l'orchestre comme ceci :

i p1 p2pulsations p2secondes p3pulsations p3secondes p4 p5 ...

Sort

Cette routine trie toutes les instructions d'action temporelle chronologiquement selon la valeur de p2. Elle place aussi les évènements simultanés par ordre de priorité. Chaque fois qu'une *instruction f* et une *instruction i* ont la même valeur en p2, l'*instruction f* sera placée en premier. Chaque fois que plusieurs *instructions i* ont la même valeur en p2, elles seront triées par ordre croissant de leur valeur en p1. Si elles ont aussi la même valeur en p1, elles seront triées par ordre croissant de leur valeur en p3. Le tri de la partition est effectué par section (voir l'*instruction s*). Ce tri automatique permet d'écrire les instructions de partition dans n'importe quel ordre à l'intérieur d'une section.

Note

Les opérations Carry, Tempo et Sort sont combinées dans une seule passe en trois phases sur le fichier de partition, pour produire un nouveau fichier dans un format lisible par l'orchestre (voir l'exemple de Tempo). Ce traitement peut être invoqué explicitement par la commande *Scsort*, ou implicitement par Csound qui traite la partition avant d'appeler l'orchestre. Les fichiers en format source et en format lisible par l'orchestre sont encodés en caractères ASCII, et peuvent être consultés ou modifiés dans un éditeur de texte standard. L'utilisateur peut écrire ses propres routines pour modifier les fichiers de partition avant ou après le processus décrit ci-dessus, pourvu que le format final lisible par l'orchestre soit respecté. Les sections de formats différents peuvent être traitées séquentiellement par lots ; et les sections de même format peuvent être réunies pour le tri automatique.

Instructions de Partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des évènements de la partition
- *x* - Ignore le reste de la section courante
- *{* - Commence une boucle imbricable ne délimitant pas de section
- *}* - Termine une boucle imbricable ne délimitant pas de section

Symboles Next-P et Previous-P

A la fin de chacune des opération *Carry*, *Tempo*, et *Sort*, trois fonctions de partition supplémentaires sont interprétées durant l'écriture du fichier : next-p, previous-p, et *ramping*.

Les p-champs d'une *instruction i* contenant les symboles *npx* ou *ppx* (où *x* est un entier) seront remplacés par la valeur du p-champ approprié de l'instruction i suivante (ou de l'instruction i précédente) ayant le

même p1. Par exemple, le symbole *np*7 sera remplacé par la valeur du p7 de la note suivante devant être jouée par le même instrument. Les symboles *np* et *pp* sont récursifs et peuvent référencer d'autres symboles *np* et *pp* qui peuvent en référencer d'autres, etc. Les références doivent se terminer par un nombre réel ou un *symbole ramp*. Il faut éviter les références en boucle fermée. Les symboles *np* et *pp* sont interdits en p1, p2 et p3 (bien qu'ils puissent référencer ces derniers). Les symboles *np* et *pp* peuvent être reportés (Carry). Les référence de *np* et de *pp* ne peuvent traverser une limite de Section. Toute référence avant ou arrière à une instruction de note inexistante recevra la valeur zéro.

Par exemple : les instructions

```
i1   0    1    10   np4  pp5
i1   1    1    20
i1   1    1    30
```

se transformeront en

```
i1   0    1    10   20   0
i1   1    1    20   30   20
i1   2    1    30   0    30
```

Les symboles *np* et *pp* peuvent apporter à un instrument une connaissance contextuelle de la partition, ce qui permettra de réaliser un glissando ou un crescendo, par exemple, vers la hauteur ou l'intensité d'un évènement futur (qui peut être immédiatement adjacent ou non). A noter que bien que la fonction *Carry* propage *np* et *pp* vers des instructions non triées, l'opération d'interprétation de ces symboles se fait sur une version de la partition résolue temporellement et complètement triée.

Ramping

Les p-champs d'une *instruction i* contenant le symbole < seront remplacés par des valeurs issues de l'interpolation linéaire d'une pente temporelle. Les pentes sont attachées à chaque extrémité au premier nombre réel trouvé dans le même p-champ de notes précédentes et suivantes jouées par le même instrument. Par exemple : les instructions

```
i1   0    1    100
i1   1    1    <
i1   2    1    <
i1   3    1    400
i1   4    1    <
i1   5    1    0
```

se transformeront en

```
i1   0    1    100
i1   1    1    200
i1   2    1    300
i1   3    1    400
i1   4    1    200
i1   5    1    0
```

Les pentes ne peuvent pas traverser une limite de Section. Les pentes ne peuvent pas être attachées à un symbole *np* ou *pp* (mais elles peuvent être référencées par ceux-ci). Les symboles de pente sont interdits en p1, p2 et p3. Les symboles de pente peuvent être reportés. A noter cependant que, bien que la fonction *Carry* propage les symboles de pente vers des instructions non triées, l'opération d'interprétation de

ces symboles se fait sur une version de la partition résolue temporellement et complètement triée. En fait, l'interpolation linéaire temporelle est basé sur le temps de partition résolu, de façon à ce qu'une pente couvrant un groupe de notes accelerando reste linéaire par rapport au temps strictement chronologique.

A partir de la version 3.52 de Csound, l'utilisation des symboles (ou) donne une pente d'interpolation exponentielle, comme *expon*. L'utilisation du symbole ˜ donnera une distribution aléatoire uniforme entre la première et la dernière valeur de la pente. L'utilisation de ces fontions suit les mêmes règles que la fonction de pente linéaire.

Macros de Partition

Description

Les macros sont des substitutions de texte qui sont réalisées dans la partition lors de sa présentation au système. Le système de macro de Csound est très simple, et il utilise les caractères # et $ pour définir et appeler des macros. C'est un moyen de simplifier l'écriture d'une partition, et une alternative élémentaire aux systèmes de génération de partition complète. Le système de macros de partition est similaire, mais de façon indépendante, au système de macros du langage de l'orchestre.

#define NOM -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut être une combinaison de lettres et de nombres. La casse est significative. Cette forme est restrictive dans le sens que les noms de variable sont fixes. On peut obtenir plus de souplesse au moyen d'une macro avec arguments, décrite ci-dessous.

#define NOM(*a' b' c'*) -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut être suivi par une combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont remplacés par la forme : $A. En fait, les arguments sont implémentés comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et leur nom peut être n'importe quel choix de lettres. Rappelez-vous que la casse est significative dans les noms de macro.

$NOM. -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé d'un caractère $. Le nom se termine par le premier caractère qui n'est ni une lettre ni un chiffre. Si on ne veut pas terminer le nom par un espace, on peut utiliser un point qui sera ignoré. La chaîne, *$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut aussi contenir des appels de macro.

#undef NOM -- rend un nom de macro indéfini. Si l'on a plus besoin d'une macro, on peut la rendre indéfinie avec *#undef* NOM.

Syntaxe

```
#define NOM # texte de substitution #

#define NOM(a' b' c') # texte de substitution #

$NOM.

#undef  NOM
```

Initialisation

 # texte de substitution # -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est délimité par des caractères #, ce qui per-

met d'éviter l'insertion de caractères supplémentaires par inadvertance.

Exécution

Il faut prendre quelques précautions avec les macros de substitution de texte, car elle peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

Une Autre Utilisation des Macros. Lorsque l'on écrit une partition complexe, on oublie parfois trop facilement à quoi les différents numéros d'instruments font référence. On peut utiliser des macros pour nommer ces nombres. Par exemple

```
#define Flute  #i1#
#define Whoop  #i2#

$Flute.  0  10  4000  440
$Whoop.  5  1
```

Exemples

Exemple 1. Macro Simple

Une note a un ensemble de p-champs qui sont répétés :

```
#define ARGS # 1.01 2.33 138#
i1 0 1 8.00  1000 $ARGS
i1 0 1 8.01  1500 $ARGS
i1 0 1 8.02  1200 $ARGS
i1 0 1 8.03  1000 $ARGS
```

Ce sera développé avant le tri en :

```
i1 0 1 8.00  1000 1.01 2.33 138
i1 0 1 8.01  1500 1.01 2.33 138
i1 0 1 8.02  1200 1.01 2.33 138
i1 0 1 8.03  1000 1.01 2.33 138
```

On économise ainsi de la frappe au clavier, et les révisions sont plus faciles. Avec deux ensembles de p-champs on pourrait avoir une seconde macro (il n'y pas de réelle limite au nombre de macros que l'on peut définir).

```
#define ARGS1 # 1.01 2.33 138#
#define ARGS2 # 1.41 10.33 1.00#
i1 0 1 8.00  1000 $ARGS1
i1 0 1 8.01  1500 $ARGS2
i1 0 1 8.02  1200 $ARGS1
i1 0 1 8.03  1000 $ARGS2
```

Exemple 2. Macros avec arguments

```
#define ARG(A) # 2.345   1.03   $A   234.9#
i1 0 1 8.00 1000 $ARG(2.0)
i1 + 1 8.01 1200 $ARG(3.0)
```

qui se développe en

```
i1 0 1 8.00 1000 2.345   1.03   2.0   234.9
i1 + 1 8.01 1200 2.345   1.03   3.0   234.9
```

Crédits

Auteur : John ffitch

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

Partition dans Plusieurs Fichiers

Description

Disposer la partition dans plusieurs fichiers.

Syntaxe

```
#include "nomfichier"
```

Exécution

Il est parfois commode de disposer la partition dans plusieurs fichiers. On peut le faire en utilisant *#include* qui fait partie du système de macro. Par une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère *"* peut être remplacé par n'importe quel caractère adéquat. Pour la plupart des usages, le symbole des guillemets sera probablement le plus adapté. Le nom de fichier peut comprendre un nom de chemin complet.

On prend en entrée le contenu du fichier nommé, puis on revient à l'entrée précédente. La profondeur des fichiers inclus et des macros est actuellement limitée à 20.

On peut utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur. On peut aussi l'utiliser pour répéter des sections.

```
s
#include :section1:
;; Répéter ceci
s
#include :section1:
```

Pour d'autres méthodes de répétition, utiliser l'*instruction r*, l'*instruction m*, et l'*instruction n*.

Crédits

Auteur : John ffitch

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

Merci à Luis Jure d'avoir relevé la syntaxe incorrecte dans l'instruction d'inclusion de fichiers.

Evaluation des Expressions

Dans les anciennes versions de Csound les nombres présents dans une partition étaient utilisés tels quels. Dans certains cas, une évaluation simple serait plus facile. Ce besoin est accru s'il y a des macros. Pour y arriver, on a introduit la syntaxe des expressions arithmétiques entre crochets [ ]. On peut utiliser des expressions avec les opérations +, -, *, /, % ("modulo"), et ^ ("élévation à une puissance"), les groupements se faisant par parenthèses ( ). Les signes unaires plus et moins sont aussi supportés. Les expressions peuvent inclure des nombres et, naturellement, des macros dont la valeur est une chaîne numérique ou arithmétique. Tous les calculs sont faits en nombres en virgule flottante. Les règles de précédence usuelles sont suivies lors de l'évaluation : les expressions entre parenthèse () sont évaluées en premier et ^ est évalué avant *, /, et % qui sont évalués avant + et -.

En plus des opérations arithmétiques, les opérateurs logiques bit à bit suivants sont aussi disponibles : & (ET), | (OU), et # (OU exclusif). Ces opérateurs arrondissent leurs opérandes à l'entier (long) le plus proche avant l'évaluation. Les opérateurs logiques ont la même précédence que les opérateurs arithmétiques *, /, et %.

On a ajouté dans la version 3.56 de Csound @*x* (la première puissance de deux supérieure ou égale à *x*) et @@*x* (la première puissance de deux plus un supérieure ou égale à *x*).

Finalement, on peut utiliser le symbole tilde ~ dans une expression chaque fois qu'un nombre est permis. Chaque ~ sera remplacé par un nombre aléatoire compris entre zéro (0) et un (1).

Exemple

```
r3  CNT

i1  0  [0.3*$CNT.]
i1  +  [($CNT./3)+0.2]

e
```

Comme les trois copies de la section comprennent la macro $CNT. avec les valeurs successives 1, 2 et 3, le développement est

```
s
i1  0  0.3
i1  0.3  0.533333
s
i1  0  0.6
i1  0.6  0.866667
s
i1  0  0.9
i1  0.9  1.2
e
```

C'est une forme extrême, mais on peut aussi utiliser le système d'évaluation pour répéter des sections avec des différences subtiles.

Voici quelques exemples simples de chaque opérateur :

```
i1  0   1   [ 110 + 220 ]       ; evaluates to 330
i1  +   .   [ 330 - 55 ]        ; 275
i1  +   .   [ 44 * 10 ]         ; 440
i1  +   .   [ 1100 / 2 ]        ; 550
i1  +   .   [ 5 ^ 4 ]           ; 625
i1  +   .   [ 5660 % 1000 ]     ; 660
i1  +   .   [ 110 & 220 ]       ; 76
i1  +   .   [ 110 | 220 ]       ; 254
i1  +   .   [ 110 # 220 ]       ; 178
i1  +   .   [~]                 ; random between 0-1
i1  +   .   [~ * 4 + 1]         ; random between 1-5
i1  +   .   [~ * 95 + 5]        ; random between 5-100

i1  +   .   [ 8 / 2 * 3 ]       ; 12
i1  +   .   [ 4 + 3 - 2 + 1 ]   ; 6
i1  +   .   [ 4 + 3 * 2 + 1 ]   ; 11
i1  +   .   [(4 + 3)*(2 + 1)]   ; 21

i1  +   .   [ 2 * 2 & 3 ]       ; 4
i1  +   .   [ 3 & 2 * 2 ]       ; 0
i1  +   .   [ 4 | 3 * 3 ]       ; 13
```

Crédits

Auteur : John ffitch

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

Chaînes de caractères dans les p-champs

On peut passer une chaîne de caractères dans un p-champ au lieu d'un nombre, comme ceci :

```
i 1 0 10 "A4"
```

Cette chaîne de caractères peut être reçue par l'instrument et traitée par les *opcodes de chaîne de caractères*.

Note

Actuellement un seul p-champ peut contenir une chaîne de caractères (c-à-d qu'on n'autorise pas plus d'une chaîne de caractères par ligne). On peut contourner ceci en utilisant *strset* et *strget*.

Frontaux

Voici une liste (non exhaustive) des frontaux disponibles pour Csound.

Csound5GUI

Csound5GUI est une interface utilisateur graphique (GUI) multi plates-formes, polyvalente qui fait partie de la distribution standard de Csound. Elle implémente la plupart des options de configuration de Csound.

CSDplayer

C'est un simple programme java pour jouer des fichiers csd. Il est inclus dans la distribution standard.

Winsound

Egalement présent dans l'arborescence principale de Csound (bien qu'absent de certaines distributions), Winsound est un portage multi plates-formes en FLTK du frontal original de Barry Vercoe pour Csound.

WinXoundPro

Un frontal commmode pour windows avec coloration syntaxique. On peut l'obtenir à WinXsound Front Page [http://www.ibiart.it/winxound/index.html].

Csound Editor

Un frontal commmode pour windows avec coloration syntaxique. On peut l'obtenir à Flavio Tordini's Home Page [http://flavio.tordini.org/csound-editor/].

MacCsound

Plus qu'un frontal pour le Mac à MacCsound Page [http://www.csounds.com/matt/MacCsound/].

Cabel

Cabel est une interface utilisateur graphique pour construire des instruments csound en interconnectant des modules similaires aux modules des synthétiseurs. Multi plates-formes, écrit en Python. A http://cabel.sourceforge.net/.

Blue

Frontal orienté composition, écrit en Java. Son interface ressemble beaucoup à un multipiste numérique, mais en diffère en intégrant des axes temporels dans des axes temporels (polyObjects). Cela permet une organisation compositionnelle qui me semble très intuitive, instructive et flexible. Téléchargeable à : Blue Home Page [http://csounds.com/stevenyi/blue/].

CsoundAC

Programmation Python

Vous pouvez utiliser CsoundAC comme un module d'extension de Python. Vous pouvez faire cela dans un interpréteur Python standard tel que la ligne de commande Python ou le Idle Python GUI.

Pour utiliser CsoundAC dans un interpréteur Python standard, importez CsoundAC.

```
import CsoundAC
```

Le module CsoundAC crée automatiquement une instance de CppSound nommée csound, qui fournit une interface orientée objet à l'API de Csound. Dans un interpréteur Python standard, vous pouvez charger un fichier Csound `.csd` et l'exécuter de cette manière :

```
C:\Documents and Settings\mkg>python
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import CsoundAC
>>> csound.load("c:/projects/csound5/examples/trapped.csd")
1
>>> csound.exportForPerformance()
1
>>> csound.perform()
BEGAN CppSound::perform(5, 988ee0)...
BEGAN CppSound::compile(5, 988ee0)...
Using default language
0dBFS level = 32767.0
Csound version 5.00 beta (float samples) Jun  7 2004
libsndfile-1.0.10pre6
orchname:  temp.orc
scorename: temp.sco
orch compiler:
398 lines read
instr   1
instr   2
instr   3
instr   4
instr   5
instr   6
instr   7
instr   8
instr   9
instr   10
instr   11
instr   12
instr   13
instr   98
instr   99
sorting score ...
... done
Csound version 5.00 beta (float samples) Jun  6 2004
displays suppressed
0dBFS level = 32767.0
orch now loaded
audio buffered in 16384 sample-frame blocks
SFDIR undefined.  using current directory
writing 131072-byte blks of shorts to test.wav
WAV
SECTION 1:
ENDED CppSound::compile.
ftable 1:
ftable 2:
ftable 3:
ftable 4:
ftable 5:
ftable 6:
ftable 7:
ftable 8:
ftable 9:
ftable 10:
ftable 11:
ftable 12:
ftable 13:
ftable 14:
ftable 15:
ftable 16:
ftable 17:
```

```
ftable 18:
ftable 19:
ftable 20:
ftable 21:
ftable 22:
new alloc for instr 1:
B  0.000 ..  1.000 T  1.000 TT  1.000 M:     32.7      0.0
new alloc for instr 1:
B  1.000 ..  3.600 T  3.600 TT  3.600 M:    207.6      0.1

...

B 93.940 .. 94.418 T 98.799 TT281.799 M:    477.6     85.0
B 94.418 ..100.000 T107.172 TT290.172 M:    118.9     11.5
end of section 4         sect peak amps:  25950.8  26877.4
inactive allocs returned to freespace
end of score.              overall amps:  32204.8  31469.6
overall samples out of range:        0        0
0 errors in performance
782 131072-byte soundblks of shorts written to test.wav WAV
Elapsed time = 13.469000 seconds.
ENDED CppSound::perform.
1
>>>
```

Le script `koch.py` montre comment utiliser Python pour faire une composition algorithmique pour Csound. Vous pouvez utiliser des chaînes de caractères litérales à triples guillemets pour incorporer vos fichiers Csound directement dans votre script, et les assigner à Csound :

```
csound.setOrchestra('''sr = 44100
kr = 441
ksmps = 100
nchnls = 2
0dbfs = .1
instr 1,2,3,4,5 ; FluidSynth General MID
I; INITIALIZATION
; Channel, bank, and program determine the preset, that is, the actual sound.
ichannel         =                 p1
iprogram         =                 p6
ikey                 =                         p4
ivelocity                =                         p5 + 12
ijunk6                        =                         p6
ijunk7                 =                 p7
; AUDIO
istatus                =                 144;
print                iprogram, istatus, ichannel, ikey, ivelocityaleft, aright
fluid                "c:/projects/csound5/samples/VintageDreamsWaves-v2.sf2", \\
iprogram, istatus, ichannel, ikey, ivelocity, 1
outs                         aleft, arightendin''')
csound.setCommand("csound --opcode-lib=c:/projects/csound5/fluid.dll \\
-RWdfo ./koch.wav ./temp.orc ./temp.sco")
csound.exportForPerformance()
csound.perform()
```

CsoundVST

CsoundVST est un frontal multi-fonction pour Csound, basé sur l'API de Csound. CsoundVST s'exécute comme une interface utilisateur graphique autonome pour Csound, et il s'exécute aussi comme un instrument VST ou un plugin d'effet dans des hôtes VST tels que Cubase avec la même interface utilisateur. CsoundVST fait partie de l'arbre principal des sources de Csound, mais il n'est pas inclus dans les distributions standard à cause des limitations de la license du SDK VST de Steinberg.

Utilisation autonome

Pour lancer CsoundVST comme frontal autonome pour Csound, exécutez CsoundVST. Au démarrage du programme, vous verrez une interface graphique utilisateur avec une rangée de boutons en haut. Cliquez sur le bouton *Open...* pour charger un fichier `.csd`. Vous pouvez aussi cliquer sur le bouton *Open...* et charger un fichier `.orc`, cliquez ensuite sur le bouton *Import...* pour ajouter un fichier `.sco`. Vous pou-

vez éditer la commande de Csound, le fichier orchestre, ou le fichier partition dans les onglets respectifs de l'interface utilisateur. Quand tout est prêt, cliquez sur le bouton *Perform* pour lancer Csound. Vous pouvez arrêter une exécution à n'importe quel moment en cliquant sur le bouton *Stop*.

Plugin VST

Les instructions suivantes sont pour Cubase 4.0. Des procédures à peu près similaires seraient utilisées dans d'autres programmes hôtes.

Utilisez le menu *Devices*, la boîte de dialogue *Plug-In Information*, l'onglet *VST Plug-Ins*, la boîte de dialogue *VST 2.x Plug-in Paths*, le bouton *Add* pour ajouter votre répertoire `csound/bin` au chemin des plugins de Cubase. Vous pouvez avoir plusieurs répertoires séparés par des points-virgules. Sélectionnez ensuite le chemin de CsoundVST et cliquez sur le bouton *Set as Shared Folder*

Quittez Cubase, et redémarrez-le.

Utilisez le menu *File*, la boîte de dialogue *New Project* pour créer un nouveau morceau (song).

Utilisez le menu *Project*, le sous-menu *Add Track*, pour ajouter une nouvelle piste MIDI.

Utilisez l'outil crayon pour dessiner un *Part* de quelques mesures sur la piste. Ecrivez un peu de musique dans le *Part* à l'aide de l'éditeur *Event* ou de l'éditeur *Score*.

Utilisez le menu *Devices* (ou la touche F11) pour ouvrir la boîte de dialogue *VST Instruments*.

Cliquez sur une des étiquettes *No VST Instrument*, et sélectionnez *CsoundVST* dans la liste qui apparaît.

Cliquez sur le bouton *e* (pour edit) pour ouvrir la boîte de dialogue de *CsoundVST*.

Sur la page des Réglages, cochez la case *Instrument* dans le groupe VST Plugin, et la case *Classic* dans le groupe *Csound performance mode*. Cliquez ensuite sur le bouton *Apply*.

Cliquez sur le bouton *Open* pour faire apparaître la boîte de dialogue de sélection de fichier. Naviguez vers un répertoire contenant un fichier csd Csound adéquat pour une exécution MIDI, tel que `csound/examples/CsoundVST.csd`. Cliquez sur le bouton OK pour charger le fichier. Vous pouvez aussi ouvrir et importer des fichiers `.orc` et `.sco` adéquats comme décrit ci-dessus.

Dans tous les cas, la ligne de commande dans le champ texte *Classic Csound command line* doit spécifier `-+rtmidi=null -M0`, et devrait ressembler à ceci :

```
csound -f -h -+rtmidi=null -M0 -d -n -m7 --midi-key-oct=4 --midi-velocity=5 temp.orc temp.sco
```

Cliquez sur le bouton on/off de la boîte de dialogue *VST Instruments* pour l'allumer. Ceci devrait compiler l'orchestre Csound.

Dans l'*Inspecteur de Piste de Cubase*, cliquez sur l'étiquette *out: Not Assigned* et sélectionnez *CsoundVST* dans la liste qui apparaît.

Sur la règle en haut de la fenêtre *Arrangement*, sélectionnez le point de fin de boucle et tirez-le jusqu'à la fin de votre part, cliquez ensuite sur le bouton loop pour activer la mise en boucle.

Cliquez sur le bouton *play* de la barre de *Transport*. Vous devriez entendre votre musique jouée par CsoundVST.

Essayez d'assigner votre piste à différents canaux ; un instrument Csound différent jouera chaque canal.

Quand vous sauvegardez votre song, votre orchestre Csound sera sauvegardé comme une partie du song et rechargé quand vous rechargerez le song.

Vous pouvez cliquer sur l'onglet *Orchestra* et éditer vos instruments Csound pendant que CsoundVST est en train de jouer. Pour entendre vos changements, il suffit de cliquer sur le bouton CsoundVST *Perform* pour recompiler l'orchestre.

Vous pouvez assigner jusqu'à 16 canaux à un seul plugin CsoundVST.

TclCsound

TclCsound fut introduit pour fournir une interface simple de scripting à Csound. Tcl est un langage simple aisément extensible et qui facilite des opérations comme l'accès aux fichiers et la mise en réseau sous TCP. Avec son composant Tk, il peut aussi gérer une interface graphique pilotée par évènements. TclCsound donne trois "points de contact" avec Tcl :

1. un interpréteur tcl connaissant csound (cstclsh)

2. un shell de fenêtrage connaissant csound (cswish)

3. un module de commandes csound pour Tcl/Tk (bibliothèque dynamique tclcsound)

L'interpréteur Tcl : cstclsh

Avec cstclsh, on peut contrôler de manière interactive une exécution csound. La commande démarre un shell interactif, qui maintient une instance de Csound. On peut ensuite utiliser plusieurs commandes pour la contrôler. Par exemple, la commande suivante peut compiler du code csound et le charger en mémoire, prêt à être exécuter :

```
csCompile -odac orchestre partition -m0
```

Ceci fait, on peut démarrer l'exécution de deux manières : avec csPlay ou avec csPerform. La commande

```
csPlay
```

démarrera l'exécution Csound dans un thread séparé et retournera à l'invite de cstclsh. On peut utiliser ensuite plusieurs commandes pour contrôler Csound. Par exemple,

```
csPause
```

suspendra l'exécution ; et

```
csRewind
```

reviendra au début de la liste de notes. On peut utiliser les commandes csNote, csTable et csEvent pour ajouter des évènements de partition pendant l'exécution, à la volée. La commande csPerform, à l'inverse de csPlay, ne lancera pas un thread séparé, mais démarrera Csound dans le même thread, ne retournant que quand l'exécution est finie. Il existe une variété d'autres commandes, donnant un contrôle total de Csound.

Cswish: le shell de fenêtrage

Avec Cswish, on peut utiliser des commandes et des contrôleurs graphiques Tk pour se doter d'une interface graphique avec gestion d'évènements. Comme pour cstclsh, le lancement de la commande cswish ouvre aussi un shell interactif. Par exemple, on peut utiliser les commandes suivantes pour créer un panneau de transport pour Csound :

```
frame .fr
button .fr.play -text play -command csPlay
button .fr.pause -text pause -command csPause
button .fr.rew -text rew -command csRewind
pack .fr .fr.play .fr.pause .fr.rew
```

De même, on peut lier des touches à des commandes afin d'utiliser le clavier de l'ordinateur pour jouer

avec Csound.

Les commandes de contrôle de canal fournies par TclCsound sont particulièrement utiles. Par exemple, on peut enregistrer des canaux d'E/S nommés avec TclCsound et les utiliser avec les opcodes invalue et outvalue. De plus, l'API de Csound fournit aussi un bus logiciel complet pour les canaux audio, de contrôle et de chaînes. Dans TclCsound, on peut accéder aux canaux du bus de contrôle et de chaînes (le bus audio n'est pas implémenté, car Tcl n'est pas capable de traiter ce genre de données). Avec ces commandes de TclCsound, on peut connecter facilement des contrôleurs graphiques Tk aux paramètres de synthèse.

Un serveur Csound

Dans Tcl, il est très simple de configurer des connexions réseau TCP. On peut construire un serveur csound avec quelques lignes de code. Celui-ci peut accepter des connexions depuis la machine locale ou depuis des clients distants. Non seulement les clients Tcl/Tk peuvent lui envoyer des commandes, mais des connexions TCP peuvent être établies depuis un autre logiciel, comme par exemple, Pure Date (PD). On montre ci-dessous un script Tcl qui peut être lancé dans l'interpréteur standard tclsh. Il utilise le module Tclcsound, une bibliothèque dynamique qui ajoute les commandes de l'API de Csound à Tcl.

```
# load tclcsound.so
#(OSX: tclcsound.dylib, Windows: tclcsound.dll)
load tclcsound.so Tclcsound
set forever 0

# This arranges for commands to be evaluated
proc ChanEval { chan client } {
if { [catch { set rtn [eval [gets $chan]]} err] } {
puts "Error: $err"
} else {
puts $client $rtn
flush $client
}
}



# this arranges for connections to be made



proc NewChan { chan host port } {
puts "Csound server: connected to $host on port $port ($chan)"
fileevent $chan readable [list ChanEval $chan $host]
}



# this sets up a server to listen for
# connections



set server [socket -server NewChan 40001]
set sinfo [fconfigure $server -sockname]
puts "Csound server: ready for connections on port [lindex $sinfo 2]"
vwait forever
```

Lorsque le serveur est actif, il est alors possible de configurer des clients pour contrôler le serveur Csound. On peut lancer de tels clients depuis des interpréteurs Tcl/Tk standard, car ils n'évaluent pas eux-mêmes les commandes Csound. Voici un exemple de connexions client à un serveur Csound au moyen de Tcl :

```
# connect to server
set sock [socket localhost 40001]

# compile Csound code
puts $sock "csCompile -odac orchestra score"
```

```
flush $sock


# start performance
puts $sock "csPlay"
flush $sock


# stop performance
puts $sock "csStop"
flush $sock
```

Comme il est mentionné ci-dessus, on peut configurer des clients utilisant d'autres systèmes logiciels, tels que PD. De tels clients n'ont besoin que de se connecter au serveur (au moyen d'un objet netsend) et de lui envoyer des messages. Le premier élément de chaque message est une commande. D'autres éléments facultatifs peuvent y être ajoutés comme arguments de cette commande.

Un Environnement de Scripting

Avec TclCsound, on peut transformer le populaire éditeur de texte emacs en environnement de scripting et d'exécution de Csound. Lorqu'il est en mode Tcl, l'éditeur permet d'évaluer des expressions Tcl par sélection et utilisation d'une simple séquence d'échappement (Ctrl-C Ctrl-X). Grâce à cela, on peut éditer et exécuter du code Csound et Tcl/Tk de façon intégrée

Dans Tcl il est possible d'écrire des fichiers de partition et d'orchestre qui peuvent être sauvegardés, compilés et exécutés par le même script, sous l'environnement emacs. L'exemple suivant montre un script Tcl qui construit un instrument csound et lance ensuite une exécution de csound. Il crée 10 oscillateurs en parallèle légèrement désaccordés, ce qui génère des sons semblables à ceux que l'on trouve dans *Inharmonique* de Risset.

```
load tclcsound.so Tclcsound


# set up some intermediary files



set orcfile "tcl.orc"
set scofile "tcl.sco"
set orc [open $orcfile w]
set sco [open $scofile w]


# This Tcl procedure builds an instrument
proc MakeIns { no code } {
global orc sco
puts $orc "instr $no"
puts $orc $code
puts $orc "endin"
}



# Here is the instrument code
append ins "asum init 0 \n"
append ins "ifreq = p5 \n"
append ins "iamp = p4 \n"



for { set i 0 } { $i < 10 } { incr i } {
append ins "a$i oscili iamp,
ifreq+ifreq*[expr $i * 0.002], 1\n"
}


for { set i 0 } {$i < 10 } { incr i } {
if { $i } {
append ins " + a$i"
} else {
```

```
append ins "asum = a$i "
}
}


append ins "\nk1 linen 1, 0.01, p3, 0.1 \n"
append ins "out asum*k1"


# build the instrument and a dummy score


MakeIns 1 $ins
puts $sco "f0 10"
close $orc
close $sco


# compile
csCompile $orcfile $scofile -odac -d -m0


# set a wavetable
csTable 1 0 16384 10 1 .5 .25 .2 .17 .15 .12 .1


# send in a sequence of events and perform it
for {set i 0} { $i < 60 } { incr i } {
csNote 1 [expr $i * 0.1] .5 \
[expr ($i * 10) + 500] [expr 100 + $i * 10]
}
csPerform


# it is possible to run it interactively as
# well
csNote 1 0 10 1000 200
csPlay
```

De telles facilités comme celles fournies par emacs permettent d'émuler un environnement assez proche de ce qu'on trouve dans les soi-disant "systèmes de synthèse modernes", tels que SuperCollider (SC). En fait, on peut exécuter Csound dans une configuration client-serveur, ce qui est une des fonctionnalités de SC3. Csound a l'avantage majeur de fournir trois ou quatre fois plus de générateurs unitaires que ce qu'on trouve dans ce langage (de même qu'il fournit une approche du traitement du signal à un plus bas niveau, en fait ce ne sont là que quelques-uns des avantages de Csound).

TclCsound comme encapsuleur de langage

On peut utiliser TclCsound à un niveau légèrement plus bas, car beaucoup des fonctions de l'API C ont été encapsulées dans des commandes Tcl. Par exemple, il est possible de créer un frontal "classique" pour csound en ligne de commande complètement écrit en Tcl. Le script suivant le démontre :

```
#!/usr/local/bin/cstclsh


set result 1
csCompileList $argv
while { $result != 0 } {
set result csPerformKsmps
}
```

Référence des Commandes de TclCsound

Commandes de contrôle de l'exécution :

csCompile [ligne de commande csound] : compile un orc/sco/csd + des options

csCompileList arglist : compile un orc/sco/csd + des options, donnés comme une liste Tcl 'arglist'

csPerform : joue la partition, retournant à la fin

csPerformKsmps : exécute un bloc de ksmps échantillons audio, puis retourne

csPerformBuffer : exécute un bloc d'échantillons audio de la taille d'un tampon, puis retourne

csPlay : démarre une exécution asynchrone dans un thread séparé, retournant immédiatement

csPause : suspend la reproduction

csStop : arrête l'exécution et réinitialise csound

csRewind : repositionne la partition au début

csOffset secs : décale le point de reproduction dans la partition de 'secs' secondes

csGetoffset : retourne le point de décalage dans la partition en secondes

csGetScoreTime : retourne le temps de la partition en secondes

Commandes d'évènements :

csNote [p-champs] : envoie un évènement dans une instruction i

csTable [p-champs] : envoie un évènement dans une instruction f

csEvent opcode [p-champs] : envoie un évènement de partition défini par 'opcode' plus les p-champs

csNoteList arglist : envoie un évènement dans une instruction i avec les p-champs dans une liste Tcl 'arglist'

csTableList arglist : envoie un évènement dans une instruction f avec les p-champs dans une liste Tcl 'arglist'

csEventList arglist : envoie un évènement de partition défini par 'opcode' avec les p-champs dans une liste Tcl 'arglist'

Commandes de canal de contrôle et de chaîne, invalue, outvalue, pvsin, pvsout :

csInChannel nom : enregistre un canal csound invalue

csOutChannel nom : enregistre un canal csound outvalue et crée la variable tcl globale 'nom'

csInValue canal valeur : fixe une valeur sur un canal csound invalue

csOutValue canal : retourne la valeur d'un canal csound outvalue

csPvsIn number [size olaps wsize wtype] : enregistre un canal du bus d'entrée pvs, initialisant optionnellement les valeurs de fsig à une taille de tfr de 'size' (par défaut : 1024), une taille de chevauchement de 'olaps' (par défaut : size/4), une taille de fenêtre de 'wsize' (par défaut : size) et le type de fenêtre à 'wtype' (par défaut : 1, fenêtre de Hanning, voir la page de manuel pour pvsanal). Fonctionne avec l'opcode pvsin (seulement le format PVS_AMP_FREQ).

csPvsOut number [size olaps wsize wtype] : enregistre un canal du bus de sortie pvs. Fonctionne avec

l'opcode pvsout (seulement le format PVS_AMP_FREQ).

csPvsInSet channel bin amp freq : fixe l'amplitude et la fréquence d'un bin du canal d'entrée pvs 'channel'.

csPvsOutGet channel bin [isFreq] : retourne l'amplitude ou la fréquence d'un bin du canal de sortie pvs 'channel'. L'argument optionnel 'isFreq' (par défaut : 0) contrôle si la valeur retournée est l'amplitude du bin (0) ou sa fréquence (1).

csSetControlChannel channel value : fixe la valeur du canal de contrôle 'channel', le créant s'il n'existe pas.

csGetControlChannel channel : retourne la valeur du canal de contrôle 'channel', le créant s'il n'existe pas.

csSetStringChannel channel string : fixe la chaîne dans le canal 'channel', le créant s'il n'existe pas.

csGetStringChannel channel : retourne la chaîne qui est dans le canal 'channel', le créant s'il n'existe pas.

Commandes de message :

csMessageOutput var : ajoute tous les messages csound à la variable tcl 'var'.

Commandes de table :

csGetTableSize ftn : retourne la taille de la table de fonction ftn (-1 si elle n'existe pas).

csSetTable ftn index value : fixe la valeur de la position 'index' dans la table de fonction 'ftn' à 'value'.

csGetTable ftn index : retourne la valeur de la position 'index' dans la table de fonction 'ftn'.

Commandes de variable d'environnement :

csOpcodedir opcodedir : fixe le répertoire des opcode.

csSetenv envvar value : fixe la valeur d'une variable d'environnement (par exemple SFDIR, SADIR).

Construire Csound

Csound est devenu un projet complexe et peut impliquer plusieurs dépendances. A moins d'être un développeur de Csound ou d'avoir besoin d'écrire des plugins pour Csound, il vaut mieux utiliser une des distibutions pré-compilées de http://www.sourceforge.net/projects/csound. Cependant, la construction à partir des sources est sans doute la meilleure option sous GNU/Linux.

Cette section met l'accent sur le système principal de construction de Csound 5, qui utilise SCons [http://www.scons.org], un programme Python qui remplace *make* pour la configuration et la construction multi plates-formes.

Lorsque l'on construit Csound à partir des sources plutôt que d'utiliser un paquetage précompilé, il faut d'abord télécharger les sources d'une publication de Csound à partir de http://www.sourceforge.net/projects/csound. Les paquetages source ont une extension zip ou tar.gz.

Le code source de Csound le plus récent (potentiellement instable) est également disponible par Concurrent Versions System (CVS). Il est probable (si vous êtes sous Mac OS X ou Linux) que CVS est déjà installé sur votre machine. Si ce n'est pas le cas, il peut être téléchargé à partir de (http://www.cvshome.org). Il y a de nombreux frontaux graphiques pour cvs, mais il est facile de télécharger les sources au moyen de la version en ligne de commande.

La page d'accueil du CVS de Csound se trouve ici : http://sourceforge.net/cvs/?group_id=81968 On peut trouver de l'information sur la manière d'accéder à l'entrepôt CVS de Csound dans le document de SourceForge http://sourceforge.net/docs/E04/ Pour télécharger les sources de Csound avec CVS, exécutez les commandes suivantes (à partir d'un terminal ou d'un shell DOS) :

```
cvs -d:pserver:anonymous@csound.cvs.sourceforge.net:/cvsroot/csound login

cvs -z3 -d:pserver:anonymous@csound.cvs.sourceforge.net:/cvsroot/csound co -P csound5
```

Pour mettre à jour les sources de Csound5 que vous auriez déjà dans votre répertoire csound5, allez dans ce répertoire et tapez :

```
cvs -z3 update -d
```

Pour mettre à jour un seul fichier, allez dans le répertoire des sources et tapez :

```
cvs -z3 update filename
```

Conditions nécessaires pour construire Csound 5 sur toutes les plates-formes

- Installer libdsndfile version 1.0.13 ou ultérieure depuis www.mega-nerd.com/libsndfile [http://www.mega-nerd.com/libsndfile].
- Installer Python depuis www.python.org [http://www.python.org]. Dans la plupart des cas il vaut mieux installer la version stable la plus récente. Scons a besoin de Python pour fonctionner.
- Installer le système de construction SCons depuis www.scons.org [http://www.scons.org].

Ce sont les conditions minimales pour une construction, mais csound a beaucoup de composants optionnels qui améliorent ses fonctionnalités et qui ajoutent des opcodes pouvant avoir besoin de bibliothèques

supplémentaires.

Configurations optionnelles (TOUTES les plates-formes)

Dans la plupart des cas, il vaut mieux installer les versions stables les plus récentes des bibliothèques optionnelles.

- L'audio en temps réel peut utiliser la bibliothèque multi plates-formes PortAudio (version principale ou branche devel-19) depuis www.portaudio.com/usingcvs.html [http://www.portaudio.com/usingcvs.html]. A de noter que la version stable 18 ne fonctionnera pas. Csound peut aussi utiliser plusieurs APIS spécifiques aux plates-formes telles que ALSA, JACK, CoreAudio et la bibliothèque multimedia de Windows. Voir les notes de chaque plate-forme pour les détails.
- Le MIDI en temps réel peut utiliser la bibliothèque multi plates-formes PortMidi depuis www.cs.cmu.edu/~music/portmusic [http://www.cs.cmu.edu/~music/portmusic]
- Pour les contrôleurs graphiques, installer FLTK 1.1 ou 1.3 depuis www.fltk.org [http://www.fltk.org]. Il faut configurer et construire FLTK avec `--enable-shared --enable-threads`.
- Pour générer les interfaces Python et Java, installer le Software Interface and Wrapper Generator (SWIG) depuis http://www.swig.org.
- *CsoundAC* nécessite FLTK et les bibliothèques de template C++ boost pour les nombres aléatoires et l'algèbre linéaire, depuis http://www.boost.org. *CsoundAC* nécessite au moins la version 1.32.1.
- Les opcodes fluid nécessitent la bibliothèque Fluidsynth depuis http://savannah.nongnu.org/download/fluid.
- Les opcodes OSC nécessitent la dernière version de la bibliothèque liblo depuis http://plugin.org.uk/liblo. Sous Windows, liblo nécessite une version Windows de la bibliothèque de processus légers POSIX (pthreads) qui est disponible à http://sourceware.org/pthreads-win32 ; copier libpthreadGC2.a vers libpthread.a. On peut aussi avoir besoin de la dernière version d'autoconf de MinGW.
- Les opcodes STK nécessitent le code source de STK depuis http://ccrma.stanford.edu/software/stk, à copier dans `csound5/Opcodes/stk`.
- Les opcodes de Loris nécessitent le code source de Loris depuis http://sourceforge.net/projects/loris, à copier dans `csound5/Opcodes/Loris`.

Windows

On a besoin des éléments suivants pour la construction sous Windows (on peut trouver des instructions plus complètes pour la construction sous Windows dans le document csound-build.tex (csound-build.pdf)) :

- Installer un compilateur comme gcc ou Microsoft Visual Studio (le compilateur C++ d'Intel est également supporté). Si l'on utilise MinGW (gcc), installer l'ensemble de la distribution actuelle de MinGW au moyen de l'installeur automatisé de MinGW depuis www.mingw.org [http://www.mingw.org], par exemple dans `c:/mingw`. Ceci installera gcc, g++, GNU binutils, les runtime de MinGW et l'API win32. Installer ensuite la version actuelle de MSys.

  Sous Windows on peut utiliser Microsoft Visual C++ (sauf pour CsoundAC). L'Express Edition libre, depuis http://www.microsoft.com/express/vc/ fonctionne très bien. Il vous faudra une copie du fichier

d'en-tête de Windows `dirent.h`, par exemple depuis http://www.softagalleria.net/dirent.php. On peut aussi avoir besoin de la bibliothèque `bufferoverflowu.lib` de Microsoft à déposer dans le répertoire `lib` de Visual C++. Ouvrir ensuite un shell pour compiler Csound (habituellement appelé Visual Studio Command Prompt command, depuis le menu du programme Visual C++).

Les configurations optionnelles pour Windows comprennent :

- La bibliothèque multimedia de Windows pour l'audio en temps réel et le MIDI. Ce module sera construit automatiquement si les en-têtes sont trouvés.
- Les en-têtes VST de Steinberg pour les opcodes de l'hôte VST.

Linux

Les configurations optionnelles pour Linux comprennent :

- ALSA (www.alsa-project.org [http://www.alsa-project.org]) et JACK (www.jackaudio.org/ [http://www.jackaudio.org/]) en plus de PortAudio, pour l'audio en temps-réel. Les distributions de linux fournissent habituellement les paquetages de développement pour ces systèmes dans leurs entrepôts.
- Les en-têtes LADSPA et DSSI pour les opcodes de l'hôte DSSI.

Mac OS X

Les configurations optionnelles pour Mac OS X comprennent :

- CoreAudio (système audio natif d'OSX) et JACK, en plus de PortAudio, pour l'audio en temps-réel.
- Les en-têtes LADSPA et DSSI pour les opcodes de l'hôte DSSI.

Construire Csound 5 avec SCons

Lorsque vous avez tous les paquetages requis et leur sources (ou les paquetages -dev) pour besoins particuliers sur votre plate-forme, exécutez "scons -h" pour découvrir les options de configuration.

La construction est considérablement facilitée si les bibliothèques et les en-têtes téléchargés sont installés dans leurs répertoires par défaut. Si l'on veut modifier la construction par défaut, en particulier pour prendre en compte les options non-standard des dépendances de tierces parties pour lesquelles il faut trouver les en-têtes et les bibliothèques :

- Sous Windows, si l'on utilise Microsoft Visual C++, modifier custom-msvc.py
- Sous Windows, si l'on utilise MinGW/MSys, modifier custom-mingw.py
- Sous Linux et Mac OSX éditer custom.py

Si vous modifiez ce fichier, marquez-le en lecture seule (c.à.d. protégez-le) afin que le CVS ne l'écrase pas lors d'une prochaine mise à jour des fichiers sources. Evitez de modifier le fichier SConstruct.

Exécutez scons avec les variables pour les options que vous désirez. Par exemple :

```
scons buildOSC=1 buildCsound5GUI=1 buildPythonOpcodes=1 useOSC=1 buildLoris=0
```

Note

Il est important de positionner la variable d'environnement `OPCODEDIR` sur le répertoire dans lequel les bibliothèques de plugin se trouvent ; dans le cas d'une construction en double précision, il faut plutôt positionner `OPCODEDIR64`. Les installeurs s'occupent habituellement de ceci, mais Csound doit pouvoir trouver ses bilbliothèques de plugin lorsqu'on le construit à partir des sources.

Options de construction

Tableau 4. Options de construction de SCons

Variable d'ajustement	Effet si positionnée à 1
buildCsoundVST	Construire CsoundVST. Nécessite CsoundAC, FLTK, boost, Python, SWIG.
buildCsoundAC	Construire CsoundAC. Nécessite FLTK, boost, Python, SWIG.
buildCsound5GUI	Construire le frontal graphique FLTK. Nécessite FLTK 1.1.7 ou ultérieur.
buildCSEditor	Construire l'éditeur de texte avec coloration syntaxique de Csound. Nécessite les en-têtes et les bibliothèques de FLTK.
buildDSSI	Construire les opcodes de l'hôte DSSI/LADSPA.
buildImageOpcodes	Construire les opcodes d'image. 1 par défaut. Mettre à 0 pour désactiver.
buildInterfaces	Construire la bibliothèque d'interface pour Python, JAVA, Lua, C++ et d'autres langages.
buildJavaWrapper	Construire la sur-couche Java pour la bibliothèque d'interface.
buildLoris	Construire les opcodes et l'extension Python de Loris.
buildNewParser	Activer le nouveau parser. Nécessite Flex/Bison.
buildOSXGUI	Construire le frontal graphique de base. Seulement sous OSX.
buildPDClass	Construire la classe PD csoundapi~. Nécessite m_pd.h à l'endroit standard.
buildPythonOpcodes	Construire les opcodes Python
buildRelease	Construire en mode release. Positionne noDebug.
buildSDFT	Construire le code SDFT. 1 par défaut. Mettre à 0 pour désactiver.
buildTclcsound	Construire le frontal Tclcsound (cstclsh, cswish et le module dynamique tclcsound). Nécessite les en-têtes et les bibliothèques Tcl/Tk.
buildUtilities	Construire des exécutables autonomes pour les uti-

Variable d'ajustement	Effet si positionnée à 1
	litaires que l'on peut aussi appeler avec -U.
buildVirtual	Construire le clavier virtuel MIDI. Nécessite les en-têtes et les bibliothèques de FLTK 1.1.7 ou ultérieur.
buildvst4cs	Construire les plugins vst4cs. Nécessite les en-têtes VST de Steinberg.
buildWinsound	Construire le frontal Winsound. Nécessite les en-têtes et les bibliothèques FLTK.
dynamicCsoundLibrary	Construire une bibliothèque Csound dynamique au lieu de libcsound.a.
gcc3opt	Autoriser les optimisations de gcc 3.3.x ou ultérieur pour l'architecture CPU spécifiée (par exemple pentium3) ; positionne noDebug.
gcc4opt	Autoriser les optimisations de gcc 4.0 ou ultérieur pour l'architecture CPU spécifiée (par exemple pentium3) ; positionne noDebug.
generateTags	Générer des TAGS.
generatePdf	Générer la documentation PDF.
install	Autoriser les cibles d'installation.
Lib64	Construire pour lib64 plutôt que pour lib.
noDebug	Construire sans information de débogage.
noFLTKThreads	Ne pas utiliser de thread séparé pour les contrôles graphiques de FLTK.
useAltivec	Sous OSX, utiliser les options d'optimisation du gcc AltiVec.
useALSA	ALSA pour les entrées et les sorties audio en temps réel et MIDI.
useCoreAudio	Utiliser CoreAudio pour les entrées et les sorties audio en temps réel.
useDouble	Utiliser des nombres réels en double précision pour les échantillons audio.
useFLTK	Utiliser FLTK pour les graphiques et les opcodes de contrôle graphique.
useGettext	Utiliser le schéma de localisation de GNU
useGprof	Construire avec des informations de profilage (-pg).
usePortAudio	utiliser PortAudio pour les entrées et les sorties audio en temps réel.
usePortMIDI	Construire le plugin PortMidi pour les entrées et les sorties MIDI en temps réel.
useJack	A utiliser si vous avez compilé PortAudio pour utiliser Jack ; construit également le plugin Jack.
useLrint	Utiliser lrint() and lrintf() pour la conversion des nombres réels en entiers.
useOSC	Pour le support d'OSC.
useUDP	Pour le support d'UDP. 1 par défaut. Mettre à 0 pour désactiver.

Variable d'ajustement	Effet si positionnée à 1
withICL	Construire avec le compilateur C++ d'Intel (nécessite également Microsoft Visual C++). Fixer à 0 pour MinGW. Seulement sous Windows.
withMSVC	Construire avec Microsoft Visual C++, ou fixer à 0 pour construire avec MinGW. Seulement sous Windows.
Word64	Construire pour des machines 64 bit.
pythonVersion	Fixer à la version de Python que l'on veut utiliser.

Liens Csound

La "page d'accueil" de Csound est maintenue par Richard Boulanger à http://www.csounds.com.

Le code source de Csound est maintenu par John ffitch et d'autres à http://www.sourceforge.net/projects/csound. Les versions les plus récentes et les paquetages précompilés pour la plupart des plates-formes peuvent être téléchargés ici [http://sourceforge.net/project/showfiles.php?group_id=81968].

Il existe une liste de diffusion Csound pour discuter de Csound. Elle est animée par John ffitch de Bath University, UK. Pour vous inscrire sur cette liste de diffusion envoyez un message vide à : csound-subscribe@lists.bath.ac.uk [mailto:csound-subscribe@lists.bath.ac.uk]. Vous pouvez aussi souscrire à la version condensée (1 message par jour) en envoyant un message vide à : csound-digest-subscribe@lists.bath.ac.uk [mailto:csound-digest-subscribe@lists.bath.ac.uk]. Les messages envoyés à csound@lists.bath.ac.uk [mailto:csound@lists.bath.ac.uk] sont distribués à tous les membres de la liste. On peut parcourir les archives de la liste de diffusion de Csound ici [http://agentcities.cs.bath.ac.uk/%7ebwillkie/list_arch.php].

De même, la liste de diffusion `Csound-devel` existe pour discuter du développement de Csound. Pour plus d'information sur cette liste, aller à http://lists.sourceforge.net/lists/listinfo/csound-devel. Les messages envoyés à csound-devel@lists.sourceforge.net [mailto:csound-devel@lists.sourceforge.net] vont à tous les membres de la liste.

Partie II. Vue d'Ensemble des Opcodes

Table des matières

Générateurs de Signal ... 93
Synthèse/Resynthèse Additive ... 93
Oscillateurs Elémentaires ... 93
Oscillateurs à Spectre Dynamique ... 93
Synthèse FM ... 94
Synthèse Granulaire ... 94
Synthèse Hyper Vectorielle ... 95
Générateurs Linéaires et Exponentiels ... 95
Générateurs d'Enveloppe ... 96
Modèles et Emulations ... 96
Phaseurs ... 97
Générateurs de Nombres Aléatoires (de Bruit) ... 98
Reproduction de Sons Echantillonnés ... 99
Soundfonts ... 99
Synthèse par Balayage ... 100
Accès aux Tables ... 102
Synthèse par Terrain d'Ondes ... 103
Modèles Physiques par Guide d'Onde ... 103
Entrée et Sortie de Signal ... 104
Entrées et Sorties Fichier ... 104
Entrée de Signal ... 104
Sortie de Signal ... 104
Bus Logiciel ... 105
Impression et Affichage ... 105
Requêtes sur les Fichiers Sons ... 105
Modificateurs de Signal ... 107
Modificateurs d'Amplitude et Traitement des Dynamiques ... 107
Convolution et Morphing ... 107
Retard ... 107
Panning et Spatialisation ... 108
Réverbération ... 110
Opérateurs du Niveau Echantillon ... 110
Limiteurs de Signal ... 111
Effets Spéciaux ... 111
Filtres Standard ... 111
Filtres Spécialisés ... 113
Guides d'Onde ... 113
Distorsion Non-Linéaire et Distorsion de Phase ... 113
Contrôle d'Instrument ... 115
Contrôle d'Horloge ... 115
Valeurs Conditionnelles ... 115
Instructions de Contrôle de Durée ... 115
Contrôleurs Graphiques FLTK et GUI ... 115
Conteneurs FLTK ... 118
Valuateurs FLTK ... 118
Autres Contrôleurs Graphiques FLTK ... 119
Modifier l'Apparence des Contrôleurs Graphiques FLTK ... 120
Opcodes Généraux relatifs aux Contrôleurs Graphiques FLTK ... 120
Appel d'Instrument ... 121
Contrôle Séquentiel d'un Programme ... 121
Controle de l'Exécution en Temps Réel ... 122
Initialisation et Réinitialisation ... 122
Détection et Contrôle ... 123

Piles 124
Contrôle de sous-instrument 125
Lecture du Temps 125
Contrôle des Tables de Fonction 126
Requêtes sur une Table 126
Opérations de Lecture/Ecriture de Table 126
Lecture de Table avec Sélection Dynamique 127
Opérations Mathématiques 128
Conversion d'Amplitude 128
Opérations Arithmétiques et Logiques 128
Comparateurs et Accumulateurs 128
Fonctions Mathématiques 129
Opcodes Equivalents à des Fonctions 129
Fonctions aléatoires 130
Fonctions Trigonométriques 130
Linear Algebra Opcodes 131
Conversion des Hauteurs 141
Fonctions 141
Opcodes de Hauteurs 141
Support MIDI en Temps-Réel 142
Clavier Virtuel MIDI 143
Entrée MIDI 146
Sortie de Message MIDI 146
Entrée et Sortie Génériques 147
Convertisseurs 147
Extension d'Evènements 147
Sortie de Note-on/Note-off 147
Opcodes pour l'Interopérabilité MIDI/Partition 148
Messages System Realtime 149
Banques de Réglettes 149
Traitement Spectral 151
Resynthèse par Tranformée de Fourier à Court-Terme (STFT) 151
Resynthèse par Codage Prédictif Linéaire (LPC) 152
Traitement Spectral Non-standard 152
Outils pour le Traitement Spectral en Temps Réel (opcodes pvs) 152
Traitement Spectral avec ATS 154
Opcodes Loris 154
Chaînes de Caractères 159
Opcodes de Manipulation de Chaîne 160
Opcodes de Conversion de Chaîne 160
Opcodes Vectoriels 162
Opérateurs de Tableaux de Vecteurs 162
Opérations Entre un Signal Vectoriel et un Signal Scalaire 162
Opérations Entre deux Signaux Vectoriels 163
Générateurs Vectoriels d'Enveloppe 163
Limitation et Enroulement des Signaux Vectoriels de Contrôle 164
Chemins de Retard Vectoriel au Taux de Contrôle 164
Générateurs de Signal Aléatoire Vectoriel 164
Système de Patch Zak 166
Accueil de Plugin 167
DSSI et LADSPA pour Csound 167
VST pour Csound 167
OSC et Réseau 169
OSC 169
Réseau 169
Opcodes pour le Traitement à Distance 169
Opcodes Mixer 170
Opcodes Python 171

Introduction 171
Syntaxe de l'Orchestre 171
Opcodes pour le traitement d'image 173
Opcodes divers 174

Générateurs de Signal

Synthèse/Resynthèse Additive

Les opcodes pour la synthèse et la resynthèse additives sont :

- *adsyn*
- *adsynt*
- *adsynt2*
- *hsboscil*

Voir la section *Traitement Spectral* pour plus d'information et des opcodes de synthèse/resynthèse additive supplémentaires.

Oscillateurs Elémentaires

Les opcodes des oscillateurs élémentaires sont : (noter que les opcodes qui se terminent par un 'i' implémentent l'interpolation linéaire et que ceux qui se terminent par un '3' implémentent l'interpolation cubique)

- Banques d'Oscillateurs : *oscbnk*
- Oscillateurs simples à table : *oscil*, *oscil3* et *oscili*.
- Oscillateur sinusoïdal simple, rapide : *oscils*
- Oscillateurs de précision : *poscil* et *poscil3*.
- Oscillateurs plus flexibles : *oscilikt*, *osciliktp*, *oscilikts* et *osciln* (aussi appelé *oscilx*).

On peut aussi construire des oscillateurs à partir des opcodes de lecture de table. Voir la section *Opérations de Lecture/Ecriture de Table*.

LFOs

- *lfo*
- *vibr*
- *vibrato*

Voir la section *Accès aux Tables* pour d'autres opcodes de lecture de table que l'on peut utiliser comme oscillateurs. Voir aussi la section *Oscillateurs à Spectre Dynamique*.

Oscillateurs à Spectre Dynamique

Les opcodes qui génère des spectres dynamiques sont :

- Spectres harmoniques : *buzz* et *gbuzz*
- Générateur d'impulsions : *mpulse*
- Oscillateurs à bande limitée (d'après des modèles analogiques) : *vco* et *vco2*

On peut utiliser les opcodes suivants pour générer des formes d'onde à bande limitée pour une utilisation avec *vco2* et d'autres oscillateurs :

- *vco2init*
- *vco2ft*
- *vco2ift*

Synthèse FM

Les opcodes de synthèse FM sont :

- *foscil*
- *foscili*

Modèles d'instrument FM

- *fmb3*
- *fmbell*
- *fmmetal*
- *fmpercfl*
- *fmrhode*
- *fmvoice*
- *fmwurlie*

Synthèse Granulaire

Les opcodes de synthèse granulaire sont :

- *diskgrain*
- *fof*

- *fof2*
- *fog*
- *grain*
- *grain2*
- *grain3*
- *granule*
- *partikkel*
- *partikkelsync*
- *sndwarp*
- *sndwarpst*
- *syncgrain*
- *syncloop*
- *vosim*

Synthèse Hyper Vectorielle

- *vphaseseg*

- *hvs1*
- *hvs2*
- *hvs3*

Générateurs Linéaires et Exponentiels

Les opcodes qui génèrent des courbes ou des segments linéaires ou exponentiels sont :

- *expon*
- *expcurve*
- *expseg*
- *expsega*
- *expsegr*

- *gainslider*
- *jspline*
- *line*
- *linseg*
- *linsegr*
- *logcurve*
- *loopseg*
- *loopsegp*
- *lpshold*
- *lpsholdp*
- *rspline*
- *scale*
- *transeg*

Générateurs d'Enveloppe

Les générateurs d'enveloppe suivants sont disponibles :

- *adsr*
- *madsr*
- *mxadsr*
- *xadsr*
- *linen*
- *linenr*
- *envlpx*
- *envlpxr*

Consulter la section des *Générateurs Linéaires et Exponentiels* pour d'autres méthodes de création d'enveloppes.

Modèles et Emulations

Les opcodes suivants réalisent la modélisation ou l'émulation des sons d'autres instruments (certains basés sur la boîte à outils STK par Perry Cook) :

- *bamboo*

- *barmodel*
- *cabasa*
- *crunch*
- *dripwater*
- *gogobel*
- *guiro*
- *mandol*
- *marimba*
- *moog*
- *sandpaper*
- *sekere*
- *shaker*
- *sleighbells*
- *stix*
- *tambourine*
- *vibes*
- *voice*

Autres modèles et émulations

- *lorenz*
- *planet*
- *prepiano*
- Générateur de Nombres Fractals (ensemble de Mandelbrot) : *mandel*
- *chuap*

Phaseurs

Les opcodes qui génèrent une valeur de phase mobile :

- *phasor*
- *phasorbnk*
- *syncphasor*

Ces opcodes sont utiles en combinaison avec les opcodes d'*Accès aux Tables*.

Générateurs de Nombres Aléatoires (de Bruit)

Les opcodes qui génèrent des nombres aléatoires sont :

- *betarnd*
- *bexprnd*
- *cauchy*
- *cuserrnd*
- *duserrnd*
- *exprand*
- *gauss*
- *linrand*
- *noise*
- *pcauchy*
- *pinkish*
- *poisson*
- *rand*
- *randh*
- *randi*
- *rnd31*
- *random*
- *randomh*
- *randomi*
- *trirand*
- *unirand*
- *urd*
- *weibull*
- *jitter*
- *jitter2*
- *trandom*

Voir *seed* qui fixe la valeur de la racine globale pour tous les générateurs de bruit de classe x, ainsi que d'autres opcodes qui utilisent un appel de fonction aléatoire comme *grain*. *rand*, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

Voir aussi les fonctions qui génèrent des nombres aléatoires dans la section *Fonctions Aléatoires*.

Reproduction de Sons Echantillonnés

Les opcodes qui implémentent la reproduction de sons échantillonnés (samples) et les boucles sont :

- *bbcutm*
- *bbcuts*
- *flooper*
- *flooper2*
- *loscil*
- *loscil3*
- *loscilx*
- *lphasor*
- *lposcil*
- *lposcil3*
- *lposcila*
- *lposcilsa*
- *lposcilsa2*
- *sndloop*
- *waveset*

Voir aussi la section *Entrée de Signal* pour d'autres types d'entrées sonores.

Soundfonts

Opcodes Fluid

La famille des opcodes fluid encapsule le lecteur SoundFont 2 de Peter Hannape, FluidSynth : *fluidEngine* pour instancier un moteur FluidSynth, *fluidSetInterpMethod* pour fixer la méthode d'interpolation d'un canal dans un moteur FluidSynth, *fluidLoad* pour charger des SoundFonts, *fluidProgramSelect* pour assigner des presets d'un SoundFont à un canal MIDI d'un moteur FluidSynth, *fluidNote* pour jouer une note sur un canal MIDI d'un moteur FluidSynth, *fluidCCi* pour envoyer un message de contrôleur au temps-i sur un canal MIDI d'un moteur FluidSynth, *fluidCCk* pour envoyer un message de contrôleur au taux k sur un canal MIDI d'un moteur FluidSynth. *fluidControl* pour jouer et contrôler les Soundfonts chargés (en utilisant des messages MIDI 'bruts'), *fluidOut* pour recevoir de l'audio depuis un seul moteur FluidSynth, et *fluidAllOut* pour recevoir de l'audio depuis tous les moteurs FluidSynth.

- *fluidAllOut*
- *fluidCCi*
- *fluidCCk*
- *fluidControl*
- *fluidEngine*
- *fluidLoad*
- *fluidNote*
- *fluidOut*
- *fluidProgramSelect*
- *fluidSetInterpMethod*

Anciens opcodes Soundfont

Ces opcodes peuvent aussi employer des soundfonts pour générer du son. L'utilisation des opcodes fluid (ci-dessus) est recommandées plutôt que celle de ces opcodes.

- *sfilist*
- *sfinstr*
- *sfinstr3*
- *sfinstr3m*
- *sfinstrm*
- *sfload*
- *sfpassign*
- *sfplay*
- *sfplay3*
- *sfplay3m*
- *sfplaym*
- *sflooper*
- *sfplist*
- *sfpreset*

Synthèse par Balayage

La synthèse par balayage (scanned synthesis) est une variante des modèles physiques, dans laquelle un

réseau de masses connectées par des ressorts est utilisé pour générer une forme d'onde dynamique. L'opcode *scanu* définit le réseau de masses/ressorts et le met en mouvement. L'opcode *scans* suit un chemin prédéfini (une trajectoire) à travers le réseau et donne en sortie la forme d'onde détectée. Plusieurs instances de *scans* peuvent suivre différents chemins à travers le même réseau.

Ce sont des algorithmes de modélisation mécanique hautement efficaces à la fois pour la synthèse et l'animation sonore via un traitement algorithmique. Il vaut mieux les utiliser en temps réel. Ainsi, la sortie est utile soit directement pour l'audio, soit comme valeurs de contrôleur pour d'autres paramètres.

L'implémentation dans Csound ajoute le support pour un chemin de balayage ou matrice. Essentiellement, ceci offre la possibilité de reconnecter les masses dans d'autres configurations, provoquant une propagation du signal assez différente. Elles ne doivent pas nécessairement être connectées à leurs voisines directes. La matrice a essentiellement l'effet de « modeler » la surface en une forme radicalement différente.

Pour produire les matrices, le format du tableau est direct. Par exemple, pour 4 masses nous avons la grille suivante qui décrit les connexions possibles :

	1	2	3	4
1				
2				
3				
4				

Chaque fois que deux masses sont connectées, le point qu'elles définissent vaut 1. Si deux masses ne sont pas connectées, le point qu'elles définissent vaut alors 0. Par exemple, une corde unidirectionnelle a les connexions suivantes : (1,2), (2,3), (3,4). Si elle est bidirectionnelle, elle a aussi (2,1), (3,2), (4,3). Pour la corde unidirectionnelle, la matrice est :

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

Le format de tableau ci-dessus pour la matrice de connexion n'est donné que par commodité conceptuelle. Les valeurs actuellement montrées dans le tableau sont obtenues par *scans* depuis un fichier ASCII en utilisant *GEN23*. Le fichier ASCII lui-même est créé à partir du tableau modèle ligne par ligne. Ainsi, le fichier ASCII pour le tableau de l'exemple montré ci-dessus devient :

0100001000010000

Cet exemple de matrice est très simple et très petit. En pratique, la plupart des instruments de synthèse par balayage utiliseront bien plus que quatre masses, et donc leurs matrices seront bien plus grandes et plus complexes. Voir l'exemple dans la documentation de *scans*.

Prière de noter que les tables d'onde dynamiques générées sont très instables. Certaines valeurs de masses, de centrage, et d'amortissement peuvent provoquer une « explosion » du système et l'apparition des sons les plus intéressants sur vos haut-parleurs.

Le supplément de ce manuel contient un tutoriel sur la synthèse par balayage. Le tutoriel, des exemples, et d'autres informations sur la synthèse par balayage sont disponibles sur la page Scanned Synthesis à

csounds.com [http://www.csounds.com/scanned/].

La synthèse par balayage a été développée par Bill Verplank, Max Mathews et Rob Shaw à Interval Research entre 1998 et 2000.

Les opcodes qui implémentent la synthèse par balayage sont :

- *scanhammer*
- *scans*
- *scantable*
- *scanu*
- *xscanmap*
- *xscans*
- *xscansmap*
- *xscanu*

Accès aux Tables

Les opcodes qui permettent l'accès aux tables sont :

- *oscil1*
- *oscil1i*
- *osciln*
- *oscilx*
- *table*
- *table3*
- *tablei*

Les opcodes se terminant par 'i' implémentent l'interpolation linéaire et les opcodes se terminant par '3' implémentent l'interpolation cubique.

Les opcodes suivants implémentent la lecture/écriture rapide dans une table sans en tester les limites :

- *tab*
- *tab_i*
- *tabw*
- *tabw_i*

Voir les sections *Requêtes de Table*, *Opérations de Lecture/Ecriture de Table* et *Lecture de Table avec*

Sélection Dynamique pour d'autres opérations de table.

Note

Bien que des tables avec une taille qui n'est pas une puissance de deux puissent être créées en utilisant une taille négative (voir *instruction de partition f*), certains opcodes ne les accepteront pas.

Synthèse par Terrain d'Ondes

L'opcode qui utilise la synthèse par terrain d'ondes est : *wterrain*.

Modèles Physiques par Guide d'Onde

Les opcodes qui implémentent les modèles physiques par guide d'onde sont :

- *pluck*
- *repluck*
- *wgbow*
- *wgbowedbar*
- *wgbrass*
- *wgclar*
- *wgflute*
- *wgpluck*
- *wgpluck2*
- *wguide1*
- *wguide2*

Entrée et Sortie de Signal

Entrées et Sorties Fichier

Les opcodes pour les entrées et sorties fichier sont :

- Ouverture/fermeture de fichier : *fiopen* et *ficlose*.
- Sortie fichier : *dumpk*, *dumpk2*, *dumpk3*, *dumpk4*, *fout*, *fouti*, *foutir* et *foutk*
- Entrée fichier : *readk*, *readk2*, *readk3*, *readk4*, *fin*, *fini* et *fink*
- Utilitaires à utiliser avec les opcodes fout : *clear*, *vincr*
- Impression dans un fichier : *fprints* et *fprintks*

Entrée de Signal

Les opcodes qui reçoivent des signaux audio sont :

- Entrée synchrone : *in*, *in32*, *inch*, *inh*, *ino*, *inq*, *inrg*, *ins* et *inx*
- Flux de fichier : *diskin*, *diskin2* et *soundin*
- Canal d'entrée défini par l'utilisateur : *invalue*
- Flux d'entrée : *soundin*
- Entrée directe dans zak : *inz*

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

Sortie de Signal

Les opcodes qui écrivent des signaux audio sont :

- Sortie synchrone : *out*, *out32*, *outc*, *outch*, *outh*, *outo*, *outrg*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2* et *outx*
- Flux de sortie : *soundout* et *soundouts*
- Canal de sortie défini par l'utilisateur : *outvalue*
- Sortie directe depuis zak : *outz*

L'opcode *monitor* peut être utilisé pour surveiller la sortie complète de csound (trame de sortie spout).

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

Bus Logiciel

Csound implémente un bus logiciel pour le routage interne ou le routage vers des logiciels externes en appelant l'API de Csound.

Les opcodes pour utiliser le bus logiciel sont :

- *chn_k*
- *chn_a*
- *chn_S*
- *chnclear*
- *chnexport*
- *chnmix*
- *chnparams*

Impression et Affichage

Les opcodes pour imprimer et afficher des valeurs sont :

- *dispfft*
- *display*
- *flashtxt*
- *print*
- *printf*
- *printf_i*
- *printk*
- *printk2*
- *printks*
- *prints*

Requêtes sur les Fichiers Sons

Les opcodes qui demandent de l'information sur les fichiers sont :

- *filelen*
- *filenchnls*

- *filepeak*
- *filesr*

Modificateurs de Signal

Modificateurs d'Amplitude et Traitement des Dynamiques

Les opcodes qui modifient l'amplitude sont :

- *balance*
- *compress*
- *clip*
- *dam*
- *gain*

L'opcode *0dbfs* facilite la manipulation d'amplitude en supprimant la nécessité d'utiliser des valeurs d'échantillon explicites.

Convolution et Morphing

Les opcodes qui font la convolution et le morphing de signaux sont :

- *convolve* aussi nommé *convle*
- *cross2*
- *dconv*
- *ftconv*
- *ftmorf*
- *pconvolve*

Retard

Retards fixes

- *delay*
- *delay1*
- *delayk*

Lignes à retard

- *delayr*
- *delayw*
- *deltap*
- *deltap3*
- *deltapi*
- *deltapn*
- *deltapx*
- *deltapxw*

Retards variables

- *vdelay*
- *vdelay3*
- *vdelayx*
- *vdelayxs*
- *vdelayxq*
- *vdelayxw*
- *vdelayxwq*
- *vdelayxws*

Retards multiples

- *multitap*

Panning et Spatialisation

Spatialisation d'Amplitude

- *locsend*
- *locsig*
- *pan*

- *pan2*
- *space*
- *spdist*
- *spsend*

Spatialisation 3D avec simulation d'acoustique des salles

- *spat3d*
- *spat3di*
- *spat3dt*

Panning d'Amplitude à Base Vectorielle

- *vbap16*
- *vbap16move*
- *vbap4*
- *vbap4move*
- *vbap8*
- *vbap8move*
- *vbaplsinit*
- *vbapz*
- *vbapzmove*

Spatialisation Binaurale

- *hrtfer*
- *hrtfmove*
- *hrtfmove2*
- *hrtfstat*

Ambisonics

- *bformdec*
- *bformenc*

Réverbération

Les opcodes qu'on peut utiliser pour la réverbération sont :

- *alpass*
- *babo*
- *comb*
- *freeverb*
- *nestedap*
- *nreverb* (aussi appelé *reverb2*)
- *reverb*
- *reverbsc*
- *valpass*
- *vcomb*

Opérateurs du Niveau Echantillon

Les opérateurs que l'on peut utiliser pour modifier les signaux sont :

- *a(k)*
- *denorm*
- *diff*
- *downsamp*
- *fold*
- *i(k)*
- *integ*
- *interp*
- *i(k)*
- *ntrpol*
- *samphold*

- *upsamp*
- *vaget*
- *vaset*

Limiteurs de Signal

Les opcodes que l'on peut utiliser pour limiter des signaux sont :

- *limit*
- *mirror*
- *wrap*

Effets Spéciaux

Les opcodes qui génèrent des effets spéciaux sont :

- *distort*
- *distort1*
- *flanger*
- *harmon*
- *phaser1*
- *phaser2*

Filtres Standard

Filtres passe-bas à résonance

- *areson*
- *lowpass2*
- *lowres*
- *lowresx*
- *lpf18*
- *moogvcf*
- *moogladder*

- *reson*
- *resonr*
- *resonx*
- *resony*
- *resonz*
- *rezzy*
- *statevar*
- *svfilter*
- *tbvcf*
- *vlowres*
- *bqrez*

Filtres standard

- Filtres passe-haut : *atone*, *atonex*
- Filtres passe-bas : *tone*, *tonex*
- Filtres biquadratiques : *biquad* et *biquada*.
- Filtres de Butterworth : *butterbp*, *butterbr*, *butterhp*, *butterlp* (qui sont aussi appelés *butbp*, *butbr*, *buthp*, *butlp*)
- Filtres généraux : *clfilt*

Filtres de signal de contrôle

- *aresonk*
- *atonek*
- *lineto*
- *port*
- *portk*
- *resonk*
- *resonxk*
- *tlineto*
- *tonek*

Filtres Spécialisés

Filtres passe-haut

- *dcblock*
- *dcblock2*

Egaliseurs paramétriques

- *pareq*
- *rbjeq*
- *eqfil*

Autres filtres

- *nlfilt*
- *filter2*
- *fofilter*
- *hilbert*
- *zfilter2*

Guides d'Onde

Les opcodes qui utilisent des guides d'onde pour modifier un signal sont :

- *streson*
- *wguide1*
- *wguide2*

Distorsion Non-Linéaire et Distorsion de Phase

Ces opcodes peuvent exécuter de façon dynamique une distorsion non-linéaire ou une distorsion de phase. Il diffèrent des méthodes traditionnelles de distorsion non-linéaire basées sur une table, en calculant directement la fonction de transfert avec un ou plusieurs paramètres variables pour modifier l'importance ou les résultats de la distorsion. La plupart de ces opcodes peuvent être utilisés sur un signal audio (pour la distorsion non-linéaire) ou sur un phaseur (pour la distorsion de phase) mais ils ont tendance à fonctionner au mieux pour une de ces applications.

Ces opcodes sont adaptés à la distorsion non-linéaire :

- *chebyshevpoly*
- *clip*
- *distort*
- *distort1*
- *polynomial*
- *powershape*

Ces opcodes sont adaptés à la distorsion de phase :

- *pdclip*
- *pdhalf*
- *pdhalfy*

Contrôle d'Instrument

Contrôle d'Horloge

Les opcodes pour démarrer et arrêter les horloges internes sont :

- *clockoff*
- *clockon*

Ces horloges comptent le temps CPU. On dispose de 32 horloges indépendantes. On peut utiliser l'opcode *readclock* pour lire les valeurs courantes d'une horloge. Voir *Lecture du Temps* pour d'autres opcodes de chronométrage.

Valeurs Conditionnelles

Les opcodes pour les valeurs conditionnelles sont ==, >=, >, <, <= et *!=*.

Instructions de Contrôle de Durée

Les opcodes que l'on peut utiliser pour manipuler la durée d'une note sont :

- *ihold*
- *turnoff*
- *turnoff2*
- *turnon*

Pour d'autres contrôles d'instrument en temps réel voir *Controle de l'Exécution en Temps Réel* et *Appel d'Instrument*.

Contrôleurs Graphiques FLTK et GUI

Les contrôleurs graphiques permettent de dessiner une Interface Utilisateur Graphique (GUI) personnalisée pour contrôler un orchestre en temps réel. Ils sont dérivés de la bibliothèque libre FLTK (Fast Light ToolKit). Cette bibliothèque est une des plus rapides parmi les bibliothèques disponibles, supporte OpenGL et devrait être compatible avec différentes plates-formes (Windows, Linux, Unix et Mac OS). Le sous-ensemble de FLTK implémenté dans Csound fournit les types d'objets suivants :

Conteneurs

: Les *Conteneurs FLTK* sont des contrôleurs graphiques qui contiennent d'autres contrôleurs tels que des panneaux, des fenêtres, etc. Csound fournit les objets conteneurs suivants :

- Panneaux
- Zones déroulantes

- Paquets
- Onglets
- Groupes

Valuateurs

Les objets les plus utiles sont appelés *Valuateurs FLTK*. Ces objets permettent à l'utilisateur de modifier les valeurs des paramètres de synthèse en temps réel. Csound fournit les objets valuateurs suivants :

- Réglettes
- Boutons rotatifs
- Boutons roulants
- Champs texte
- Joysticks
- Compteurs

Autres contrôleurs graphiques

Il y a d'*autres contrôleurs graphiques FLTK* qui ne sont ni des valuateurs ni des conteneurs :

- Boutons
- Bancs de boutons
- Etiquettes
- Détection Clavier et Souris

Il y a aussi d'autres opcodes utiles pour modifier l'*apparence des contrôleurs graphiques* :

- Mettre à jour la valeur d'un contrôleur.
- Choisir les couleurs principale et de sélection d'un contrôleur.
- Choisir le type, la taille et la couleur de police des contrôleurs.
- Redimensionner un contrôleur.
- Cacher et Montrer un contrôleur.

Il y a aussi ces *opcodes généraux* qui permettent les actions suivantes :

- Lancer le processus léger (thread) des contrôleurs graphiques : *FLrun*
- Charger des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLgetsnap* et *FLloadsnap*.
- Sauvegarder des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLsavesnap* et *FLsetsnap*

- Fixer le groupe d'instantanés d'un valuateur déclaré : *FLsetSnapGroup*

Ci-dessous un exemple simple de code Csound pour créer une fenêtre. Noter que tous les opcodes sont de taux-init et ne doivent être appelés qu'une seule fois par session. La meilleure manière de les utiliser est de les placer dans la section d'en-tête de l'orchestre, avant tout instrument. Même s'il n'est pas interdit de les placer dans un instrument, cela peut conduire à des résultats imprévisibles si l'instrument est appelé plus d'une fois.

Chaque conteneur est fait d'un couple d'opcodes : le premier indique le début du bloc du conteneur et le deuxième indique la fin du bloc du conteneur. Certains blocs de conteneur peuvent être imbriqués mais il ne peuvent pas se chevaucher. Après avoir défini tous les conteneurs, il faut lancer un processus léger de contrôleurs graphiques en utilisant l'opcode spécial *FLrun* qui ne prend pas d'argument.

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio    Pas de messages
   -odac           -iadc            -d           ;;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*******************************
sr=48000
kr=480
ksmps=100
nchnls=1

;*** Il est recommandé de placer presque tout le code GUI dans la
;*** section d'en-tête de l'orchestre

        FLpanel         "Panel1",450,550 ;***** début du conteneur
; placer ici quelques contrôleurs graphiques
        FLpanelEnd      ;***** fin du conteneur

        FLrun           ;***** lance le thread FLTK, toujours requis !
instr 1
; placer ici du code de synthèse
endin
;*******************************
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>
```

Le code précédent crée simplement un panneau (une fenêtre vide car aucun contrôleur graphique n'est défini à l'intérieur du conteneur).

L'exemple suivant crée deux panneaux et insère une réglette dans chacun d'entre eux :

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio    Pas de messages
   -odac           -iadc            -d           ;;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*******************************
sr=48000
kr=480
ksmps=100
```

```
nchnls=1

         FLpanel         "Panel1",450,550,100,100 ;***** début de conteneur
gk1,iha  FLslider        "FLslider 1", 500, 1000, 0 ,1, -1, 300,15, 20,50
         FLpanelEnd      ;***** fin de conteneur

         FLpanel         "Panel2",450,550,100,100 ;***** début de conteneur
gk2,ihb  FLslider        "FLslider 2", 100, 200, 0 ,1, -1, 300,15, 20,50
         FLpanelEnd      ;***** fin de conteneur

         FLrun           ;***** lance le thread FLTK, toujours requis !

instr 1
; les variables gk1 et gk2 qui contiennent les valeurs de sortie des valuateurs
; définis précédemment, peuvent être utilisées à l'intérieur des instruments
printk2 gk1
printk2 gk2   ; imprime les valeurs des valuateurs chaque fois qu'elles changent
endin
;********************************
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>
```

Tous les opcodes de contrôleur graphique sont des opcodes de taux-init, même si les valuateurs donnent en sortie des variables de taux-k. Ceci est dû au fait qu'un processus léger indépendant est exécuté sur la base d'un mécanisme de fonctions de rappel. Cela permet de consommer très peu de ressources système car on évite la scrutation. (A la différence des autres opcodes de contrôleurs basés sur le MIDI). On peut ainsi utiliser n'importe quel nombre de fenêtres et de valuateurs sans dégrader l'exécution en temps réel.

Conteneurs FLTK

Les opcodes pour les conteneurs FLTK sont :

- *FLgroup*
- *FLgroupEnd*
- *FLpack*
- *FLpackEnd*
- *FLpanel*
- *FLpanelEnd*
- *FLscroll*
- *FLscrollEnd*
- *FLtabs*
- *FLtabsEnd*

Valuateurs FLTK

Les opcodes pour les valuateurs FLTK sont :

- *FLcount*
- *FLjoy*
- *FLknob*
- *FLroller*
- *FLslider*
- *FLtext*

Autres Contrôleurs Graphiques FLTK

Les opcodes des autres contrôleurs FLTK sont :

- *FLbox*
- *FLbutBank*
- *FLbutton*
- *FLkeyIn*
- *FLhvsBox*
- *FLhvsBoxSetValue*
- *FLmouse*
- *FLprintk*
- *FLprintk2*
- *FLprintk2*
- *FLslidBnk*
- *FLslidBnk2*
- *FLslidBnkGetHandle*
- *FLslidBnkSet*
- *FLslidBnk2Set*
- *FLslidBnk2Setk*
- *FLvalue*
- *FLvslidBnk*
- *FLvslidBnk2*
- *FLxyin*

Modifier l'Apparence des Contrôleurs Graphiques FLTK

Les opcodes suivants modifient l'apparence des contrôleurs graphiques FLTK :

- *FLcolor*
- *FLcolor2*
- *FLhide*
- *FLlabel*
- *FLsetAlign*
- *FLsetBox*
- *FLsetColor*
- *FLsetColor2*
- *FLsetFont*
- *FLsetPosition*
- *FLsetSize*
- *FLsetText*
- *FLsetTextColor*
- *FLsetTextSize*
- *FLsetTextType*
- *FLsetVal_i*
- *FLsetVal*
- *FLshow*

Opcodes Généraux relatifs aux Contrôleurs Graphiques FLTK

Les opcodes généraux relatifs aux contrôleurs graphiques FLTK sont :

- *FLgetsnap*
- *FLloadsnap*
- *FLrun*
- *FLsavesnap*
- *FLsetsnap*
- *FLupdate*

- *FLsetSnapGroup*

Appel d'Instrument

Les opcodes que l'on peut utiliser pour créer des évènements de partition depuis un orchestre sont :

- *event*
- *event_i*
- *scoreline_i*
- *scoreline*
- *schedule*
- *schedwhen*
- *schedkwhen*
- *schedkwhennamed*

L'opcode *mute* peut être utilisé pour rendre silencieux/sonore un instrument pendant une exécution.

Les définitions d'intrument peuvent être supprimées au moyen de l'opcode *remove*.

Contrôle Séquentiel d'un Programme

Les opcodes pour modifier l'ordre d'exécution des instructions de l'orchestre sont :

- *cggoto*
- *cigoto*
- *ckgoto*
- *cngoto*
- *elseif*
- *else*
- *endif*
- *goto*
- *if*
- *igoto*
- *kgoto*
- *tigoto*

- *timout*

Les opcodes pour créer des structures de boucle sont :

- *loop_ge*
- *loop_gt*
- *loop_le*
- *loop_lt*

Avertissement

Certains de ces opcodes fonctionnent au taux-i même s'ils contiennent des comparaisons aux taux-k ou -a. Voir la section *Réinitialisation*.

Controle de l'Exécution en Temps Réel

Les opcodes qui surveillent et contrôlent l'exécution en temps réel sont :

- *active*
- *cpuprc*
- *maxalloc*
- *prealloc*
- *jacktransport*

Le processus csound en cours peut être terminé au moyen de *exitnow*.

Initialisation et Réinitialisation

Les opcodes utilisés pour l'initialisation des variables sont :

- *init*
- *tival*
- =
- *pset*

Les opcodes qui peuvent générer une autre passe d'initialisation sont :

- *reinit*

- *rigoto*
- *rireturn*

L'opcode *p* peut être utilisé pour lire les valeurs des p-champs aux taux-i ou -k.

nstrnum retourne le numéro d'instrument d'un instrument nommé.

Détection et Contrôle

Contrôleurs graphiques TCL/TK

- *button*
- *checkbox*
- *control*
- *setctrl*

Détection clavier et souris

- *sensekey* (aussi appelé *sense*)
- *xyin*

Suiveurs d'enveloppe

- *follow*
- *follow2*
- *peak*
- *rms*

Estimation de Tempo et de Hauteur

- *ptrack*
- *pitch*
- *pitchamdf*

- *tempest*

Tempo et Séquencement

- *tempo*
- *miditempo*
- *tempoval*
- *seqtime*
- *seqtime2*
- *trigger*
- *trigseq*
- *timedseq*
- *changed*

Système

- *getcfg*

Contrôle de la partition

- *rewindscore*
- *setscorepos*

Piles

Csound implémente une pile globale qui peut être manipulée par les opcodes suivants :

- *stack*
- *pop*
- *push*
- *pop_f*
- *push_f*

Contrôle de sous-instrument

Ces opcodes permettent la définition et l'utilisation d'un sous-instrument :

- *subinstr*
- *subinstrinit*

Voir aussi les sections *UDO* et *Macros d'Orchestre* pour des fonctionnalités similaires.

Lecture du Temps

Les opcodes que l'on peut utiliser pour lire des valeurs temporelles sont :

- *readclock*
- *rtclock*
- *timeinstk*
- *timeinsts*
- *times*
- *timek*

On peut obtenir la date du système au moyen de :

- *date* - Retourne le nombre de secondes écoulées depuis le 1er janvier 1970.
- *dates* - Retourne sous format chaîne la date et le temps spécifiés.

On peut aussi mettre en place des compteurs au moyen de *clockoff* et de *clockon*.

Contrôle des Tables de Fonction

Se reporter aux sections *Instruction de partition f*, *ftgen*, *ftgentmp* et *Routines GEN* pour savoir comment créer des tables.

On peut supprimer des tables de la mémoire au moyen de l'opcode *ftfree*.

Pour savoir comment accéder aux tables, consulter la section *Accès aux Tables*.

Les tables à utiliser avec l'opcode *loscilx* peuvent être chargées au moyen de *sndload*.

Les tables requièrent par défaut une taille qui est une puissance de deux. On peut cependant générer des tables de n'importe quelle taille en spécifiant celle-ci comme un nombre négatif (voir l'*instruction de partition f*).

Note

Certains opcodes n'acceptent pas des tables dont la taille n'est pas une puissance de deux, car ceci peut être une nécessité pour le traitement interne.

Requêtes sur une Table

Les opcodes qui permettent d'obtenir des informations sur une table sont :

- Pour les tables chargées avec le contenu d'un fichier son (au moyen de *GEN01*) : *ftchnls*, *ftlen*, *ftlptim* et *ftsr*
- Pour n'importe quelle table : *nsamp*, *ftlen*, *tableng*

Opérations de Lecture/Ecriture de Table

Les opcodes pour la lecture et l'écriture dans une table sont :

- *ftloadk*
- *ftload*
- *ftsavek*
- *ftsave*
- *tablecopy*
- *tablegpw*
- *tableicopy*
- *tableigpw*
- *tableimix*
- *tableiw*

- *tablemix*
- *tablera*
- *tablew*
- *tablewa*
- *tablewkt*
- *tabmorph*
- *tabmorpha*
- *tabmorphak*
- *tabmorphi*
- *tabrec*
- *tabplay*
- *ftmorf*

Les valeurs d'une table peuvent être lues depuis une expression grâce à la famille d'opcodes *tb*.

Plusieurs oscillateurs sont en fait des lecteurs de table spécialisés. Voir la section *Oscillateurs Elémentaires*.

Lecture de Table avec Sélection Dynamique

Les opcodes qui permettent de sélectionner des tables dynamiquement (au taux-k) sont :

- *tableikt*
- *tablekt*
- *tablexkt*

Opérations Mathématiques

Conversion d'Amplitude

Les opcodes pour opérer des conversions entre différentes mesures d'amplitude sont :

- *ampdb*
- *ampdbfs*
- *db*
- *dbamp*
- *dbfsamp*

Utiliser *rms* pour trouver la valeur de la moyenne quadratique d'un signal. Voir aussi *0dbfs* pour un autre moyen de gérer les amplitudes dans csound.

Opérations Arithmétiques et Logiques

Les opcodes qui effectuent les opérations arithmétiques et logiques sont : -, +, &&, ||, *, /, ^ et %.

Voir aussi la section *Valeurs Conditionnelles* et la famille des opcodes *if* pour l'utilisation des opérateurs logiques.

Comparateurs et Accumulateurs

Les opcodes suivants effectuent la comparaison entre des signaux de taux-a ou de taux-k, trouvent les maxima ou les minima, ou accumulent les résultats de plusieurs calculs ou comparaisons :

- *max*
- *max_k*
- *maxabs*
- *maxabsaccum*
- *maxaccum*
- *min*
- *minabs*
- *minabsaccum*
- *minaccum*
- *vincr*
- *clear*

Fonctions Mathématiques

Les opcodes qui réalisent les fonctions mathématiques sont :

- *abs*
- *ceil*
- *exp*
- *floor*
- *frac*
- *int*
- *log*
- *log10*
- *logbtwo*
- *pow*
- *powershape*
- *powoftwo*
- *round*
- *sqrt*

Opcodes Equivalents à des Fonctions

Les opcodes suivants sont équivalents à des fonctions mathématiques :

- *chebyshevpoly*
- *divz*
- *mac*
- *maca*
- *polynomial*
- *pow*
- *product*
- *sum*
- *taninv2*

Fonctions aléatoires

Les opcodes qui effectuent des fonctions aléatoires sont :

- *birnd*
- *rnd*

Voir la section *Générateurs de Nombres Aléatoires (Bruit)* pour les opcodes qui génèrent des signaux aléatoires.

Fonctions Trigonométriques

Les opcodes qui effectuent les fonctions trigonométriques sont :

- *cos*, *cosh* et *cosinv*
- *sin*, *sinh* et *sininv*
- *tan*, *tanh*, *taninv*et *taninv2*.

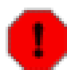
Linear Algebra Opcodes

Linear Algebra Opcodes — Arithmétique scalaire, vectorielle et matricielle sur des valeurs réelles et complexes.

Description

Ces opcodes implémentent plusieurs opérations d'algèbre linéaire, depuis l'arithmétique scalaire, vectorielle et matricielle jusqu'aux décompositions en valeurs propres basées sur la décomposition QR. Les opcodes sont conçus pour le traitement numérique du signal, et bien sûr pour d'autres opérations mathématiques, dans le langage d'orchestre de Csound.

L'implémentation numérique utilise la bibliothèque gmm++ de *home.gna.org/getfem/gmm_intro* [http://home.gna.org/getfem/gmm_intro].



Avertissement

Pour les applications avec des variables f-sig, l'arithmétique sur les tableaux ne peut être exécutée que si le f-sig est "actuel", car le taux-f est une fraction du taux-k ; ce caractère actuel peut être déterminé avec l'opcode la_k_current_f.

Pour les applications que utilisent des affectations entre vecteurs réels et variables de taux-a, l'arithmétique sur les tableaux ne peut être exécutée que si les vecteurs sont "actuels", car la taille du vecteur peut être un multiple entier de ksmps ; ce caractère actuel peut être déterminé au moyen de l'opcode la_k_current_vr.

Tableau 5. Types de Données de l'Algèbre Linéaire

Type Mathématique	Code	Type(s) de Csound Correspondant(s)
scalaire réel	r	variable de taux-i ou de taux-k
scalaire complexe	c	paire de variables de taux-i ou de taux-k, par exemple "kr, ki"
vecteur réel	vr	variable de taux-i contenant l'adresse d'un tableau
vecteur réel	a	variable de taux-a
vecteur réel	t	numéro d'une table de fonction
vecteur complexe	vc	variable de taux-i contenant l'adresse d'un tableau
vecteur complexe	f	variable fsig
matrice réelle	mr	variable de taux-i contenant l'adresse d'un tableau
matrice complexe	mc	variable de taux-i contenant l'adresse d'un tableau

Tous les tableaux sont indexés à partir de 0 ; le premier indice parcourt les lignes pour donner les colonnes, le deuxième indice parcourt les colonnes pour donner les éléments.

Tous les tableaux sont généraux et denses ; les routines pour les matrices bande, hermitiennes, symétriques et creuses ne sont pas implémentées.

Un tableau peut avoir pour code de type vr, vc, mr ou mc et il est stocké dans un objet de taux-i. Dans le code de l'orchestre, un tableau est passé comme une variable MYFLT de taux-i qui contient l'adresse de l'objet tableau, celui-ci étant stocké dans l'espace d'allocation de l'instance de l'opcode. Bien que les variables tableau soient de taux-i, leurs valeurs et même leur forme peuvent être modifées au taux-i ou au taux-k.

Tous les opérandes doivent être pré-alloués ; à l'exception des opcodes de création, aucun opcode n'alloue de tableau. Ceci reste vérifié même si le tableau apparaît à gauche d'un opcode ! Cependant, certaines opérations peuvent reformater les tableaux pour y faire entrer leurs résultats.

Les tableaux sont libérés automatiquement lorsque leur instrument est libéré.

Afin d'améliorer l'exécution et aussi de rendre plus aisée la mémorisation des noms des opcodes, le taux d'exécution, les types de valeur en sortie, les noms des opérations et les types de valeur en entrée sont encodés explicitement dans le nom de l'opcode :

1. "la" pour "famille d'opcode d'algèbre linéaire".
2. "i" ou "k" pour le taux d'exécution.
3. Code(s) de type (voir ci-dessus) pour la ou les valeurs de sortie, seulement si le type n'est pas déduit implicitement des valeurs en entrée.
4. Nom d'opération : nom mathématique usuel (de préférence) ou abréviation.
5. Code(s) de type pour les valeurs en entrée, s'ils ne sont pas implicites.

Pour plus de détails, voir la documentation de gmm++ à *http://download.gna.org/getfem/doc/gmmuser.pdf*.

Syntaxe

Création de Tableau

```
ivr                    la_i_vr_create        irows
```

Crée un vecteur réel de *irows* lignes.

```
ivc                    la_i_vc_create        irows
```

Crée un vecteur complexe de *irows* lignes.

```
imr                    la_i_mr_create        irows, icolumns  [, odiagonal]
```

Crée une matrice réelle de *irows* lignes et *icolumns* colonnes, avec une valeur facultative sur sa diagonale.

```
imc                    la_i_mc_create        irows, icolumns  [, odiagonal_r, odiagonal_i]
```

Crée une matrice complexe de *irows* lignes et *icolumns* colonnes, avec une valeur facultative sur sa diagonale.

Examen de Tableau

```
irows                     la_i_size_vr        ivr
```

Retourne le nombre de lignes du vecteur réel *ivr*.

```
irows                     la_i_size_vc        ivc
```

Retourne le nombre de lignes du vecteur complexe *ivc*.

```
irows, icolumns           la_i_size_mr        imr
```

Retourne le nombre de lignes et de colonnes de la matrice réelle *imr*.

```
irows, icolumns           la_i_size_mc        imc
```

Retourne le nombre de lignes et de colonnes de la matrice complexe *imc*.

```
kfiscurrent               la_k_current_f      fsig
```

Retourne 1 si le fsig est actuel, c'est-à-dire si la valeur du fsig changera lors de la prochaine période-k.

```
kvriscurrent              la_k_current_vr     ivr
```

Retourne 1 si le vecteur réel est actuel, c'est-à-dire, si la trame d'échantillon actuelle de Csound se trouve à l'indice 0 du vecteur.

```
                          la_i_print_vr       ivr
```

Affiche la valeur du vecteur réel *ivr*.

```
                          la_i_print_vc       ivc
```

Affiche la valeur du vecteur complexe *ivc*.

```
                          la_i_print_mr       imr
```

Affiche la valeur de la matrice réelle *imr*.

```
                          la_i_print_mc       imc
```

Affiche la valeur de la matrice complexe *imc*.

Affectation et Conversion de Tableau

```
ivr                       la_i_assign_vr      ivr
```

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux-i.

```
ivr                       la_k_assign_vr      ivr
```

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux-k.

```
ivc                       la_i_assign_vc      ivc

ivc                       la_k_assign_vc      ivr
```

imr	**la_i_assign_mr**	imr
imr	**la_k_assign_mr**	imr
imc	**la_i_assign_mc**	imc
imc	**la_k_assign_mc**	imr

Avertissement

Les affectations vers des vecteurs à partir de tables ou de fsigs peuvent reformater les vecteurs.

Les affectations vers des vecteurs à partir de variables de taux-a, ou vers des variables de taux-a à partir de vecteurs, seront exécutées de manière incrémentielle, un bloc de ksmps éléments par période-k. C'est pourquoi l'arithmétique vectorielle sur ces vecteurs ne peut être pratiquées que si ceux-ci sont actuels, selon la détermination par l'opcode *la_k_currrent_vr*.

ivr	**la_k_assign_a**	asig
ivr	**la_i_assign_t**	itablenumber
ivr	**la_k_assign_t**	itablenumber
ivc	**la_k_assign_f**	fsig
asig	**la_k_a_assign**	ivr
itablenum	**la_i_t_assign**	ivr
itablenum	**la_k_t_assign**	ivr
fsig	**la_k_f_assign**	ivc

Remplissage des Tableaux par des Eléments Aléatoires

ivr	**la_i_random_vr**	[ifill_fraction]
ivr	**la_k_random_vr**	[kfill_fraction]
ivc	**la_i_random_vc**	[ifill_fraction]
ivc	**la_k_random_vc**	[kfill_fraction]
imr	**la_i_random_mr**	[ifill_fraction]
imr	**la_k_random_mr**	[kfill_fraction]
imc	**la_i_random_mc**	[ifill_fraction]
imc	**la_k_random_mc**	[kfill_fraction]

Accès aux Eléments d'un Tableau

ivr **la_i_vr_set** irow, ivalue

kvr **la_k_vr_set** krow, kvalue

ivc **la_i_vc_set** irow, ivalue_r, ivalue_i

kvc **la_k_vc_set** krow, kvalue_r, kvalue_i

imr **la_i mr_set** irow, icolumn, ivalue

kmr **la_k mr_set** krow, kcolumn, ivalue

imc **la_i_mc_set** irow, icolumn, ivalue_r, ivalue_i

kmc **la_k_mc_set** krow, kcolumn, kvalue_r, kvalue_i

ivalue **la_i_get_vr** ivr, irow

kvalue **la_k_get_vr** ivr, krow

ivalue_r, ivalue_i **la_i_get_vc** ivc, irow

kvalue_r, kvalue_i **la_k_get_vc** ivc, krow

ivalue **la_i_get_mr** imr, irow, icolumn

kvalue **la_k_get_mr** imr, krow, kcolumn

ivalue_r, ivalue_i **la_i_get_mc** imc, irow, icolumn

kvalue_r, kvalue_i **la_k_get_mc** imc, krow, kcolumn

Opérations sur un Tableau

imr **la_i_transpose_mr** imr

imr **la_k_transpose_mr** imr

imc **la_i_transpose_mc** imc

imc **la_k_transpose_mc** imc

ivr **la_i_conjugate_vr** ivr

ivr **la_k_conjugate_vr** ivr

ivc **la_i_conjugate_vc** ivc

ivc **la_k_conjugate_vc** ivc

imr **la_i_conjugate_mr** imr

imr **la_k_conjugate_mr** imr

imc **la_i_conjugate_mc** imc

imc **la_k_conjugate_mc** imc

Opérations scalaires

ir **la_i_norm1_vr** ivr

kr **la_k_norm1_vr** ivc

ir **la_i_norm1_vc** ivc

kr **la_k_norm1_vc** ivc

ir **la_i_norm1_mr** imr

kr **la_k_norm1_mr** imr

ir **la_i_norm1_mc** imc

kr **la_k_norm1_mc** imc

ir **la_i_norm_euclid_vr** ivr

kr **la_k_norm_euclid_vr** ivr

ir **la_i_norm_euclid_vc** ivc

kr **la_k_norm_euclid_vc** ivc

ir **la_i_norm_euclid_mr** mvr

kr **la_k_norm_euclid_mr** mvr

ir **la_i_norm_euclid_mc** mvc

kr **la_k_norm_euclid_mc** mvc

ir **la_i_distance_vr** ivr

kr **la_k_distance_vr** ivr

ir **la_i_distance_vc** ivc

kr **la_k_distance_vc** ivc

ir **la_i_norm_max** imr

kr **la_k_norm_max** imc

ir **la_i_norm_max** imr

kr	**la_k_norm_max**	imc
ir	**la_i_norm_inf_vr**	ivr
kr	**la_k_norm_inf_vr**	ivr
ir	**la_i_norm_inf_vc**	ivc
kr	**la_k_norm_inf_vc**	ivc
ir	**la_i_norm_inf_mr**	imr
kr	**la_k_norm_inf_mr**	imr
ir	**la_i_norm_inf_mc**	imc
kr	**la_k_norm_inf_mc**	imc
ir	**la_i_trace_mr**	imr
kr	**la_k_trace_mr**	imr
ir, ii	**la_i_trace_mc**	imc
kr, ki	**la_k_trace_mc**	imc
ir	**la_i_lu_det**	imr
kr	**la_k_lu_det**	imr
ir	**la_i_lu_det**	imc
kr	**la_k_lu_det**	imc

Opérations sur les Eléments entre Tableaux

ivr	**la_i_add_vr**	ivr_a, ivr_b
ivc	**la_k_add_vc**	ivc_a, ivc_b
imr	**la_i_add_mr**	imr_a, imr_b
imc	**la_k_add_mc**	imc_a, imc_b
ivr	**la_i_subtract_vr**	ivr_a, ivr_b
ivc	**la_k_subtract_vc**	ivc_a, ivc_b
imr	**la_i_subtract_mr**	imr_a, imr_b
imc	**la_k_subtract_mc**	imc_a, imc_b
ivr	**la_i_multiply_vr**	ivr_a, ivr_b

ivc	**la_k_multiply_vc**	ivc_a, ivc_b
imr	**la_i_multiply_mr**	imr_a, imr_b
imc	**la_k_multiply_mc**	imc_a, imc_b
ivr	**la_i_divide_vr**	ivr_a, ivr_b
ivc	**la_k_divide_vc**	ivc_a, ivc_b
imr	**la_i_divide_mr**	imr_a, imr_b
imc	**la_k_divide_mc**	imc_a, imc_b

Produits Scalaires

ir	**la_i_dot_vr**	ivr_a, ivr_b
kr	**la_k_dot_vr**	ivr_a, ivr_b
ir, ii	**la_i_dot_vc**	ivc_a, ivc_b
kr, ki	**la_k_dot_vc**	ivc_a, ivc_b
imr	**la_i_dot_mr**	imr_a, imr_b
imr	**la_k_dot_mr**	imr_a, imr_b
imc	**la_i_dot_mc**	imc_a, imc_b
imc	**la_k_dot_mc**	imc_a, imc_b
ivr	**la_i_dot_mr_vr**	imr_a, ivr_b
ivr	**la_k_dot_mr_vr**	imr_a, ivr_b
ivc	**la_i_dot_mc_vc**	imc_a, ivc_b
ivc	**la_k_dot_mc_vc**	imc_a, ivc_b

Inversion de Matrice

imr, icondition	**la_i_invert_mr**	imr
imr, kcondition	**la_k_invert_mr**	imr
imc, icondition	**la_i_invert_mc**	imc
imc, kcondition	**la_k_invert_mc**	imc

Décompositions et Résolutions de Matrice

ivr	**la_i_upper_solve_mr**	imr [, j_1_diagonal]

ivr	**la_k_upper_solve_mr**	imr [, j_1_diagonal]
ivc	**la_i_upper_solve_mc**	imc [, j_1_diagonal]
ivc	**la_k_upper_solve_mc**	imc [, j_1_diagonal]
ivr	**la_i_lower_solve_mr**	imr [, j_1_diagonal]
ivr	**la_k_lower_solve_mr**	imr [, j_1_diagonal]
ivc	**la_i_lower_solve_mc**	imc [, j_1_diagonal]
ivc	**la_k_lower_solve_mc**	imc [, j_1_diagonal]
imr, ivr_pivot, isize	**la_i_lu_factor_mr**	imr
imr, ivr_pivot, ksize	**la_k_lu_factor_mr**	imr
imc, ivr_pivot, isize	**la_i_lu_factor_mc**	imc
imc, ivr_pivot, ksize	**la_k_lu_factor_mc**	imc
ivr_x	**la_i_lu_solve_mr**	imr, ivr_b
ivr_x	**la_k_lu_solve_mr**	imr, ivr_b
ivc_x	**la_i_lu_solve_mc**	imc, ivc_b
ivc_x	**la_k_lu_solve_mc**	imc, ivc_b
imr_q, imr_r	**la_i_qr_factor_mr**	imr
imr_q, imr_r	**la_k_qr_factor_mr**	imr
imc_q, imc_r	**la_i_qr_factor_mc**	imc
imc_q, imc_r	**la_k_qr_factor_mc**	imc
ivr_eig_vals	**la_i_qr_eigen_mr**	imr, i_tolerance
ivr_eig_vals	**la_k_qr_eigen_mr**	imr, k_tolerance
ivr_eig_vals	**la_i_qr_eigen_mc**	imc, i_tolerance
ivr_eig_vals	**la_k_qr_eigen_mc**	imc, k_tolerance

Avertissement

Une matrice doit être hermitienne si l'on veut calculer ses valeurs propres.

ivr_eig_vals, imr_eig_vecs	**la_i_qr_sym_eigen_mr**	imr, i_tolerance
ivr_eig_vals, imr_eig_vecs	**la_k_qr_sym_eigen_mr**	imr, k_tolerance

```
ivc_eig_vals, imc_eig_vecs  la_i_qr_sym_eigen_mc  imc, i_tolerance
```

```
ivc_eig_vals, imc_eig_vecs  la_k_qr_sym_eigen_mc  imc, k_tolerance
```

Crédits

Michael Gogins

Nouveau dans la version 5.09 de Csound

Conversion des Hauteurs

Fonctions

Les opcodes qui effectuent les fonctions de hauteur communes sont :

- *cent*
- *cpsmidinn*
- *cpsoct*
- *cpspch*
- *octave*
- *octcps*
- *octmidinn*
- *octpch*
- *pchmidinn*
- *pchoct*
- *semitone*

Opcodes de Hauteurs

Les opcodes qui effectuent les fonctions d'accordage sont :

- *cps2pch*
- *cpsxpch*
- *cpstun*
- *cpstuni*

Support MIDI en Temps-Réel

Csound supporte les entrées et les sorties MIDI en temps réel, ainsi que les entrées depuis les fichiers MIDI. L'entrée MIDI en temps réel est activée au moyen de l'option de ligne de commande *-M* (ou *--midi-device*=PERIPHERIQUE). Vous devez spécifier le numéro ou le nom de périphérique après le *-M*. Par exemple, pour utiliser le périphérique numéro 2, vous utiliserez quelque chose comme :

```
csound -M2 monmiditr.csd
```

Vous pouvez trouver les périphériques disponibles en utilisant un numéro trop grand :

```
csound -M99 monmiditr.csd
```

Note

Ceci ne fonctionnera que si le module MIDI peut être atteint par numéro de périphérique. Pour alsa, il faut d'abord trouver le nom du périphérique en utilisant :

```
cat /proc/asound/cards
```

Il faut alors taper quelque chose comme :

```
csound -+rtmidi=alsa -M hw:3 monmiditr.csd
```

La sortie MIDI en temps réel est activée au moyen de *-Q*, avec un numéro ou un nom de périphérique comme c'est montré ci-dessus.

Vous pouvez aussi charger un fichier MIDI en utilisant l'option de ligne de commande *-F* ou *--midifile*=FICHIER. Le fichier MIDI est lu en temps réel, et se comporte comme s'il était joué ou reçu en temps réel. Ainsi le programme csound ne sait pas si l'entrée MIDI vient d'un fichier MIDI ou directement d'une interface MIDI.

Une fois l'entrée et/ou la sortie MIDI activée(s), les opcodes comme *MIDI Input* et *MIDI Output* seront effectifs.

Quand l'entrée MIDI est activée (avec -M ou -F), chaque message de *noteon* entrant génèrera un évènement de note pour un instrument qui a le même numéro que le canal de l'évènement (voir *massign* et *pgmassign* pour changer ce comportement). Cela signifie que les instruments contrôlés par le MIDI sont polyphoniques par défaut, car chaque note génèrera une nouvelle instance de l'instrument.

Voir les opcodes pour l'*Interopérabilité MIDI/Partition* pour savoir comment concevoir des instruments utilisables depuis une partition ou pilotés par le MIDI.

Plusieurs modules MIDI en temps réel sont disponibles, et il faut utiliser l'option *-+rtmidi* (voir *-+rtmidi*), pour spécifier le module. Le module par défaut est portmidi qui fournit des E/S MIDI adéquates sur toutes les plates-formes, cependant, pour des performances améliorées et plus fiables, des modules spécifiques à certaines plates-formes sont également fournis.

Actuellement les modules midi disponibles sont :

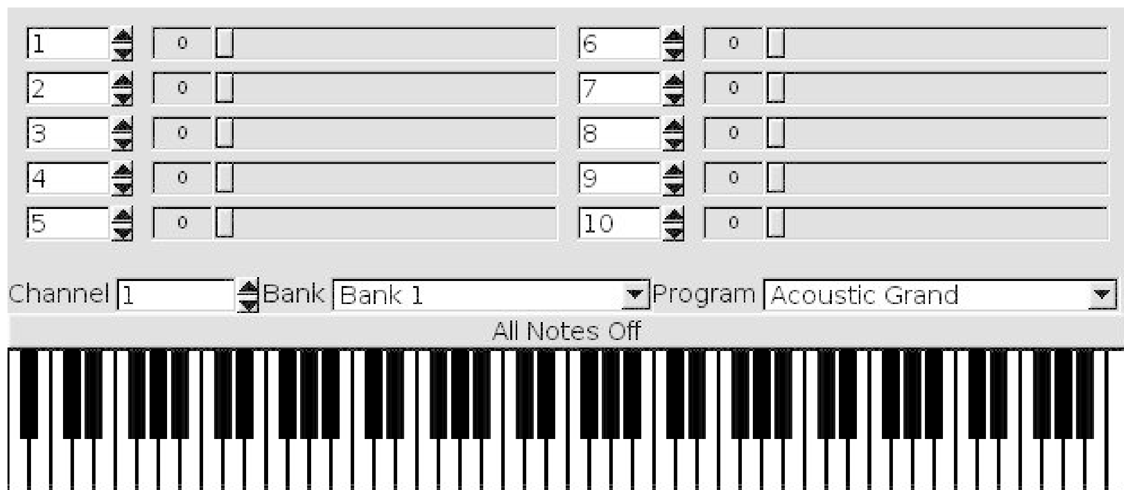
- *alsa* - Pour utiliser le système midi ALSA (seulement sur Linux)

- *winmme* - Pour utiliser le système windows MME (seulement sur Windows)
- *portmidi* - Pour utiliser le système portmidi (sur toutes les plates-formes). C'est le réglage par défaut.
- *virtual* - Pour utiliser un clavier virtuel graphique (voir ci-dessous) comme entrée MIDI (sur toutes les plates-formes)

Astuce

Lors de son exécution, Csound traite la partition puis se termine. S'il n'y a pas d'évènements dans la partition, Csound se termine immédiatement. Si l'on désire n'utiliser que des évèments MIDI au lieu des évènements de partition, il faut demander à Csound de s'exécuter pendant un certain temps au moyen d'une *instruction f* comme "f 0 3600".

Clavier Virtuel MIDI



Clavier Virtuel MIDI.

Le module du clavier virtuel MIDI (activé par l'option -+rtmidi=virtual sur la ligne de commande) fournit un moyen d'envoyer des informations MIDI en temps réel à Csound sans avoir besoin d'un périphérique MIDI. Il peut envoyer des informations de note, des changements de contrôle, des changements de banque et de programme sur un canal spécifié. L'information MIDI en provenance du clavier virtuel est traitée par Csound exactement de la même manière que si elle venait d'autres pilotes MIDI, si bien que si votre orchestre Csound est conçu pour travailler avec des périphériques matériels MIDI, cela marchera aussi.

Le clavier virtuel utilise l'option de périphérique (-M) pour récupérer le nom d'un fichier de mappage du clavier. Comme tous les pilotes MIDI, celui-ci nécessite un périphérique pour être activé. Si l'on désire seulement utiliser les réglages par défaut du clavier, il suffit de passer 0 (c'est-à-dire -M0). Si au lieu de 0 un nom de fichier est donné, le clavier essaiera de charger le fichier pour le mappage du clavier. Si le fichier n'a pas pu être ouvert ou lu correctement, les réglages par défaut seront utilisés.

Les fichiers de Mappage du Clavier permettent à l'utilisateur de personnaliser le nom et le numéro des banques ainsi que le nom et le numéro des programmes d'une banque. L'exemple suivant de mappage de clavier (nommé keyboard.map) a des commentaires intégrés sur le format de fichier. Ce fichier est aussi disponible dans la distribution des sources de Csound dans le répertoire InOut/virtual_keyboard.

```
# Carte de Personnalisation du Clavier pour le Clavier Virtuel
# Steven Yi
#
# USAGE
#
# Lors de l'utilisation du clavier virtuel, vous pouvez fournir un nom de fichier
# pour un mappage des banques et des programmes via l'option -M, par exemple :
#
# csound -+rtmidi=virtual -Mkeyboard.map mon_projet.csd
#
# INFORMATION SUR LE FORMAT
#
# -les lignes commençant par '#' sont des commentaires
# -les lignes avec [] commencent les définitions d'une nouvelle banque,
#  les contenus sont numBanque=nomBanque, avec numBanque=[1,16384]
# -les lignes suivant les instructions de banque sont des définitions de programme
#  dans le format numProgramme=nomProgramme, avec numProgramme=[1,128]
# -les numéros de banque et de programme sont définis dans ce fichier
#  en commençant à 1, mais ils sont convertis en valeurs midi (commençant
#  à 0) lorsqu'ils sont lus
#
# NOTES
#
# -si une définition de banque invalide est trouvée, toutes les
#  définitions de programme qui suivent seront ignorées jusqu'à ce
#  qu'une nouvelle définition de banque valide soit trouvée
# -si une définition valide de banque sans programmes valides est
#  trouvée, elle prendra par défaut les définitions de programme
#  General MIDI
# -si une définition de programme invalide est trouvée, elle sera
#  ignorée

[1=Ma Banque]
1=Mon Patch de Test 1
2=Mon Patch de Test 2
30=Mon Patch de Test 30

[2=Ma Banque2]
1=Mon Patch de Test 1(banque2)
2=Mon Patch de Test 2(banque2)
30=Mon Patch de Test 30(banque2)
```

Les dix réglettes du haut sont affectées par défaut aux contrôleurs MIDI numéro 1-10, mais on peut les changer à volonté. Les numéros de contrôleur et les valeurs de chaque réglette sont fixés par canal, si bien que l'on peut utiliser différents réglages et valeurs pour chaque canal.

Par défaut il y a 128 banques et pour chaque banque 128 patches réglés par défaut sur les noms General Midi. Le standard de banque MIDI utilise une résolution sur 14 bit pour supporter 16384 banques possibles, mais les numéros de banque par défaut sont 0-127. Pour utiliser des valeurs supérieures à 127, il faut utiliser un mappage de clavier personnalisé et fixer la valeur du numéro de banque désiré pour le nom de la banque. Le clavier virtuel transmettra correctement le numéro de banque comme MSB et LSB avec les contrôleurs 0 et 32.

Outre l'entrée disponible par l'interaction avec la GUI via la souris, on peut aussi déclencher les notes MIDI à partir du clavier ASCII quand la fenêtre du clavier virtuel a le focus. L'arrangement est organisé à la manière d'un traceur et offre deux octaves et une tierce majeure, en partant du do médiant (note MIDI 60). La correspondance entre le clavier ASCII et les valeurs de note MIDI est donnée dans la table suivante.

Tableau 6. Valeurs des Notes MIDI du Clavier ASCII

Touche	Valeur MIDI
z	60
s	61
x	62
d	63

Touche	**Valeur MIDI**
c	64
v	65
g	66
b	67
h	68
n	69
j	70
m	71
q	72
2	73
w	74
3	75
e	76
r	77
5	78
t	79
6	80
y	81
7	82
u	83
i	84
9	85
o	86
0	87
p	88

Voici un exemple de l'utilisation du clavier MIDI virtuel. Il utilise le fichier *virtual.csd* [examples/virtual.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in     Virtual MIDI    -M0 is needed anyway
-odac           -iadc     -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>
; By Mark Jamerson 2007

sr=44100
ksmps=10
nchnls=2

massign 1,1
prealloc 1,10

instr 1  ;Midi FM synth

inote cpsmidi
iveloc ampmidi 10000
idur = 2
    xtratim 1

kgate oscil 1,10,2
```

```
anoise noise 100*inote,.99
acps  samphold anoise,kgate
aosc oscili 1000,acps,1
aout = aosc

; Use controller 7 to control volume
kvol ctrl7 1, 7, 0.2, 1

outs kvol * aout, kvol * aout

endin

</CsInstruments>
<CsScore>
f0 3600
f1 0 1024 10 1
f2 0 16 7 1 8 0 8
f3 0 1024 10 1 .5 .6 .3 .2 .5

e
</CsScore>
</CsoundSynthesizer>
```

Entrée MIDI

Les opcodes suivants peuvent recevoir des informations MIDI :

- Information MIDI pour tous les instruments : *aftouch*, *chanctrl* et *polyaft*, *pchbend*.
- Information MIDI pour les instruments déclenchés par le MIDI : *veloc*, *midictrl* et *notnum*. Voir aussi *Converters*.
- Entrée de Contrôleur MIDI pour tous les instruments : *midic7*, *midic14* et *midic21*.
- Entrée de Contrôleur MIDI pour les instruments déclenchés par le MIDI : *ctrl7*, *ctrl14* et *ctrl21*.
- Valeur d'initialisation de contrôleur MIDI : *initc7*, *initc14*, *initc21* et *ctrlinit*.

massign peut être utilisé pour spécifier l'instrument csound à déclencher par un canal MIDI particulier. *pgmassign* peut être utilisé pour assigner un instrument csound à un programme MIDI spécifique.

Sortie de Message MIDI

Les opcodes qui produisent des sorties MIDI sont :

- *mdelay*
- *nrpn*
- *outiat*
- *outic*
- *outic14*
- *outipat*
- *outipb*
- *outipc*

- *outkat*
- *outkc*
- *outkc14*
- *outkpat*
- *outkpb*
- *outkpc*

Entrée et Sortie Génériques

Les opcodes pour les entrées et les sorties MIDI génériques sont : *midiin* et *midiout*.

Convertisseurs

Les opcodes suivants peuvent convertir de l'information MIDI provenant d'une instance d'un instrument déclenché par le MIDI :

- Convertisseurs de numéros de note MIDI en fréquence : *cpsmidi*, *cpsmidib*, *cpstmid*, *octmidi*, *octmidib*, *pchmidi* et *pchmidib*.
- Convertisseurs de vélocité MIDI en amplitude : *ampmidi*.

Extension d'Evènements

Les opcodes qui permettent d'étendre la durée d'un évènement sont :

- *release*
- *xtratim*

Sortie de Note-on/Note-off

Les opcodes pour sortir des messages MIDI noteon ou noteoff sont :

- *midion*
- *midion2*
- *moscil*
- *noteoff*
- *noteon*
- *noteondur*

- *noteondur2*

Opcodes pour l'Interopérabilité MIDI/Partition

Les opcodes suivants peuvent être utilisés pour concevoir des instruments qui fonctionnent de manière interchangeable avec du MIDI en temps réel et avec des évènements de partition :

- *midichannelaftertouch*
- *midichn*
- *midicontrolchange*
- *mididefault*
- *midinoteoff*
- *midinoteoncps*
- *midinoteonkey*
- *midinoteonoct*
- *midinoteonpch*
- *midipitchbend*
- *midipolyaftertouch*
- *midiprogramchange*.

Adapter un instrument Csound déclenché par une partition.

Pour adapter un instrument Csound ordinaire conçu pour être activé depuis une partition, à l'interopérabilité partition/MIDI :

- Changer tous les opcodes *linen*, *linseg*, et *expseg* respectivement en *linenr*, *linsegr*, et *expsegr*, sauf pour une enveloppe de décliquage ou d'atténuation. Cela ne changera en rien les exécutions pilotées par une partition.
- Ajouter les lignes suivantes au début de la définition de l'instrument :

```
; Pour être sûr qu'un instrument activé par le MIDI
; aura un champ p3 positif.
mididefault 60, p3
; Met le numéro de touche MIDI traduit en cycles par
; seconde dans p4, et la vélocité MIDI dans p5
midinoteoncps p4, p5
```

  Bien entendu, *midinoteoncps* pourrait être changé en *midinoteonoct* ou tout autre option, et le choix des p-champs est arbitraire.

Options de ligne de commande d'Entrée/Sortie MIDI en Temps Réel

Les nouvelles *options d'E/S MIDI* dans Csound 5.02, peuvent remplacer la plupart des utilisations de ces opcodes d'interopérabilité, et en rendre l'usage plus facile.

Messages System Realtime

Les opcodes pour les messages MIDI System Realtime sont : *mclock* et *mrtmsg*.

Banques de Réglettes

Les opcodes pour les banques de réglettes de contrôleurs MIDI sont :

- *slider8*
- *slider8f*
- *slider16*
- *slider16f*
- *slider32*
- *slider32f*
- *slider64*
- *slider64f*
- *s16b14*
- *s32b14*
- *sliderKawai*

Les opcodes pour stocker des banques de réglettes de contrôleurs MIDI dans des tables sont :

- *slider8table*
- *slider8tablef*
- *slider16table*
- *slider16tablef*
- *slider32table*
- *slider32tablef*
- *slider64table*
- *slider64tablef*

Traitement Spectral

voir la section *Synthèse/Resynthèse Additive* pour les opcodes élémentaires de resynthèse.

Resynthèse par Tranformée de Fourier à Court-Terme (STFT)

Utilisation des fichiers PVOC-EX avec les anciens opcodes pvoc de Csound

Tous les opcodes pvoc originaux peuvent lire maintenant des fichiers PVOC-EX, aussi bien que le format de fichier natif non portable. Comme un fichier PVOC-EX utilise une fenêtre d'analyse de taille double, les utilisateurs trouveront sans doute que le résultat est utilement amélioré, pour certains sons et certains traitements, malgré le fait que la resynthèse n'utilise pas la même taille de fenêtre.

En dehors du paramètre de taille de fenêtre, la différence principale entre le format original .pv et PVOC-EX est l'intervalle d'amplitude des trames d'analyse. Lorque la pondération est appliquée, afin qu'il n'y ait pas de différences notables dans le niveau de sortie, quelque soit le format de fichier utilisé, de légères pertes d'amplitude peuvent encore se produire, car l'utilisation d'une fenêtre double modifie l'amplitude des trames, sans que le code de resynthèse en tienne compte. Noter que tous les opcodes pvoc originaux attendent un fichier d'analyse mono, et que les fichiers PVOC-EX multi-canaux seront ainsi réjetés.

Les opcodes qui implémentent la resynthèse STFT sont :

- *tableseg*
- *pvadd*
- *pvbufread*
- *pvcross*
- *pvinterp*
- *pvoc*
- *pvread*
- *tableseg*
- *tablexseg*
- *vpvoc*

L'utilitaire *PVANAL* permet de générer les fichiers d'analyse pv.

Resynthèse par Codage Prédictif Linéaire

(LPC)

Les opcodes de resynthèse par prédiction linéaire sont :

- *lpfreson*
- *lpinterp*
- *lpread*
- *lpreson*
- *lpslot*

On peut créer des fichiers d'analyse LPC au moyen de l'utilitaire *LPANAL*.

Traitement Spectral Non-standard

Ces unités génèrent et traitent des types de données de signaux non-standard, tels que des signaux de contrôle du domaine temporel et des signaux audio sous-échantillonnés, et leur représentation dans le domaine fréquentiel (spectrale). Les types de données (*d*-, *w*-) se définissent par eux-mêmes et leur contenu n'est pas utilisable par les autres unités de Csound. Ces générateurs unitaires sont expérimentaux, et sujets à modification entre les différentes versions de Csound ; ils seront aussi complétés par d'autres unités plus tard.

Les opcodes pour le traitement spectral non-standard sont *specaddm*, *specdiff*, *specdisp*, *specfilt*, *spechist*, *specptrk*, *specscal*, *specsum* et *spectrum*.

Outils pour le Traitement Spectral en Temps Réel (opcodes pvs)

Avec ces opcodes, deux nouvelles facilités fondamentales sont ajoutées à Csound. Ils offrent une qualité audio améliorée, et une exécution rapide, permettant une analyse et une resynthèse de grande qualité (avec les transformations) à appliquer en temps réel aux signaux instantanés. Le vocodeur de phase original de Csound n'est pas changé ; les nouveaux opcodes utilisent un ensemble de fonctions complètement séparé basé sur « pvoc.c » dans la distribution CARL, écrite par Mark Dolson.

Les utilitaires de Csound *dnoise* et *srconv* (également par Dolson, de CARL) utilisent aussi ce moteur pvoc. pvoc de CARL est aussi la base pour le vocodeur de phase inclu dans le Composer's Desktop Project. Quelques petites modifications, mais importantes, ont été apportées au code CARL original pour supporter les flots de données en temp réel.

1. Support du nouveau format de fichier d'analyse PVOC-EX. C'est un format totalement portable et ouvert (multi plates-formes), supportant trois formats d'analyse, et les signaux multi-canaux. Actuellement seul le format standard amplitude+fréquence a été implémenté dans les opcodes, mais le format de fichier lui-même supporte les formats amplitude+phase et le format complexe (réel-imaginaire). En plus des nouveaux opcodes, les opcodes pvoc originaux de Csound ont été étendus (avec pour conséquence une qualité audio améliorée dans certains cas) pour lire les fichiers PVOC-EX aussi bien que le format original (non portable).

   Les détails complets de la structure d'un fichier PVOC-EX son disponibles sur le site web : *http://www.cs.bath.ac.uk/~jpff/NOS-DREAM/researchdev/pvocex/pvocex.html*. Ce site donne aussi

les détails des programmes de console disponibles librement *pvocex* et *pvocex2* qui peuvent être utilisés pour créer des fichiers PVOC-EX dans tous les formats supportés.

2. Un nouveau type de signal du domaine fréquentiel, totalement transportable par flot de données, avec *f* comme premier caractère. Dans ce document on y fait référence par *fsig*. Le support principal des fsigs est fourni par les opcodes *pvsanal* et *pvsynth*, qui effectuent l'analyse et la resynthèse traditionnelles par chevauchement-addition avec un vocodeur de phase, indépendamment du taux de contrôle de l'orchestre. La seule obligation est que le taux de contrôle *kr* soit supérieur ou égal au taux d'analyse, ce qui peut s'exprimer par *ksmps <= overlap*, où *overlap* est la distance en échantillons entre deux trames d'analyse, comme spécifié pour *pvsanal*. Comme *overlap* vaut typiquement au moins 128, et plus souvent 256, ce n'est pas une restriction coûteuse en pratique. L'opcode *pvsinfo* peut être utilisé au moment de l'initialisation pour acquérir les propriétés d'un fsig.

   Le fsig permet la séparation nominale entre les étapes d'analyse et de resynthèse du vocodeur de phase pour une mise à disposition du programmeur Csound, ce qui permet non seulement d'employer des alternatives pour l'une ou les deux de ces étapes (pas seulement la resynthèse par banc d'oscillateur, mais aussi la génération synthétique de flots de données fsig), mais aussi les opcodes opérant sur le flot fsig peuvent être eux-mêmes plus élémentaires. Ainsi le fsig permet la création d'un véritable environnement de plugin de flots de données pour les signaux du domaine fréquentiel. Avec les vieux opcodes pvoc, chaque opcode doit pouvoir agir comme un resynthétiseur, si bien que des facilités comme la transposition de hauteur sont dupliquées dans chaque opcode ; et dans la plupart des cas les opcodes ont beaucoup de paramètres. La séparation des étapes d'analyse et de synthèse au moyen du fsig encourage le développement d'une grande variété d'opcodes qui sont des briques élémentaires implémentant une ou deux fonctions, et avec lesquelles on peut construire des processus plus élaborés.

Cette réalisation en est encore à ses débuts et présente un caractère expérimental, et il est possible que la définition précise des opcodes change en réponse aux avis des utilisateurs. De plus, de nombreuses nouvelles possibilités d'opcode sont ouvertes ; ces facteurs peuvent aussi avoir une influence rétrospective sur les opcodes présentés ici.

Noter que certains paramètres d'opcode ont actuellement une implémentation restreinte ou manquante. Ceci, au moins en partie, afin de préserver la simplicité des opcodes à ce niveau, et aussi parce qu'ils concernent d'importantes questions de conception pour lesquelles aucun décision n'a encore été prise, et pour lesquelles l'opinion des utilisateurs est souhaitée.

Un point important au sujet de ce nouveau type de signal est que, parce que le taux d'analyse est typiquement très inférieur à *kr*, les nouvelles trames d'analyse ne sont pas disponibles à chaque k-cycle. En interne, les opcodes tracent *ksmps*, et maintiennent également un compteur de trames, afin que les trames soient lues et écrites aux bons moments ; ce processus est généralement transparent pour l'utilisateur. Cependant, cela signifie que les signaux de taux-k n'agissent sur un fsig qu'au taux d'analyse, pas à chaque k-cycle. L'opcode *pvsftw* retourne un drapeau au taux-k qui est positionné lorque de nouvelles données fsig sont disponibles.

A cause de la nature du système de chevauchement-addition, l'utilisation des ces opcodes infère un délai, ou latence, petit mais significatif déterminé par la taille de la fenêtre (max(ifftsize,iwinsize)). Il vaut typiquement 23ms. Dans cette première réalisation, le délai dépasse légèrement le minimum théorique, et l'on espère qu'il pourra être réduit, lorsque les opcodes seront optimisés pour le transport par flot de données en temps-réel.

Les opcodes pour le traitement spectral en temps réel sont *pvsadsyn*, *pvsanal*, *pvscross*, *pvsfread*, *pvsftr*, *pvsftw*, *pvsinfo*, *pvsmaska* et *pvsynth*.

De plus il y a un certain nombre d'opcodes disponibles sous forme de plugins dans Csound 5. Ce sont *pvsdiskin*, *pvscent*, *pvsdemix*, *pvsfreeze*, *pvsbuffer*, *pvsbufread*, *pvscale*, *pvshift*, *pvsifd*, *pvsinit*, *pvsin*, *pvsout*, *pvsosc*, *pvsbin*, *pvsdisp*, *pvsfwrite*, *pvsmix*, *pvsmooth*, *pvsfilter*, *pvsblur*, *pvstencil*, *pvsarp*, *pvsvoc*, *pvsmorph pvsbandp pvsbandr*

Un certain nombre d'opcodes sont conçus pour générer et traiter des flots de données de pistes de partiels. Ce sont *partials*, *trcross*, *trfilter*, *trsplit*, *trmix*, *trscale*, *trshift*, *trlowest*, *trhighest tradsyn*, *sinsyn*, *resyn*, *binit*

Voir la section *Piles* pour une information sur les opcodes qui peuvent empiler les signaux de type f.

Traitement Spectral avec ATS

Ces opcodes peuvent lire, transformer et resynthétiser des fichiers d'analyse ATS. Prière de noter que l'application ATS est nécessaire pour produire les fichiers d'analyse. Voici un extrait du Manuel de Référence d'ATS.

« ATS est une bibliothèque de fonctions pour l'Analyse spectrale, la Transformation et la Synthèse du son basée sur un modèle sinusoïdal plus du bruit de bande critique. Un son dans ATS est un objet symbolique représentant un modèle spectral qui peut être sculpté au moyen de diverses fonctions de transformation. »

Pour plus d'information sur ATS visiter : http://www-ccrma.stanford.edu/~juan/ATS.html.

Les fichiers d'analyse ATS peuvent être produits avec le logiciel ATS ou l'utilitaire csound *ATSA*.

Les opcodes pour le traitement ATS sont :

- *ATSinfo* : lit les données de l'en-tête d'un fichier ATS.
- *ATSread*, *ATSreadnz*, *ATSbufread*, *ATSinterpread*, *ATSpartialtap* : lisent les données d'un fichier ou d'un tampon ATS.
- *ATSadd*, *ATSaddnz*, *ATScross*, *ATSsinnoi* : Resynthétisent le son.

Crédits

Auteur : Alex Norman
Seattle, Washington
2004

Opcodes Loris

Note

Ces opcodes sont un composant facultatif de Csound5. Pour savoir s'ils sont installés utilisez la commande 'csound -z' qui donne la liste des opcodes disponibles.

La famille des opcodes Loris encapsule : *lorisread* qui importe un ensemble de partiels à largeur de bande adaptée depuis un fichier de données au format SDIF, en appliquant, au taux de contrôle, des enveloppes de pondération de fréquence, d'amplitude et de largeur de bande, et qui stocke les partiels modifiés en mémoire ; *lorismorph*, qui opère une transformation (morphing) entre deux ensembles stockés de partiels à largeur de bande adaptée et stocke un nouvel ensemble de partiels représentant le son transformé. La transformation est réalisée en en interpolant linéairement les enveloppes des paramètres (fréquence, amplitude, et largeur de bande, ou aspect bruiteux) des partiels à largeur de bande adaptée selon des fonctions de transformation de la fréquence, de l'amplitude et de la largeur de bande, variant au taux de contrôle ; *lorisplay*, qui restitue un ensemble de partiels à largeur de bande adaptée en utili-

sant la méthode de Synthèse Additive à Largeur de Bande Adaptée implémentée dans le logiciel Loris, avec application d'enveloppes de pondération de fréquence, d'amplitude, et de largeur de bande, variant au taux de contrôle.

Pour plus d'information sur la transformation du son et sa manipulation avec Loris et le Modèle Additif à Largeur de Bande Adaptée Réassignée, visiter le site web de Loris à www.cerlsoundgroup.org/Loris [http://www.cerlsoundgroup.org/Loris].

Exemples

Exemple 3. Jouer les partiels sans modification

```
;
; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; et sans modification de fréquence, d'amplitude,
; ou de largeur de bande.
;
instr 1
    ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes
             lorisread   ktime, "clarinet.sdif", 1, 1, 1, 1, .001
    asig     lorisplay   1, 1, 1, 1
             out         asig
endin
```

Exemple 4. Ajouter une intonation et un vibrato

```
; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; ajout d'une intonation et d'un vibrato, accroissement
; du "souffle" (aspect bruiteux) et de l'amplitude
; générale, et ajout d'un filtre passe-haut.
;
instr 2
    ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes

    ; calcule le rapport de fréquence pour l'intonation
    ; (la hauteur originale était sol#3)
    ifscale  =           cpspch(p4)/cpspch(8.08)

    ; faire une enveloppe de vibrato
    kvenv    linseg      0, p3/6, 0, p3/6, .02, p3/3, .02, p3/6, 0, p3/6, 0
    kvib     oscil       kvenv, 4, 1   ; table 1, sinusoid

    kbwenv   linseg      1, p3/6, 1, p3/6, 2, 2*p3/3, 2
             lorisread   ktime, "clarinet.sdif", 1, 1, 1, 1, .001
    a1       lorisplay   1, ifscale+kvib, 2, kbwenv
    a2       atone       a1, 1000      ; filtre passe-haut, coupure à 1000 Hz
             out         a2
endin
```

L'instrument du premier exemple synthétise un son de clarinette en utilisant du début à la fin les partiels dérivés de l'analyse à bande adaptée réassignée d'un son de clarinette de trois secondes, stockés dans le fichier `clarinet.sdif`. L'instrument de l'exemple 2 ajoute une intonation et un vibrato au son de clarinette synthétisé par l'instrument 1, renforce son amplitude et son aspect bruiteux, et applique un filtre passe-haut au résultat. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```
    ; créer une sinus dans la table 1
    f 1 0 4096 10 1
```

```
; jouer instr 1
;    début   dur
i 1    0      3
i 1    +      1
i 1    +      6
s

; jouer instr 2
;    début   dur    hauteur
i 2     1     3     8.08
i 2     3.5   1     8.04
i 2     4     6     8.00
i 2     4     6     8.07

e
```

Exemple 5. Transformation de partiels

```
; Transforme les partiels de clarinet.sdif vers
; les partiels de flute.sdif sur la durée de la
; partie tenue des deux notes (de 0,2 à 2,0 secondes
; pour la clarinette, et de 0,5 à 2,1 secondes
; pour la flûte). Les portions d'attaque et de
; chute dans le son transformé sont spécifiées
; par les paramètres p4 et p5, respectivement.
; Le temps de transformation est le temps entre
; l'attaque et la chute. Les partiels de la
; clarinette sont transposés pour s'accorder à
; la hauteur de la note de la flûte (ré au-dessus
; du do médium).
;
instr 1
    ionset   =           p4
    idecay   =           p5
    itmorph  =           p3 - (ionset + idecay)
    ipshift  =           cpspch(8.02)/cpspch(8.08)

      ; fonction temporelle de la clarinette, transformation de 0,2 à 2,0 secondes
    ktcl     linseg      0, ionset, .2, itmorph, 2.0, idecay, 2.1
      ; fonction temporelle de la flûte, transformation de 0,5 à 2,1 secondes
    ktfl     linseg      0, ionset, .5, itmorph, 2.1, idecay, 2.3
    kmurph   linseg      0, ionset, 0, itmorph, 1, idecay, 1
             lorisread   ktcl, "clarinet.sdif", 1, ipshift, 2, 1, .001
             lorisread   ktfl, "flute.sdif", 2, 1, 1, 1, .001
             lorismorph  1, 2, 3, kmurph, kmurph, kmurph
    asig     lorisplay   3, 1, 1, 1
             out         asig
endin
```

Exemple 6. Plus de transformation

```
; Transforme les partiels de trombone.sdif vers les
; partiels de meow.sdif. Les dates de début et de fin
; de la transformation sont spécifiées par les
; paramètres p4 et p5, respectivement. La transformation
; a lieu sur la deuxième des quatre notes dans chaque
; son, de 0,75 à 1,2 secondes pour le trombone flatterzung,
; et de 1,7 à 2,2 secondes pour le miaulement de chat.
; Des fonctions de transformation différentes sont
; utilisées pour les enveloppes de fréquence et
; d'amplitude, afin que l'amplitude des partiels
; ait une transition plus rapide du trombone au
; chat que les fréquences. (Les enveloppes de largeur
; de bande utilisent la même fonction de transformation
; que les amplitudes).
;
instr 2
    ionset   =           p4
```

```
    imorph   =           p5 - p4
    irelease =           p3 - p5

    kttbn    linseg      0, ionset, .75, imorph, 1.2, irelease, 2.4
    ktmeow   linseg      0, ionset, 1.7, imorph, 2.2, irelease, 3.4

    kmfreq   linseg      0, ionset, 0, .75*imorph, .25, .25*imorph, 1, irelease, 1
    kmamp    linseg      0, ionset, 0, .75*imorph, .9, .25*imorph, 1, irelease, 1

             lorisread   kttbn, "trombone.sdif", 1, 1, 1, 1, .001
             lorisread   ktmeow, "meow.sdif", 2, 1, 1, 1, .001
             lorismorph  1, 2, 3, kmfreq, kmamp, kmamp
    asig     lorisplay   3, 1, 1, 1
             out         asig
endin
```

L'instrument dans le premier exemple effectue une transformation du son entre une note de clarinette et une note de flûte en utilisant les partiels à bande adaptée réassignée stockés dans `clarinet.sdif` et dans `flute.sdif`.

La transformation est effectuée sur les portions tenues des notes, 0,2 à 2,0 secondes dans le cas de la note de clarinette et 0,5 à 2,1 secondes dans le cas de la note de flûte. Les fonctions d'index temporel, ktcl et ktfl, alignent les portions d'attaque et de chute des notes avec les temps d'attaque et de chute du son transformé, spécifiées respectivement par les paramètres p4 et p5. L'attaque du son transformé est entièrement composée des données de partiel de la clarinette, et la chute est entièrement composée de données de la flûte. Les partiels de la clarinette sont transposés pour s'accorder à la hauteur de la note de flûte (ré au-dessus du do médium).

L'instrument dans le second exemple effectue une transformation du son entre une note de trombone flatterzung et un miaulement de chat en utilisant les partiels à bande adaptée réassignée stockés dans `trombone.sdif` et `meow.sdif`. Les données dans ces fichiers SDIF ont été réparties par canaux et séparées pour établir une correspondance entre partiels.

Les deux ensembles de partiels sont importés et stockés dans des positions mémoire étiquetées 1 et 2, respectivement. Les deux sons originaux ont quatre notes, et la transformation est effectuée sur la seconde note de chaque son (de 0,75 à 1,2 secondes pour le trombone flatterzung, et de 1,7 à 2,2 secondes pour le miaulement de chat). Les fonctions d'index temporel, kttbn et ktmeow, alignent ces segments des ensembles de partiels source et cible avec les paramètres spécifiés pour la durée du début, de la fin, et totale de la transformation. Deux fonctions de transformation différentes sont utilisées, afin que les amplitudes des partiels et les coefficients de largeur de bande se transforment rapidement des valeurs du trombone aux valeurs du miaulement de chat, tandis que les fréquences opèrent une transition plus graduelle. Les partiels transformés sont stockés dans la position mémoire étiquetée 3 et restitués par l'instruction *lorisplay* qui suit. Ils auraient pu aussi être utilisés comme source pour une autre transformation dans un instrument de transformation à trois étapes. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```
    ; jouer instr 1
    ;    début   dur  attaque  chute
    i 1    0      3     .25      .15
    i 1    +      1     .10      .10
    i 1    +      6    1.       1.
    s

    ; jouer instr 2
    ;    début   dur  début_morph   fin_morph
    i 2    0      4     .75            2.75

    e
```

Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz (loris@cerlsoundgroup.org [mailto:loris@cerlsoundgroup.org]).

Elle est construite d'après une implémentation prototype du générateur unitaire *lorisplay* écrite par Corbin Champion, et basée sur la méthode de Synthèse Additive à Largeur de Bande Adaptée et sur les algorithmes de transformation du son implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés en plugin pour Csound 5 par Michael Gogins.

Chaînes de Caractères

Les variables chaîne de caractères sont des variables dont le nom commence par S ou par gS (pour les variables chaîne locales ou globales, respectivement), et elle peuvent mémoriser n'importe quelle chaîne avec une longueur maximale définie par l'option de ligne de commande -+*max_str_len* (255 caractères par défaut). On peut utiliser ces variables comme argument d'entrée de n'importe quel opcode qui attend une chaîne constante entre apostrophes, et on peut les manipuler durant les périodes d'initilasation ou d'exécution avec les opcodes dont la liste suit.

Il est également possible d'utiliser des chaînes dans les p-champs. Un p-champ chaîne peut être utilisé directement par plusieurs opcodes de l'orchestre, ou il peut être d'abord copié dans une variable chaîne :

```
a1    diskin2 p5, 1
```

```
Snom  strget  p5
a1    diskin2 Snom, 1
```

Les chaînes dans Csound peuvent être exprimées par les doubles apostrophes traditionnelles (" "), mais aussi par {{ }}. La seconde méthode est utile si l'on veut utiliser les caractères ';' et '$' dans la chaîne sans avoir recours aux codes ASCII.

Note

Les variables chaînes et les opcodes correspondants ne sont pas disponibles dans les versions de Csound antérieures à la 5.00.

On peut également lier une chaîne à un numéro au moyen de *strset* et *strget*.

Csound 5 a aussi amélioré l'analyse des constantes chaîne. Il est possible de spécifier une chaîne multilignes en l'entourant avec {{ et }} à la place des habituelles doubles apostrophes (noter que la longueur des constantes chaîne n'est pas limitée, et n'est pas affectée par l'option -+*max_str_len*), et les séquences d'échappement suivantes sont automatiquement converties :

- \a : cloche d'alerte
- \b : retour arrière
- \n : nouvelle ligne
- \r : retour chariot
- \t : tabulation
- \\ : le caractère '\'
- \nnn : le caractère ayant le code ASCII (en octal) nnn

Les chaînes peuvent être utilement employées avec l'opcode *system* :

```
instr 1
 ; csound5 permet de placer une chaîne sur plusieurs lignes dans des accolades doubles
    system {{      ps
            date
            cd ~/Desktop
            pwd
```

```
            ls -l
            whois csounds.com
        }}
endin
```

Et avec les *opcodes python*, entre autres :

```
pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
    if random.random() < p:
        i = int(random.random() * len(pool))
        pool[i] = n
    return random.choice(pool)
}}
```

Opcodes de Manipulation de Chaîne

Ces opcodes effectuent des opérations sur les variables chaîne (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :

- *strcpy* et *strcpyk* - Assignation à une variable chaîne.
- *strcat* et *strcatk* - Concaténation de chaînes, avec mémorisation du résultat dans une variable.
- *strcmp* et *strcmpk* - Comparaison de chaînes.
- *strget* - Assignation à une variable chaîne de la valeur trouvée dans la table strset à l'index spécifié, ou d'un p-champ chaîne de la partition.
- *strlen* et *strlenk* - Retourne la longueur d'une chaîne.
- *sprintf* - conversion de sortie formatée à la manière de printf, avec mémorisation du résultat dans une variable chaîne.
- *sprintfk* - conversion de sortie formatée à la manière de printf, avec mémorisation du résultat dans une variable chaîne au taux-k.
- *puts* - Impression d'une constante ou d'une variable chaîne.
- *strindex* et *strindexk* - Retourne la première occurence d'une chaîne dans une autre chaîne.
- *strrindex* et *strrindexk* - Retourne la dernière occurence d'une chaîne dans une autre chaîne.
- *strsub* et *strsubk* - Retourne une sous-chaîne de la chaîne passée en paramètre.

Opcodes de Conversion de Chaîne

Ces opcodes convertissent des variables chaînes (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :

- *strtod* et *strtodk* - Convertit une valeur de chaîne en une valeur en virgule flottante.

- *strtol* et *strtolk* - Convertit une valeur de chaîne en un entier signé.
- *strchar* et *strchark* - Retourne le code ASCII d'un caractère dans une chaîne.
- *strlower* et *strlowerk* - Convertit une chaîne en minuscules.
- *strupper* et *strupperk* - Convertit une chaîne en majuscules.

Opcodes Vectoriels

La famille des opcodes vectoriels est conçue pour pouvoir traiter des sections de f-table comme des vecteurs pour diverses opérations sur celles-ci.

Opérateurs de Tableaux de Vecteurs

Les opcodes vectoriels suivants supportent les accès en lecture/écriture sur des tableaux de vecteurs (tableaux de tableaux) :

- *vtablei*
- *vtable1k*
- *vtablek*
- *vtablea*
- *vtablewi*
- *vtablewk*
- *vtablewa*
- *vtabi*
- *vtabk*
- *vtaba*
- *vtabwi*
- *vtabwk*
- *vtabwa*

Opérations Entre un Signal Vectoriel et un Signal Scalaire

Ces opcodes effectuent des opérations numériques entre un signal de contrôle vectoriel (contenu dans une f-table), et un signal scalaire. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs de la table. Il y a des versions de ces opcodes de taux-k et de taux-i.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Opérations Entre un Signal Vectoriel et un Signal Scalaire :

- *vadd*
- *vmult*

- *vpow*
- *vexp*
- *vadd_i*
- *vmult_i*
- *vpow_i*
- *vexp_i*

Opérations Entre deux Signaux Vectoriels

Ces opcodes effectuent des opérations entre deux vecteurs, de telle manière que chaque élément du premier vecteur est traité avec l'élément correspondant de l'autre vecteur. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs du vecteur source.

Opérations Entre deux Signaux Vectoriels :

- *vaddv*
- *vsubv*
- *vmultv*
- *vdivv*
- *vpowv*
- *vexpv*
- *vcopy*
- *vmap*
- *vaddv_i*
- *vsubv_i*
- *vmultv_i*
- *vdivv_i*
- *vpowv_i*
- *vexpv_i*
- *vcopy_i*

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Générateurs Vectoriels d'Enveloppe

Les opcodes pour générer des vecteurs contenant des enveloppes sont *vlinseg* et *vexpseg*.

Ces opérateurs sont semblables à *linseg* et *expseg*, mais ils opèrent avec des signaux vectoriels à la place des signaux scalaires.

La sortie est un vecteur dans une f-table (préalablement allouée), tandis que chaque point charnière de l'enveloppe est en fait un vecteur de valeurs. Tous les points charnière doivent contenir le même nombre d'éléments (*ielements*).

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Limitation et Enroulement des Signaux Vectoriels de Contrôle

Les opcodes pour effectuer la limitation et l'enroulement des éléments dans un vecteur sont :

- *vlimit*
- *vwrap*
- *vmirror*

Ces opérateurs sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel à la place d'un signal scalaire. Les résultats remplacent les anciennes valeurs du vecteur contenues dans une f-table si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes travaillent au taux-k.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Chemins de Retard Vectoriel au Taux de Contrôle

Chemins de Retard Vectoriel au Taux de Contrôle :

- *vdelayk*
- *vport*
- *vecdelay*

Générateurs de Signal Aléatoire Vectoriel

Ces opcodes génèrent des vecteurs de nombres aléatoires à stocker dans des tables. Ils génèrent une sorte de 'bruit vectoriel à bande limitée'. Tous ces opcodes fonctionnent au taux-k.

Générateurs de signal aléatoire vectoriel : *vrandh* et *vrandi*.

Des vecteurs d'automates cellulaires peuvent être générés au moyen de : *vcella*.

Système de Patch Zak

Les opcodes zak sont utilisés pour créer un système de patch aux taux-i, -k et -a. On peut se représenter le système zak comme un tableau global de variables. Ces opcodes sont utiles pour réaliser de manière flexible des branchements et des routages d'un instrument à l'autre. Le système est semblable à une matrice de branchement sur une console de mixage ou à une matrice de modulation sur un synthétiseur. Il est aussi utile lorque l'on a besoin d'un tableau de variables.

Le système zak est initialisé par l'opcode *zakinit* qui est habituellement placé juste après les autres initialisations globales : *sr*, *kr*, *ksmps*, *nchnls*. L'opcode *zakinit* définit deux plages de mémoire, une pour les patchs aux taux-i et -k, et l'autre pour les patchs au taux-a. L'opcode *zakinit* ne peut être appelé qu'une fois. Après l'initialisation de l'espace zak, on peut utiliser d'autres opcodes zak pour lire et écrire dans l'espace mémoire zak, ainsi qu'exécuter d'autres tâches.

Les opcodes zak sont comptés à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

Les opcodes pour le système de patch zak sont :

- Taux Audio : *zacl*, *zakinit*, *zamod*, *zar*, *zarg*, *zaw* et *zawm*.
- Taux de Contrôle : *zkcl*, *zkmod*, *zkr*, *zkw*, et *zkwm*.
- A l'initialisation : *zir*, *ziw* et *ziwm*

Accueil de Plugin

Csound accueille actuellement des plugins externes au moyen de *dssi4cs* (pour les plugins LADSPA) sur Linux et *vst4cs* (pour les plugins VST) sur Windows et Mac OS X.

DSSI et LADSPA pour Csound

dssi4cs permet l'utilisation des effets et des synthétiseurs plugin DSSI et LADSPA dans Csound sur Linux. Les opcodes suivants sont disponibles :

- *dssiinit* - Charge un plugin.
- *dssiactivate* - Active ou désactive un plugin si celui-ci le permet.
- *dssilist* - Liste tous les plugins disponibles trouvés dans les variables globales LADSPA_PATH et DSSI_PATH.
- *dssiaudio* - Traitement audio au moyen d'un Plugin.
- *dssictls* - Envoie une information de contrôle sur le port de contrôle d'un plugin.

Voir l'entrée pour *dssiinit* pour un exemple d'utilisation.

Note

Actuellement seuls les plugins LADSPA sont supportés, mais le support de DSSI est programmé.

VST pour Csound

vst4cs permet l'utilisation des effets et des synthétiseurs plugin VST dans Csound. Les opcodes suivants sont disponibles :

- *vstinit* - Charge un plugin.
- *vstaudio, vstaudiog* - Retourne la sortie d'un plugin.
- *vstmidiout* - Envoie des données MIDI à un plugin.
- *vstparamset, vstparamget* - Envoie et reçoit des données d'automatisation de et vers le plugin.
- *vstnote* - Envoie une note MIDI avec une durée définie.
- *vstinfo* - Sort les noms de Programme et de Paramètre pour un plugin.
- *vstbankload* - Charge une Banque `.fxb`
- *vstprogset* - Fixe un Programme dans une Banque `.fxb`
- *vstedit* - Ouvre l'éditeur de GUI pour le plugin, s'il est disponible.

Crédits

Par Andrés Cabrera et Michael Gogins

Utilise du code de VSTHost par Hermann Seib et de l'objet vst~ par Thomas Grill.

VST est une marque de Steinberg Media Technologies GmbH. VST Plug-In Technology par Steinberg.

OSC et Réseau

OSC

OSC permet l'interaction entre différents processus audio, et en particulier entre Csound et d'autres moteurs de synthèse. Les opcodes suivants sont disponibles :

- *OSCinit* - Démarre un thread d'écoute OSC.
- *OSClisten* - Réçoit les messages OSC.
- *OSCsend* - Envoie un message OSC.

Crédits

Par John ffitch avec le support et l'inspiration de la bibliothèque liblo.

Réseau

Les opcodes suivants peuvent envoyer ou recevoir des données audio en UDP :

- *sockrecv*
- *socksend*

Opcodes pour le Traitement à Distance

Les opcodes pour le Traitement à Distance permettent la transmission d'une partition ou d'évènements MIDI à travers un réseau, pour un traitement par des instances distantes (ou une autre instance locale). Les opcodes suivants sont disponibles :

- *insglobal* - Utilisé pour implémenter un orchestre distant.
- *insremot* - Utilisé pour implémenter un orchestre distant.
- *midiglobal* - Utilisé pour implémenter un orchestre MIDI distant.
- *midiremot* - Utilisé pour implémenter un orchestre MIDI distant.
- *remoteport* - Définit le port à utiliser avec le système distant.

Opcodes Mixer

La famille d'opcodes Mixer fournit un mélangeur global pour Csound. Les opcodes Mixer comprennent *MixerSend* pour envoyer (c'est-à-dire mélanger en entrée) un signal au taux-a depuis n'importe quel instrument vers un canal d'un bus de mixage, *MixerReceive* pour recevoir un signal de taux-a depuis un canal de n'importe quel bus de mixage dans un instrument, *MixerSetLevel* pour contrôler (au taux-k) le niveau du signal envoyé d'une source particulière vers un bus particulier, *MixerGetLevel* pour lire (au taux-k) le niveau d'envoi d'une source particulière à un bus particulier, et *MixerClear* pour réinitialiser les bus à zéro avant la k-période suivante d'une exécution.

Opcodes Python

Introduction

En utilisant la famille d'opcodes Python, vous pouvez interagir avec un interpréteur Python embarqué dans Csound de cinq manières :

1. Initialiser l'interpréteur Python (les opcodes *pyinit*),
2. Exécuter une instruction (les opcodes *pyrun*),
3. Exécuter un script (les opcodes *pyexec*),
4. Invoquer un objet appelable et lui passer des arguments (les opcodes *pycall*),
5. Evaluer une expression (les opcodes *pyeval*), ou
6. Changer la valeur d'un objet Python, avec la possibilité de créer un nouvel objet Python (les opcodes *pyassign*) ;

et vous pouvez faire toutes ces choses :

1. Au temps-i ou au temps-k,
2. Dans l'espace de nom global de Python, ou dans un espace de nom spécifique à une instance individuelle d'un instrument Csound (contexte local ou "l"),
3. Et vous pouvez récupérer de 0 à 8 valeurs de retour d'objets appelables qui acceptent N paramètres.

...cela signifie qu'il y a beaucoup d'opcodes concernant Python. Mais tous ces opcodes partagent le même préfixe *py*, et ils ont une structure de nom régulière :

```
"py" + [préfixe contextuel facultatif] + [nom d'action] + [suffixe de temps-x facultatif]
```

Syntaxe de l'Orchestre

Des blocs de code Python, voire des scripts entiers, peuvent être embarqués dans un orchestre Csound en utilisant les directives {{ et }} pour entourer le script, comme ci-dessous :

```
sr=44100
kr=4410
ksmps=10
nchnls=1
pyinit

giSinusoid ftgen 0, 0, 8192, 10, 1

pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
    if random.random() < p:
```

```
        i = int(random.random() * len(pool))
        pool[i] = n
    return random.choice(pool)
}}

instr 1
        k1 oscil 1, 3, giSinusoid
        k2 pycall1 "get_number_from_pool", k1 + 2, p4
                printk 0.01, k2
endin
```

Crédits

Copyright © 2002 par Maurizio Umberto Puxeddu. Tous droits réservés.

Copyright © 2004 et 2005 par Michael Gogins, pour certaines parties.

Opcodes pour le traitement d'image

Voici une liste des opcodes pour lire/écrire des fichiers d'image :

- *imagecreate*
- *imagesize*
- *imagegetpixel*
- *imagesetpixel*
- *imagesave*
- *imageload*
- *imagefree*

Opcodes divers

Voici une liste d'opcodes qui ne rentrent dans aucune catégorie :

- *system* - Appelle un programme externe via le mécanisme d'appel du système.

Partie III. Référence

Table des matières

Opcodes et Opérateurs de l'Orchestre ... 197
- != ... 198
- #define ... 200
- #include ... 204
- #undef ... 206
- #ifdef ... 207
- #ifndef ... 209
- $NOM ... 210
- % ... 213
- && ... 215
- > ... 216
- >= ... 218
- < ... 220
- <= ... 222
- * ... 224
- + ... 226
- - ... 228
- / ... 230
- = ... 232
- == ... 234
- ^ ... 236
- || ... 238
- 0dbfs ... 239
- << ... 242
- >> ... 244
- & ... 245
- | ... 246
- ¬ ... 247
- # ... 248
- a ... 249
- abetarand ... 251
- abexprnd ... 252
- abs ... 253
- acauchy ... 255
- active ... 256
- adsr ... 260
- adsyn ... 263
- adsynt ... 265
- adsynt2 ... 268
- aexprand ... 270
- aftouch ... 271
- agauss ... 273
- agogobel ... 274
- alinrand ... 275
- alpass ... 276
- ampdb ... 279
- ampdbfs ... 281
- ampmidi ... 283
- apcauchy ... 285
- apoisson ... 286
- apow ... 287
- areson ... 288
- aresonk ... 290

atone 291
atonek 293
atonex 294
atrirand 295
ATSadd 296
ATSaddnz 298
ATSbufread 300
ATScross 302
ATSinfo 304
ATSinterpread 306
ATSread 307
ATSreadnz 309
ATSpartialtap 311
ATSsinnoi 312
aunirand 314
aweibull 315
babo 316
balance 320
bamboo 322
barmodel 324
bbcutm 326
bbcuts 331
betarand 333
bexprnd 335
bformenc 337
bformenc1 339
bformdec 341
bformdec1 343
binit 345
biquad 346
biquada 350
birnd 351
bqrez 353
butbp 355
butbr 356
buthp 357
butlp 358
butterbp 359
butterbr 361
butterhp 363
butterlp 365
button 367
buzz 368
cabasa 370
cauchy 372
ceil 374
cent 375
cggoto 377
chanctrl 379
changed 380
chani 382
chano 383
chebyshevpoly 384
checkbox 387
chn 389
chnclear 391
chnexport 392
chnget 394

chnmix ... 396
chnparams ... 397
chnset ... 398
chuap ... 400
cigoto ... 403
ckgoto ... 405
clear ... 407
clfilt ... 408
clip ... 411
clock ... 414
clockoff ... 415
clockon ... 416
cngoto ... 417
comb ... 419
compress ... 422
control ... 424
convle ... 425
convolve ... 426
cos ... 429
cosh ... 431
cosinv ... 433
cps2pch ... 435
cpsmidi ... 439
cpsmidib ... 441
cpsmidinn ... 443
cpsoct ... 446
cpspch ... 449
cpstmid ... 452
cpstun ... 455
cpstuni ... 458
cpsxpch ... 461
cpuprc ... 465
cross2 ... 467
crunch ... 469
ctrl14 ... 471
ctrl21 ... 473
ctrl7 ... 475
ctrlinit ... 477
cuserrnd ... 478
dam ... 480
date ... 483
dates ... 485
db ... 487
dbamp ... 489
dbfsamp ... 491
dcblock ... 493
dcblock2 ... 495
dconv ... 496
delay ... 498
delay1 ... 500
delayk ... 501
delayr ... 503
delayw ... 504
deltap ... 506
deltap3 ... 508
deltapi ... 510
deltapn ... 512
deltapx ... 514

deltapxw 516
denorm 518
diff 519
diskgrain 521
diskin 524
diskin2 527
dispfft 530
display 532
distort 534
distort1 536
divz 538
downsamp 540
dripwater 542
dssiactivate 544
dssiaudio 545
dssictls 546
dssiinit 547
dssilist 549
dumpk 550
dumpk2 552
dumpk3 554
dumpk4 556
duserrnd 558
else 560
elseif 561
endif 562
endin 563
endop 565
envlpx 566
envlpxr 569
ephasor 571
eqfil 572
event 574
event_i 577
exitnow 578
exp 579
expcurve 581
expon 583
exprand 585
expseg 587
expsega 589
expsegr 591
ficlose 594
filelen 596
filenchnls 598
filepeak 600
filesr 602
filter2 604
fin 606
fini 607
fink 609
fiopen 610
flanger 612
flashtxt 614
FLbox 616
FLbutBank 621
FLbutton 624
FLcloseButton 629

FLcolor ... 632
FLcolor2 ... 634
FLcount ... 635
FLexecButton ... 638
FLgetsnap ... 641
FLgroup ... 642
FLgroupEnd ... 644
FLgroupEnd ... 645
FLhide ... 646
FLhvsBox ... 647
FLhvsBoxSetValue ... 648
FLjoy ... 649
FLkeyIn ... 652
FLknob ... 654
FLlabel ... 659
FLloadsnap ... 661
FLmouse ... 662
flooper ... 664
flooper2 ... 666
floor ... 668
FLpack ... 669
FLpackEnd ... 672
FLpack_end ... 673
FLpanel ... 674
FLpanelEnd ... 677
FLpanel_end ... 678
FLprintk ... 679
FLprintk2 ... 680
FLroller ... 681
FLrun ... 684
FLsavesnap ... 685
FLscroll ... 690
FLscrollEnd ... 693
FLscroll_end ... 694
FLsetAlign ... 695
FLsetBox ... 696
FLsetColor ... 698
FLsetColor2 ... 700
FLsetFont ... 701
FLsetPosition ... 703
FLsetSize ... 704
FLsetsnap ... 705
FLsetSnapGroup ... 707
FLsetText ... 708
FLsetTextColor ... 710
FLsetTextSize ... 711
FLsetTextType ... 712
FLsetVal_i ... 715
FLsetVal ... 716
FLshow ... 717
FLslidBnk ... 718
FLslidBnk2 ... 722
FLslidBnkGetHandle ... 725
FLslidBnkSet ... 726
FLslidBnkSetk ... 727
FLslidBnk2Set ... 729
FLslidBnk2Setk ... 730
FLslider ... 733

FLtabs 739
FLtabsEnd 744
FLtabs_end 745
FLtext 746
FLupdate 749
fluidAllOut 750
fluidCCi 753
fluidCCk 754
fluidControl 755
fluidEngine 757
fluidLoad 761
fluidNote 763
fluidOut 765
fluidProgramSelect 767
fluidSetInterpMethod 769
FLvalue 770
FLvkeybd 772
FLvslidBnk 773
FLvslidBnk2 777
FLxyin 779
fmb3 782
fmbell 785
fmmetal 788
fmpercfl 791
fmrhode 794
fmvoice 797
fmwurlie 799
fof 802
fof2 805
fofilter 811
fog 813
fold 815
follow 817
follow2 819
foscil 821
foscili 823
fout 825
fouti 830
foutir 832
foutk 834
fprintks 836
fprints 842
frac 844
freeverb 846
ftchnls 848
ftconv 850
ftfree 853
ftgen 854
ftgentmp 857
ftlen 858
ftload 860
ftloadk 861
ftlptim 862
ftmorf 864
ftsave 866
ftsavek 868
ftsr 869
gain 871

gainslider 872
gauss 874
gbuzz 876
getcfg 879
gogobel 880
goto 882
grain 884
grain2 886
grain3 890
granule 895
guiro 899
harmon 901
harmon2 904
hilbert 906
hrtfer 910
hrtfmove 912
hrtfmove2 915
hrtfstat 918
hsboscil 921
hvs1 924
hvs2 928
hvs3 934
i 937
ibetarand 938
ibexprnd 939
icauchy 940
ictrl14 941
ictrl21 942
ictrl7 943
iexprand 944
if 945
igauss 949
igoto 950
ihold 952
ilinrand 954
imagecreate 955
imagefree 957
imagegetpixel 959
imageload 961
imagesave 963
imagesetpixel 965
imagesize 967
imidic14 969
imidic21 970
imidic7 971
in 972
in32 973
inch 974
inh 975
init 976
initc14 977
initc21 978
initc7 979
ino 980
inq 981
inrg 982
ins 983
insremot 984

insglobal 986
instimek 987
instimes 988
instr 989
int 992
integ 994
interp 996
invalue 999
inx 1000
inz 1001
ioff 1002
ion 1003
iondur 1004
iondur2 1005
ioutat 1006
ioutc 1007
ioutc14 1008
ioutpat 1009
ioutpb 1010
ioutpc 1011
ipcauchy 1012
ipoisson 1013
ipow 1014
is16b14 1015
is32b14 1016
islider16 1017
islider32 1018
islider64 1019
islider8 1020
itablecopy 1021
itablegpw 1022
itablemix 1023
itablew 1024
itrirand 1025
iunirand 1026
iweibull 1027
jacktransport 1028
jitter 1030
jitter2 1032
jspline 1034
k 1035
kbetarand 1036
kbexprnd 1037
kcauchy 1038
kdump 1039
kdump2 1040
kdump3 1041
kdump4 1042
kexprand 1043
kfilter2 1044
kgauss 1045
kgoto 1046
klinrand 1048
kon 1049
koutat 1050
koutc 1051
koutc14 1052
koutpat 1053

koutpb 1054
koutpc 1055
kpcauchy 1056
kpoisson 1057
kpow 1058
kr 1059
kread 1060
kread2 1061
kread3 1062
kread4 1063
ksmps 1064
ktableseg 1065
ktrirand 1066
kunirand 1067
kweibull 1068
lfo 1069
limit 1071
line 1072
linen 1074
linenr 1076
lineto 1077
linrand 1078
linseg 1080
linsegr 1083
locsend 1086
locsig 1088
log 1091
log10 1093
logbtwo 1095
logcurve 1097
loop_ge 1099
loop_gt 1100
loop_le 1101
loop_lt 1102
loopseg 1103
loopsegp 1105
lorenz 1107
lorisread 1110
lorismorph 1112
lorisplay 1113
loscil 1114
loscil3 1117
loscilx 1120
lowpass2 1121
lowres 1123
lowresx 1125
lpf18 1127
lpfreson 1129
lphasor 1130
lpinterp 1132
lposcil 1133
lposcil3 1134
lposcila 1135
lposcilsa 1136
lposcilsa2 1137
lpread 1138
lpreson 1140
lpshold 1141

lpsholdp 1143
lpslot 1144
mac 1146
maca 1147
madsr 1148
mandel 1151
mandol 1152
marimba 1154
massign 1157
max 1159
maxabs 1160
maxabsaccum 1161
maxaccum 1162
maxalloc 1163
max_k 1165
mclock 1166
mdelay 1167
metro 1169
midic14 1171
midic21 1173
midic7 1175
midichannelaftertouch 1177
midichn 1179
midicontrolchange 1182
midictrl 1184
mididefault 1185
midiin 1186
midinoteoff 1189
midinoteoncps 1191
midinoteonkey 1193
midinoteonoct 1195
midinoteonpch 1197
midion 1199
midion2 1202
midiout 1203
midipitchbend 1205
midipolyaftertouch 1207
midiprogramchange 1209
miditempo 1210
midremot 1211
midglobal 1214
min 1215
minabs 1216
minabsaccum 1217
minaccum 1218
mirror 1219
MixerSetLevel 1220
MixerGetLevel 1222
MixerSend 1223
MixerReceive 1224
MixerClear 1226
mode 1227
monitor 1230
moog 1231
moogladder 1233
moogvcf 1235
moogvcf2 1237
moscil 1239

mpulse 1241
mrtmsg 1243
multitap 1244
mute 1245
mxadsr 1247
nchnls 1249
nestedap 1250
nlfilt 1253
noise 1255
noteoff 1258
noteon 1259
noteondur 1260
noteondur2 1262
notnum 1264
nreverb 1266
nrpn 1269
nsamp 1270
nstrnum 1272
ntrpol 1273
octave 1274
octcps 1276
octmidi 1279
octmidib 1281
octmidinn 1283
octpch 1286
opcode 1289
OSCsend 1294
OSCinit 1296
OSClisten 1297
oscbnk 1301
oscil 1306
oscil1 1308
oscil1i 1309
oscil3 1310
oscili 1312
oscilikt 1314
osciliktp 1316
oscilikts 1318
osciln 1320
oscils 1321
oscilx 1323
out 1324
out32 1325
outc 1326
outch 1327
outh 1328
outiat 1329
outic 1330
outic14 1331
outipat 1333
outipb 1334
outipc 1335
outkat 1336
outkc 1337
outkc14 1338
outkpat 1339
outkpb 1340
outkpc 1341

outo 1344
outq 1345
outq1 1346
outq2 1347
outq3 1348
outq4 1349
outrg 1350
outs 1351
outs1 1352
outs2 1353
outvalue 1354
outx 1355
outz 1356
p 1357
pan 1359
pan2 1361
pareq 1362
partials 1365
partikkel 1367
partikkelsync 1376
pcauchy 1377
pchbend 1379
pchmidi 1381
pchmidib 1383
pchmidinn 1385
pchoct 1388
pconvolve 1391
pcount 1394
pdclip 1396
pdhalf 1399
pdhalfy 1402
peak 1405
peakk 1407
pgmassign 1408
phaser1 1412
phaser2 1415
phasor 1419
phasorbnk 1421
pindex 1423
pinkish 1425
pitch 1428
pitchamdf 1431
planet 1434
pluck 1436
poisson 1438
polyaft 1442
polynomial 1444
pop 1446
pop_f 1447
port 1448
portk 1449
poscil 1451
poscil3 1453
pow 1456
powershape 1458
powoftwo 1460
prealloc 1462
prepiano 1464

print 1467
printf 1469
printk 1470
printk2 1472
printks 1474
prints 1477
product 1479
pset 1480
ptrack 1481
puts 1483
push 1484
push_f 1485
pvadd 1486
pvbufread 1489
pvcross 1491
pvinterp 1493
pvoc 1495
pvread 1497
pvsadsyn 1499
pvsanal 1501
pvsarp 1504
pvsbandp 1506
pvsbandr 1508
pvsbin 1510
pvsblur 1512
pvsbuffer 1514
pvsbufread 1516
pvscale 1518
pvscent 1520
pvscross 1521
pvsdemix 1522
pvsdiskin 1524
pvsdisp 1525
pvsfilter 1527
pvsfread 1529
pvsfreeze 1530
pvsftr 1532
pvsftw 1534
pvsfwrite 1536
pvshift 1538
pvsifd 1540
pvsinfo 1542
pvsinit 1543
pvsin 1544
pvsmaska 1545
pvsmix 1547
pvsmorph 1548
pvsmooth 1549
pvsout 1551
pvsosc 1552
pvspitch 1555
pvstencil 1558
pvsvoc 1560
pvsynth 1562
pyassign Opcodes 1564
pycall Opcodes 1565
pyeval Opcodes 1568
pyexec Opcodes 1569

pyinit Opcodes 1572
pyrun Opcodes 1573
rand 1575
randh 1577
randi 1579
random 1581
randomh 1583
randomi 1585
rbjeq 1587
readclock 1590
readk 1592
readk2 1594
readk3 1596
readk4 1598
reinit 1600
release 1602
remoteport 1603
remove 1604
repluck 1605
reson 1607
resonk 1609
resonr 1610
resonx 1613
resonxk 1615
resony 1616
resonz 1618
resyn 1620
reverb 1622
reverb2 1624
reverbsc 1625
rewindscore 1627
rezzy 1628
rigoto 1630
rireturn 1631
rms 1633
rnd 1635
rnd31 1637
round 1642
rspline 1643
rtclock 1644
s16b14 1646
s32b14 1648
scale 1650
samphold 1652
sandpaper 1653
scanhammer 1655
scans 1656
scantable 1658
scanu 1660
scoreline 1662
scoreline_i 1664
schedkwhen 1665
schedkwhennamed 1668
schedule 1670
schedwhen 1672
seed 1675
sekere 1676
semitone 1678

sense 1680
sensekey 1681
seqtime 1685
seqtime2 1688
setctrl 1690
setksmps 1692
setscorepos 1694
sfilist 1695
sfinstr 1696
sfinstr3 1698
sfinstr3m 1700
sfinstrm 1702
sfload 1704
sflooper 1705
sfpassign 1707
sfplay 1708
sfplay3 1710
sfplay3m 1712
sfplaym 1714
sfplist 1716
sfpreset 1717
shaker 1719
sin 1721
sinh 1723
sininv 1725
sinsyn 1727
sleighbells 1729
slider16 1731
slider16f 1733
slider32 1735
slider32f 1737
slider64 1739
slider64f 1741
slider8 1743
slider8f 1745
slider16table 1747
slider16tablef 1749
slider32table 1751
slider32tablef 1753
slider64table 1755
slider64tablef 1757
slider8table 1759
slider8tablef 1761
sliderKawai 1763
sndload 1764
sndloop 1766
sndwarp 1768
sndwarpst 1772
socksend 1775
sockrecv 1777
soundin 1779
soundout 1782
soundouts 1784
space 1786
spat3d 1790
spat3di 1798
spat3dt 1802
spdist 1806

specaddm 1810
specdiff 1811
specdisp 1812
specfilt 1813
spechist 1814
specptrk 1815
specscal 1817
specsum 1818
spectrum 1819
splitrig 1821
spsend 1823
sprintf 1826
sprintfk 1827
sqrt 1829
sr 1831
stack 1832
statevar 1833
stix 1835
strchar 1837
strchark 1838
strcpy 1839
strcpyk 1840
strcat 1841
strcatk 1842
strcmp 1843
strcmpk 1844
streson 1845
strget 1847
strindex 1848
strindexk 1849
strlen 1850
strlenk 1851
strlower 1852
strlowerk 1853
strrindex 1854
strrindexk 1855
strset 1856
strsub 1858
strsubk 1860
strtod 1861
strtodk 1862
strtol 1863
strtolk 1864
strupper 1865
strupperk 1866
subinstr 1867
subinstrinit 1870
sum 1871
svfilter 1872
syncgrain 1875
syncloop 1877
syncphasor 1879
system 1883
tb 1885
tab 1888
tabrec 1889
table 1890
table3 1892

tablecopy 1893
tablegpw 1894
tablei 1895
tableicopy 1896
tableigpw 1897
tableikt 1898
tableimix 1900
tableiw 1902
tablekt 1904
tablemix 1906
tableng 1908
tablera 1910
tableseg 1913
tablew 1914
tablewa 1917
tablewkt 1920
tablexkt 1923
tablexseg 1926
tabmorph 1927
tabmorpha 1929
tabmorphak 1931
tabmorphi 1933
tabplay 1935
tambourine 1936
tan 1938
tanh 1940
taninv 1942
taninv2 1944
tbvcf 1946
tempest 1949
tempo 1952
tempoval 1954
tigoto 1956
timedseq 1957
timeinstk 1959
timeinsts 1961
timek 1963
times 1965
timout 1967
tival 1968
tlineto 1969
tone 1970
tonek 1971
tonex 1972
trandom 1973
tradsyn 1974
transeg 1976
trcross 1978
trfilter 1980
trhighest 1982
trigger 1983
trigseq 1985
trirand 1987
trlowest 1989
trmix 1990
trscale 1991
trshift 1992
trsplit 1993

turnoff 1995
turnoff2 1997
turnon 1998
unirand 1999
upsamp 2001
urd 2002
vadd 2003
vadd_i 2006
vaddv 2008
vaddv_i 2011
vaget 2013
valpass 2015
vaset 2016
vbap16 2018
vbap16move 2020
vbap4 2022
vbap4move 2024
vbap8 2026
vbap8move 2028
vbaplsinit 2031
vbapz 2033
vbapzmove 2035
vcella 2037
vco 2040
vco2 2043
vco2ft 2047
vco2ift 2049
vco2init 2051
vcomb 2054
vcopy 2057
vcopy_i 2060
vdelay 2062
vdelay3 2064
vdelayx 2066
vdelayxq 2068
vdelayxs 2070
vdelayxw 2072
vdelayxwq 2074
vdelayxws 2076
vdivv 2078
vdivv_i 2081
vdelayk 2083
vecdelay 2084
veloc 2085
vexp 2087
vexp_i 2090
vexpseg 2092
vexpv 2094
vexpv_i 2097
vibes 2099
vibr 2101
vibrato 2103
vincr 2106
vlimit 2107
vlinseg 2108
vlowres 2110
vmap 2112
vmirror 2114

vmult 2115
vmult_i 2119
vmultv 2121
vmultv_i 2124
voice 2126
vosim 2129
vphaseseg 2134
vport 2136
vpow 2137
vpow_i 2140
vpowv 2142
vpowv_i 2145
vpvoc 2147
vrandh 2149
vrandi 2152
vstaudio, vstaudiog 2155
vstbankload 2156
vstedit 2157
vstinit 2158
vstinfo 2159
vstmidiout 2160
vstnote 2162
vstparamset,vstparamget 2164
vstprogset 2166
vsubv 2167
vsubv_i 2170
vtable1k 2172
vtablei 2174
vtablek 2176
vtablea 2178
vtablewi 2180
vtablewk 2181
vtablewa 2183
vtabi 2185
vtabk 2187
vtaba 2189
vtabwi 2191
vtabwk 2192
vtabwa 2193
vwrap 2194
waveset 2195
weibull 2197
wgbow 2199
wgbowedbar 2201
wgbrass 2203
wgclar 2205
wgflute 2207
wgpluck 2209
wgpluck2 2212
wguide1 2214
wguide2 2216
wrap 2219
wterrain 2220
xadsr 2222
xin 2224
xout 2226
xscanmap 2228
xscansmap 2229

xscans 2230
xscanu 2232
xtratim 2235
xyin 2238
zacl 2240
zakinit 2242
zamod 2245
zar 2247
zarg 2249
zaw 2251
zawm 2253
zfilter2 2256
zir 2258
ziw 2260
ziwm 2262
zkcl 2264
zkmod 2266
zkr 2268
zkw 2270
zkwm 2272
Instructions de Partition et Routines GEN 2275
Instructions de Partition 2275
Instruction a (ou Instruction Avancer) 2276
Instruction b 2277
Instruction e 2278
Instruction f (ou Instruction de Table de Fonction) 2279
Instruction i (Instruction d'Instrument ou de Note) 2282
Instruction m (Instruction de Marquage) 2286
Instruction n 2287
Instruction q 2288
Instruction r (Instruction Répéter) 2289
Instruction s 2291
Instruction t (Instruction de Tempo) 2292
Instruction v 2293
Instruction x 2295
{ Statement 2296
} Statement 2299
Routines GEN 2299
GEN01 2303
GEN02 2306
GEN03 2308
GEN04 2310
GEN05 2312
GEN06 2314
GEN07 2316
GEN08 2318
GEN09 2320
GEN10 2323
GEN11 2325
GEN12 2327
GEN13 2329
GEN14 2332
GEN15 2335
GEN16 2336
GEN17 2339
GEN18 2340
GEN19 2341
GEN20 2343

GEN21 2345
GEN22 2347
GEN23 2348
GEN24 2349
GEN25 2350
GEN27 2351
GEN28 2352
GEN30 2354
GEN31 2355
GEN32 2356
GEN33 2358
GEN34 2360
GEN40 2362
GEN41 2363
GEN42 2364
GEN43 2365
GEN51 2366
GEN52 2368
Les Programmes Utilitaires 2369
Répertoires. 2369
Formats des Fichiers Son. 2369
Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL) 2370
Requêtes sur un Fichier (SNDINFO) 2381
Conversion de Fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EXPORT, PV_IMPORT, SDIF2AD, SRCONV) 2383
Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MAKECSD, MIXER, SCALE) 2398
Cscore 2411
Evénements, Listes et Opérations 2411
Ecrire un Programme de Contrôle Cscore 2414
Compiler un Programme Cscore 2419
Exemples Plus Avancés 2422
Etendre Csound 2424
Ajouter des Générateurs Unitaires 2424
Créer un Générateur Unitaire Intégré 2424
Ajouter un Générateur Unitaire comme Plugin 2428
Référence de OENTRY 2428

Opcodes et Opérateurs de l'Orchestre

!=

!= — Détermine si une valeur n'est pas égale à l'autre.

Description

Détermine si une valeur n'est pas égale à l'autre.

Syntaxe

```
(a != b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* n'est pas égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *//*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur !=. Il utilise le fichier *notequal.csd* [examples/notequal.csd].

Exemple 7. Exemple de l'opérateur !=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o notequal.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it not equal to 3? (1 = true, 0 = false)
  k2 = (p4 != 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 1.000000
```

Voir Aussi

==, >=, >, <=, <

Crédits

Exemple écrit par Kevin Conder.

#define

#define — Définit une macro.

Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et $ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

#define NOM -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. La casse est significative. Cette forme est limitée dans le sens que les noms de variable sont fixes. On peut obtenir plus de flexibilité en utilisant une macro avec arguments, décrite ci-dessous.

#define NOM(*a' b' c'*) -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont appelés sous la forme : $A. En fait, l'implémentation définit les arguments comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et les noms sont une combinaison quelconque de lettres. Souvenez-vous que la casse est significative dans les noms de macro.

Syntax

```
#define NOM # texte de substitution #
```

```
#define NOM(a' b' c') # texte de substitution #
```

Initialisation

 # texte de substitution # -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

Exemples

Voici un exemple simple de définition de macro. Il utilise le fichier *define.csd* [examples/define.csd].

Exemple 8. Exemple simple de définition de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
  ; Use the macros.
  ; This will be expanded to "a1 oscil 5000, 440, 1".
  a1 oscil $VOLUME, $FREQ, $TABLE

  ; Send it to the output.
  out a1
endin


</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celles-ci :

```
Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE
```

Voici un exemple simple de définition de macro avec arguments. Il utilise le fichier *define_args.csd* [examples/define_args.csd].

Exemple 9. Exemple simple de définition de macro avec arguments.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define_args.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
  ; Use the oscillator macro.
  ; This will be expanded to "a1 oscil 5000, 440, 1".
  a1 $OSCMACRO(5000'440'1)

  ; Send it to the output.
  out a1
endin


</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celle-ci :

```
Macro definition for OSCMACRO
```

Macros Prédéfinies de Constantes Mathématiques

A partir de Csound 5.04 des Macros de Constantes Mathématiques sont prédéfinies. Les valeurs définies sont celles que l'on trouve dans le fichier d'en-tête C math.h, et elles sont automatiquement définies au démarrage de Csound et disponibles pour une utilisation dans les orchestres.

Macro	Valeur
$M_E	2.7182818284590452354
$M_LOG2E	1.4426950408889634074
$M_LOG10E	0.43429448190325182765
$M_LN2	0.69314718055994530942
$M_LN10	2.30258509299404568402
$M_PI	3.14159265358979323846
$M_PI_2	1.57079632679489661923
$M_PI_4	0.78539816339744830962
$M_1_PI	0.31830988618379067154
$M_2_PI	0.63661977236758134308
$M_2_SQRTPI	1.12837916709551257390
$M_SQRT2	1.41421356237309504880
$M_SQRT1_2	0.70710678118654752440

Voir Aussi

$NOM, *#undef*

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound

#include

#include — Inclut un fichier externe pour traitement.

Description

Inclut un fichier externe pour traitement.

Syntaxe

```
#include "nomfichier"
```

Exécution

Il est parfois commode d'organiser un orchestre sur plusieurs fichiers, par exemple avec chaque instrument dans un fichier séparé. Ce style est supporté par la fonctionnalité *#include* qui fait partie du système de macros. Une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère " peut être remplacé par n'importe quel caractère approprié. Pour la plupart des utilisations le symbole de l'apostrophe double sera probablement le plus commode. Le nom de fichier peut inclure un chemin complet.

L'entrée est prise à partir du fichier nommé jusqu'à son terme, puis revient à la source précédente. *Note :* les versions de Csound antérieures à la 4.19 limitaient à 20 la profondeur des fichiers inclus et des macros.

Il est également suggéré d'utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur.

A la limite, on pourrait définir chaque instrument comme une macro, avec un numéro d'instrument en paramètre. On pourrait alors construire un orchestre entier à partir d'un certain nombre d'instructions *#include* suivies par des appels de macro.

```
#include "clarinet"
#include "flute"
#include "bassoon"
$CLARINET(1)
$FLUTE(2)
$BASSOON(3)
```

Il faut insister sur le fait que ces changements ont lieu au niveau littéral et n'ont donc aucune incidence sémantique.

Exemples

Voici un exemple de l'opcode include. Il utilise les fichiers *include.csd* [examples/include.csd], et *table1.inc* [examples/table1.inc].

Exemple 10. Exemple de l'opcode include.

```
/* table1.inc */
; Table #1, a sine wave.
f 1 0 16384 10 1
/* table1.inc */
```

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o include.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Include the file for Table #1.
#include "table1.inc"

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

#undef

#undef — Annule la définition d'une macro.

Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et $ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

#undef NOM -- annule la définition d'un nom de macro. Si une macro n'est plus nécessaire, on peut annuler sa définition avec *#undef NOM*.

Syntaxe

```
#undef NOM
```

Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

Voir Aussi

#define, *$NOM*

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Nouveau dans la version 3.48 de Csound

#ifdef

#ifdef — Lecture de code conditionnelle.

Description

Si une macro est définie alors *#ifdef* peut incorporer du texte dans un orchestre jusqu'au prochain *#end*. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

Syntaxe

```
#ifdef NOM

....

#else

....

#end
```

Exécution

Noter que l'on peut imbriquer les *#ifdef*, comme dans le langage du préprocesseur C.

Exemples

Voici un exemple simple de cette insertion conditionnelle.

Exemple 11. Exemple simple de la forme #ifdef.

```
#define debug
     instr 1
#ifdef debug
     print "calling oscil"
#end
     a1   oscil 32000,440,1
     out  a1
     endin
```

Voir Aussi

$NOM, *#define*, *#ifndef*.

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.

Bath, UK
Avril 2005

Nouveau dans Csound5 (et 4.23f13)

#ifndef

#ifndef — Lecture de code conditionnelle.

Description

Si la macro spécifiée n'est pas définie alors *#ifndef* peut incorporer du texte dans un orchestre jusqu'au prochain *#end*. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

Syntaxe

```
#ifndef NOM

....

#else

....

#end
```

Exécution

Noter que l'on peut imbriquer les *#ifndef*, comme dans le langage du préprocesseur C.

Voir Aussi

$NOM, *#define*, *#ifdef*.

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 2005

Nouveau dans Csound5 (et 4.23f13)

$NOM

$NOM — Appelle une macro définie.

Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et $ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

$NOM -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé du caractère $. La fin du nom est marquée par le premier caractère qui n'est ni une lettre ni un chiffre. S'il est nécessaire que ce caractère ne soit pas un espace, on peut utiliser un point, qui sera ignoré, pour terminer le nom. La chaîne, *$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut lui-même comprendre des appels de macro.

Syntaxe

```
$NOM
```

Initialisation

 # texte de substitution # -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

Exemples

Voici un exemple d'appel de macro. Il utilise le fichier *define.csd* [examples/define.csd].

Exemple 12. Un exemple d'appel de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
  ; Use the macros.
  ; This will be expanded to "a1 oscil 5000, 440, 1".
  a1 oscil $VOLUME, $FREQ, $TABLE

  ; Send it to the output.
  out a1
endin


</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celles-ci :

```
Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE
```

Voici un exemple d'appel de macro avec arguments. Il utilise le fichier *define_args.csd* [examples/define_args.csd].

Exemple 13. Un exemple d'appel de macro avec arguments.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define_args.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
  ; Use the oscillator macro.
  ; This will be expanded to "a1 oscil 5000, 440, 1".
```

```
  a1 $OSCMACRO(5000'440'1)

  ; Send it to the output.
  out a1
endin


</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celle-ci :

```
Macro definition for OSCMACRO
```

Voir Aussi

#define, *#undef*

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound

%

% — Opérateur modulo.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

a + b * c.

Trois règles s'appliquent dans de tels cas :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus s'interprète comme

a + (b * c)

avec * prenant b et c puis + prenant a et b * c.

2. + et # sont prioritaires sur &&, qui devance lui-même || :

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses pour forcer un groupement particulier.

L'opérateur *%* retourne la valeur de la réduction de *a* par *b*, de telle façon que le résultat, en valeur absolue, est inférieur à la valeur absolue de *b*, par soustraction répétée. C'est l'équivalent de la fonction modulo pour les entiers. Nouveau dans la version 3.50 de Csound.

Syntaxe

```
a % b  (pas de restriction de taux)
```

où les arguments *a* et *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur %. Il utilise le fichier *modulus.csd* [examples/modulus.csd].

Exemple 14. Exemple de l'opérateur %.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o modulus.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 5 % 3
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera une ligne comme celle-ci :

```
instr 1:  i1 = 2.000
```

Voir Aussi

-, +, &&, ||, *, /, ^

Crédits

Exemple écrit par Kevin Conder.

&&

&& — Opérateur ET logique.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

a + b * c.

Trois règles s'appliquent dans de tels cas :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus s'interprète comme

a + (b * c)

avec * prenant b et c puis + prenant a et b * c.

2. + et # sont prioritaires sur &&, qui devance lui-même || :

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses pour forcer un groupement particulier.

Syntaxe

```
a && b  (ET logique ; pas de taux audio)
```

où les arguments *a* et *b* peuvent être des expressions.

Voir Aussi

-, +, //, *, /, ^, %

>

> — Détermine si une valeur est supérieure à l'autre.

Description

Détermine si une valeur est supérieure à l'autre.

Syntaxe

```
(a >  b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* supérieur à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *||*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur >. Il utilise le fichier *greaterthan.csd* [examples/greaterthan.csd].

Exemple 15. Exemple de l'opérateur >.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterthan.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it greater than 3? (1 = true, 0 = false)
  k2 = (p4 > 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 1.000000
```

Voir Aussi

==, >=, <=, <, !=

Crédits

Exemple écrit par Kevin Conder.

>=

>= — Détermine si une valeur est supérieure ou égale à l'autre.

Description

Détermine si une valeur est supérieure ou égale à l'autre.

Syntaxe

```
(a >= b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* supérieur ou égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *||*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur >=. Il utilise le fichier *greaterequal.csd* [examples/greaterequal.csd].

Exemple 16. Exemple de l'opérateur >=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterequal.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it greater than or equal to 3? (1 = true, 0 = false)
  k2 = (p4 >= 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 1.000000
```

Voir Aussi

==, >, <=, <, !=

Crédits

Exemple écrit par Kevin Conder.

<

< — Détermine si une valeur est inférieure à l'autre.

Description

Détermine si une valeur est inférieure à l'autre.

Syntaxe

```
(a <  b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* inférieur à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *||*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur <. Il utilise le fichier *lessthan.csd* [examples/lessthan.csd].

Exemple 17. Exemple de l'opérateur <.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lessthan.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it less than 3? (1 = true, 0 = false)
  k2 = (p4 < 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 0.000000
```

Voir Aussi

==, >=, >, <=, !=

Crédits

Exemple écrit par Kevin Conder.

<=

<= — Détermine si une valeur est inférieure ou égale à l'autre.

Description

Détermine si une valeur est inférieure ou égale à l'autre.

Syntaxe

```
(a <= b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* inférieur ou égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *||*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur <=. Il utilise le fichier *lessequal.csd* [examples/lessequal.csd].

Exemple 18. Exemple de l'opérateur <=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lessequal.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it less than or equal to 3? (1 = true, 0 = false)
  k2 = (p4 <= 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 0.000000
```

Voir Aussi

==, >=, >, <, !=

Crédits

Exemple écrit par Kevin Conder.

*

* — Opérateur de multiplication

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

a + b * c.

Dans de tels cas trois règles s'appliquent :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus est interprétée comme

a + (b * c)

avec * s'appliquant à b et à c et ensuite + s'appliquant à a et à b * c.

2. + et # sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

Syntaxe

```
a * b  (pas de restriction de taux)
```

où les arguments *a* et *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur *. Il utilise le fichier *multiplies.csd* [examples/multiplies.csd].

Exemple 19. Exemple de l'opérateur *.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o multiplies.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 * 8
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 192.000
```

Voir Aussi

-, +, *&&*, *||*, */*, *^*, *%*

Crédits

Exemple écrit par Kevin Conder.

+

+ — Opérateur d'addition

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

a + b * c.

Dans de tels cas trois règles s'appliquent :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus est interprétée comme

a + (b * c)

avec * s'appliquant à b et à c et ensuite + s'appliquant à a et à b * c.

2. + et # sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

Syntaxe

```
+a  (pas de restriction de taux)

a + b  (pas de restriction de taux)
```

où les arguments *a* et *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur +. Il utilise le fichier *adds.csd* [examples/adds.csd].

Exemple 20. Exemple de l'opérateur +.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adds.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 + 8
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 32.000
```

Voir Aussi

-, &&, ||, *, /, ^, %

Crédits

Exemple écrit par Kevin Conder.

-

- — Opérateur de soustraction.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

a + b * c.

Dans de tels cas trois règles s'appliquent :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus est interprétée comme

a + (b * c)

avec * s'appliquant à b et à c et ensuite + s'appliquant à a et à b * c.

2. + et # sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

Syntaxe

```
#a  (pas de restriction de taux)
```

```
a # b  (pas de restriction de taux)
```

où les arguments *a* and *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur #. Il utilise le fichier *subtracts.csd* [examples/subtracts.csd].

Exemple 21. Exemple de l'opérateur #.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subtracts.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 - 8
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 16.000
```

Voir Aussi

+, *&&*, *||*, ***, */*, *^*, *%*

Crédits

Exemple écrit par Kevin Conder.

/

/ — Opérateur de division.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

a + b * c.

Dans de tels cas trois règles s'appliquent :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus est interprétée comme

a + (b * c)

avec * s'appliquant à b et à c et ensuite + s'appliquant à a et à b * c.

2. + et # sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

Syntaxe

```
a / b  (pas de restriction de taux)
```

où les arguments *a* et *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur /. Il utilise le fichier *divides.csd* [examples/divides.csd].

Exemple 22. Exemple de l'opérateur /.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o divides.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 / 8
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 3.000
```

Voir Aussi

-, +, &&, ||, *, ^, %

Crédits

Exemple écrit par Kevin Conder.

=

= — Réalise une simple affectation.

Syntaxe

```
ares = xarg
```

```
ires = iarg
```

```
kres = karg
```

Description

Réalise une simple affectation.

Initialisation

= (simple affectation) - Met la valeur de l'expression *iarg* (*karg, xarg*) dans le résultat nommé. On peut ainsi garder en mémoire le résultat d'une évaluation pour une utilisation ultérieure.

Exemples

Voici un exemple de l'opérateur d'affectation. Il utilise le fichier *assign.csd* [examples/assign.csd].

Exemple 23. Exemple de l'opérateur d'affectation.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o assign.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Assign a value to the variable i1.
  i1 = 1234

  ; Print the value of the i1 variable.
  print i1
endin


</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 1234.000
```

Voir Aussi

divz, *init*, *tival*

Crédits

Exemple écrit par Kevin Conder.

==

== — Teste l'égalité de deux valeurs.

Description

Teste l'égalité de deux valeurs.

Syntaxe

```
(a == b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*. (Par commodité, un seul "=" fonctionnera comme "= =".)

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précédence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et *?*, et *:*) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, *, /, *&&* et *||*).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

Exemples

Voici un exemple de l'opérateur ==. Il utilise le fichier *equals.csd* [examples/equals.csd].

Exemple 24. Exemple de l'opérateur ==.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o equal.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Get the 4th p-field from the score.
  k1 =  p4

  ; Is it equal to 3? (1 = true, 0 = false)
  k2 = (p4 == 3 ? 1 : 0)

  ; Print the values of k1 and k2.
  printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin


</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 0.000000
```

Voir Aussi

>=, >, <=, <, !=

Crédits

Exemple écrit par Kevin Conder.

^

^ — Opérateur d'élévation à une puissance.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

a + b * c.

Dans de tels cas trois règles s'appliquent :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus est interprétée comme

a + (b * c)

avec * s'appliquant à b et à c et ensuite + s'appliquant à a et à b * c.

2. + et # sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

L'opérateur ^ élève *a* à la puissance *b*. *b* ne doit pas être de taux audio. A utiliser avec précaution car les règles de précédence peuvent ne pas fonctionner correctement. Voir *pow*. (Nouveau dans la version 3.493 de Csound.)

Syntaxe

```
a ^ b  (b pas de taux audio)
```

où les arguments *a* et *b* peuvent être des expressions.

Exemples

Voici un exemple de l'opérateur ^. Il utilise le fichier *raises.csd* [examples/raises.csd].

Exemple 25. Exemple de l'opérateur ^.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o raises.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 2 ^ 12
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 4096.000
```

Voir Aussi

-, *+*, *&&*, *||*, ***, */*, *%*

Crédits

Exemple écrit par Kevin Conder.

||

|| — Opérateur OU logique.

Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

a + b * c.

Trois règles s'appliquent dans de tels cas :

1. * et / s'appliquent à leurs voisins plus fortement que + et #. Ainsi l'expression ci-dessus s'interprète comme

a + (b * c)

avec * prenant b et c puis + prenant a et b * c.

2. + et # sont prioritaires sur &&, qui devance lui-même || :

a && b - c || d

est interprété comme

(a && (b - c)) || d

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

a - b - c

est interprété comme

(a - b) - c

On peut utiliser des parenthèses pour forcer un groupement particulier.

Syntaxe

```
a || b  (OU logique ; pas de taux audio)
```

où les arguments *a* et *b* peuvent être des expressions.

Voir Aussi

-, +, *&&*, *, /, ^, *%*

0dbfs

0dbfs — Fixe la valeur des 0 décibels à amplitude maximale.

Description

Fixe la valeur des 0 décibels à amplitude maximale.

Syntaxe

```
0dbfs = iarg
```

```
0dbfs
```

Initialisation

iarg -- la valeur des 0 décibels à amplitude maximale.

Exécution

La valeur par défaut est 32767, si bien que tous les orchestres existants *devraient* fonctionner.

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "*0dbfs*" représentant l'amplitude maximale possible sans écrêtage. A l'origine cette valeur valait toujours 32767 dans Csound, correspondant à l'étendue bipolaire d'un fichier son sur 16 bit ou d'un codec AN/NA sur 16 bit. Cela reste l'amplitude maximale *par défaut* dans Csound, pour des raisons de compatibilité descendante. La valeur *0dbfs* permet à Csound de produire des valeurs mises à l'échelle de n'importe quel format de sortie, que ce soit en entiers sur 16 bit ou 24 bit, en flottants sur 32 bit, et même en entiers sur 32 bit.

On peut définir 0dBFS dans l'en-tête, pour fixer l'amplitude de référence utilisée par Csound, mais on peut aussi l'utiliser comme variable dans un instrument comme ceci :

```
ipeak = 0dbfs
```

```
asig oscil 0dbfs, freq, 1
     out   asig * 0.3 * 0dbfs
```

L'opcode *0dbfs* a pour but le codage relatif à une valeur 0dbfs (et l'usage beaucoup plus fréquent des opcodes *ampdbfs()* !), plutôt que l'utilisation de valeurs d'échantillon explicites. L'utilisation de 0dbfs=1 est conforme aux pratiques de l'industrie, car l'intervalle allant de -1 à 1 est utilisé dans la plupart des formats de plugin commerciaux et dans la plupart des autres systèmes de synthèse comme Pure Data.

Les nombres en virgule flottante écrits dans un fichier, lorsque *0dbfs = 1*, ne seront effectivement pas transposés du tout en amplitude. Ainsi les nombres dans le fichier sont exactement ce que l'orchestre dit qu'ils sont.

Pour plus de détails sur les valeurs d'amplitude dans Csound, voir la section *Valeurs d'amplitude dans Csound*.

Exemples

Voici un exemple de l'opcode 0dbfs. Il utilise le fichier *0dbfs.csd* [examples/0dbfs.csd].

Exemple 26. Exemple de l'opcode 0dbfs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o 0dbfs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Set the 0dbfs to the 16-bit maximum.
0dbfs = 32767

; Instrument #1.
instr 1
  ; Linearly increase the amplitude value "kamp" from
  ; 0 to 1 over the duration defined by p3.
  kamp line 0, p3, 1

  ; Generate a basic tone using our amplitude value.
  a1 oscil kamp, 440, 1

  ; Multiply the basic tone (with its amplitude between
  ; 0 and 1) by the full-scale 0dbfs value.
  out a1 * 0dbfs
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode 0dbfs. Il utilise le fichier *0dbfs-1.csd* [examples/0dbfs-1.csd]. Cet exemple a exactement la même sortie que l'exemple précédent, mais les échantillons de sortie sont maintenant normalisés entre -1 et 1.

Exemple 27. Exemple de l'opcode 0dbfs avec une amplitude maximale de 1.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o 0dbfs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Set the 0dbfs to 1.
0dbfs = 1

; Instrument #1.
instr 1
  ; Linearly increase the amplitude value "kamp" from
  ; -90 to p4 (in dBfs) over the duration defined by p3.
  kamp line -90, p3, p4
  print ampdbfs(p4)
  ; Generate a basic tone using our amplitude value.
a1 oscil ampdbfs(kamp), 440, 1

  ; Since 0dbfs = 1 we don't need to multiply the output
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3 -6
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

ampdbfs()

Crédits

Auteur : Richard Dobson
Mai 2002

Nouveau dans la version 4.10

Exemple écrit par Kevin Conder.

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.20

<< — Bitshift left operator.

Description

The bitshift operators shift the bits to the left or to the right the number of bits given.

The priority of these operators is less binding that the arithmetic ones, but more binding that the comparisons.

Parentheses may be used as above to force particular groupings.

Syntax

```
a << b  (bitshift left)
```

where the arguments *a* and *b* may be further expressions.

Examples

Here is an example of the bitshift left operator. It uses the file *bitshift.csd* [examples/bitshift.csd].

Exemple 28. Example of the bitshift left operator.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bitshift.wav -W --nosound ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

instr 1 ;bit shift right
ival = p4>>p5
printf_i "%i>>%i = %i\n", 1, p4, p5, ival
endin

instr 2 ;bit shift left
ival = p4<<p5
printf_i "%i<<%i = %i\n", 1, p4, p5, ival
endin

</CsInstruments>
<CsScore>
i 1    0    0.1  2    1
i 1    +     .   3    1
i 1    +     .   7    2
i 1    +     .   16    1
i 1    +     .   16    2
i 1    +     .   16    3
```

```
i 2    5     0.1 1    1
i 2    +     .   1    2
i 2    +     .   1    3
i 2    +     .   1    4
i 2    +     .   2    1
i 2    +     .   2    2
i 2    +     .   2    3
i 2    +     .   3    2
e
</CsScore>
</CsoundSynthesizer>
```

The example above will produce the following output:

```
2>>1 = 1
B  0.000 ..  0.100 T  0.100 TT  0.100 M:      0.0      0.0
3>>1 = 1
B  0.100 ..  0.200 T  0.200 TT  0.200 M:      0.0      0.0
7>>2 = 1
B  0.200 ..  0.300 T  0.300 TT  0.300 M:      0.0      0.0
16>>1 = 8
B  0.300 ..  0.400 T  0.400 TT  0.400 M:      0.0      0.0
16>>2 = 4
B  0.400 ..  0.500 T  0.500 TT  0.500 M:      0.0      0.0
16>>3 = 2
B  0.500 ..  5.000 T  5.000 TT  5.000 M:      0.0      0.0
new alloc for instr 2:
1<<1 = 2
B  5.000 ..  5.100 T  5.100 TT  5.100 M:      0.0      0.0
1<<2 = 4
B  5.100 ..  5.200 T  5.200 TT  5.200 M:      0.0      0.0
1<<3 = 8
B  5.200 ..  5.300 T  5.300 TT  5.300 M:      0.0      0.0
1<<4 = 16
B  5.300 ..  5.400 T  5.400 TT  5.400 M:      0.0      0.0
2<<1 = 4
B  5.400 ..  5.500 T  5.500 TT  5.500 M:      0.0      0.0
2<<2 = 8
B  5.500 ..  5.600 T  5.600 TT  5.600 M:      0.0      0.0
2<<3 = 16
B  5.600 ..  5.700 T  5.700 TT  5.700 M:      0.0      0.0
3<<2 = 12
```

See Also

>>, &, | #

>>

>> — Bitshift right operator.

Description

The bitshift operators shift the bits to the left or to the right the number of bits given.

The priority of these operators is less binding that the arithmetic ones, but more binding that the comparisons.

Parentheses may be used as above to force particular groupings.

Syntax

```
a >> b  (bitshift left)
```

where the arguments *a* and *b* may be further expressions.

Examples

See the entry for the << operator for an example.

See Also

<<, &, | #

&

& — Opérateur ET binaire.

Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Syntaxe

```
a & b  (ET binaire)
```

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

Voir Aussi

|, #, ¬

|

| — Opérateur OU binaire.

Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Syntaxe

```
a | b  (OU binaire)
```

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

Voir Aussi

&, #, ¬

¬

¬ — Opérateur NON binaire.

Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Syntaxe

```
~ a  (NON binaire)
```

où l'argument *a* peut être une expression. Il est converti dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

Voir Aussi

&, | #

# #

— Opérateur NON-EQUIVALENCE binaire.

Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Syntaxe

```
a # b  (NON-EQUIVALENCE binaire)
```

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

Voir Aussi

&, / ¬

a

a — Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

Description

Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

Syntaxe

```
a(x) (arguments de taux-k seulement)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode a. Il utilise le fichier *opa.csd* [examples/opa.csd].

Exemple 29. Exemple de l'opcode a.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o a.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create a sine wave at k-rate.
  kwave oscil 20000, 440, 1

  ; Convert the k-rate sine wave to the audio-rate.
  awave = a(kwave)

  ; Output the audio-rate version of sine wave.
  out awave
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
```

```
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

i, *k*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

abetarand

abetarand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

abexprnd

abexprnd — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

abs

abs — Retourne une valeur absolue.

Description

Retourne la valeur absolue de *x*.

Syntaxe

```
abs(x) (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode abs. Il utilise le fichier *abs.csd* [examples/abs.csd].

Exemple 30. Exemple de l'opcode abs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = -6
  i2 = abs(i1)

  print i2
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie incluera des lignes comme :

```
instr 1:  i2 = 6.000
```

Voir Aussi

exp, *frac*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Exemple écrit par Kevin Conder.

acauchy

acauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

active

active — Retourne le nombre d'instances actives d'un instrument.

Description

Retourne le nombre d'instances actives d'un instrument.

Syntaxe

```
ir active insnum
```

```
kres active kinsnum
```

Initialisation

insnum -- numéro de l'instrument concerné

Exécution

kinsnum -- numéro de l'instrument concerné

active retourne le nombre d'instances actives de l'instrument numéro *insnum*/*kinsnum*. A partir de Csound 4.17 la sortie est mise à jour au taux-k (si l'argument d'entrée est de taux-k), pour permettre un comptage dynamique des instances d'instrument.

Exemples

Voici un exemple de l'opcode active. Il utilise le fichier *active.csd* [examples/active.csd].

Exemple 31. Exemple simple de l'opcode active.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o active.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
  ; Generate a really noisy waveform.
  anoisy rand 44100
```

```
  ; Turn down its amplitude.
  aoutput gain anoisy, 2500
  ; Send it to the output.
  out aoutput
endin

; Instrument #2 - counts active instruments.
instr 2
  ; Count the active instances of Instrument #1.
  icount active 1
  ; Print the number of active instances.
  print icount
endin


</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 2:  icount = 1.000
instr 2:  icount = 2.000
```

Voici un exemple plus avancé de l'opcode active. Il affiche le résultat de l'opcode active au taux-k. Il utilise le fichier *active_k.csd* [examples/active_k.csd].

Exemple 32. Exemple de l'opcode active au taux-k.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o active_k.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
  ; Generate a really noisy waveform.
  anoisy rand 44100
  ; Turn down its amplitude.
  aoutput gain anoisy, 2500
  ; Send it to the output.
  out aoutput
endin
```

```
; Instrument #2 - counts active instruments at k-rate.
instr 2
  ; Count the active instances of Instrument #1.
  kcount active 1
  ; Print the number of active instances.
  printk2 kcount
endin


</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
 i2     1.00000
 i2     2.00000
```

Voici un autre exemple de l'opcode active, qui utilise le nombre d'instances pour calculer le gain. Il utilise le fichier *active_scale.csd* [examples/active_scale.csd].

Exemple 33. Exemple de l'opcode active au taux-k.

```
<CsoundSynthesizer>
<CsOptions>

</CsOptions>
<CsInstruments>

sr= 44100
ksmps = 64
0dbfs = 1

;by Victor Lazzarini 2008

instr 1
kscal active  1
kamp port   1/kscal, 0.01
asig  oscili  kamp, p4, 1
kenv linseg 0, 0.1,1,p3-0.2,1,0.1, 0

        out asig*kenv
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1

i1 0 10 440
i1 1 3  220
i1 2 5  350
i1 4 3  700
</CsScore>
```

```
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Juillet 1999

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.57 de Csound

adsr

adsr — Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

Description

Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

Syntaxe

```
ares adsr iatt, idec, islev, irel [, idel]
```

```
kres adsr iatt, idec, islev, irel [, idel]
```

Initialisation

iatt -- durée de l'attaque (attack)

idec -- durée de la première chute (decay)

islev -- niveau d'entretien (sustain)

irel -- durée de la chute (release)

idel -- délai de niveau zéro avant le démarrage de l'enveloppe

Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

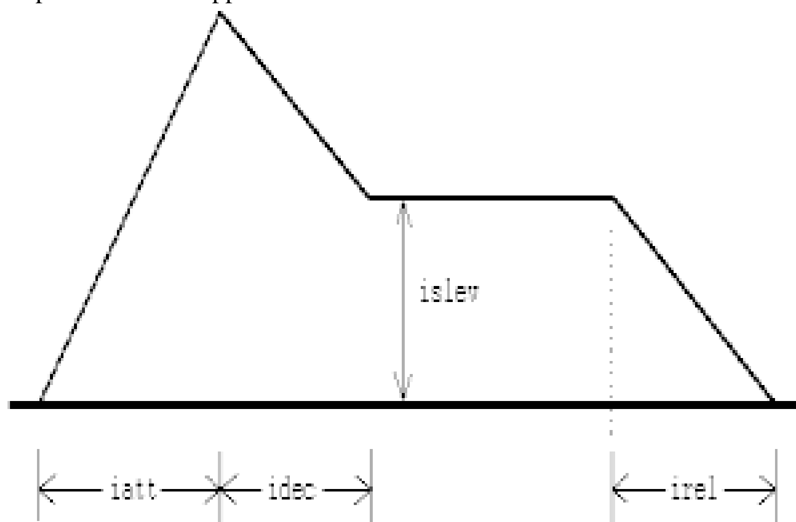


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des évènements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et

peut donc être utilisé dans les applications MIDI.

adsr est nouveau dans la version 3.49 de Csound.

Exemples

Voici un exemple de l'opcode adsr. Il utilise le fichier *adsr.csd* [examples/adsr.csd].

Exemple 34. Exemple de l'opcode adsr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adsr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
  ; Set the amplitude.
  kamp init 20000
  ; Get the frequency from the fourth p-field.
  kcps = cpspch(p4)

  a1 vco kamp, kcps, 1
  out a1
endin

; Instrument #2 - instrument with an ADSR envelope.
instr 2
  iatt = 0.05
  idec =  0.5
  islev = 0.08
  irel = 0.008

  ; Create an amplitude envelope.
  kenv adsr iatt, idec, islev, irel
  kamp = kenv * 20000

  ; Get the frequency from the fourth p-field.
  kcps = cpspch(p4)

  a1 vco kamp, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Set the tempo to 120 beats per minute.
t 0 120

; Play a melody with Instrument #1.
; p4 = frequency in pitch-class notation.
i  1  0   1  8.04
i  1  1   1  8.04
i  1  2   1  8.05
```

```
i  1  3   1  8.07
i  1  4   1  8.07
i  1  5   1  8.05
i  1  6   1  8.04
i  1  7   1  8.02
i  1  8   1  8.00
i  1  9   1  8.00
i  1  10  1  8.02
i  1  11  1  8.04
i  1  12  2  8.04
i  1  14  2  8.02

; Repeat the melody with Instrument #2.
; p4 = frequency in pitch-class notation.
i  2  16  1  8.04
i  2  17  1  8.04
i  2  18  1  8.05
i  2  19  1  8.07
i  2  20  1  8.07
i  2  21  1  8.05
i  2  22  1  8.04
i  2  23  1  8.02
i  2  24  1  8.00
i  2  25  1  8.00
i  2  26  1  8.02
i  2  27  1  8.04
i  2  28  2  8.04
i  2  30  2  8.02
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

madsr, *mxadsr*, *xadsr*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.49

Exemple écrit par Kevin Conder.

adsyn

adsyn — La sortie est la somme d'un ensemble de sinusoïdes contrôlées individuellement, jouées par un banc d'oscillateurs.

Description

La sortie est la somme d'un ensemble de sinusoïdes contrôlées individuellement, jouées par un banc d'oscillateurs.

Syntaxe

```
ares adsyn kamod, kfmod, ksmod, ifilcod
```

Initialisation

ifilcod -- entier ou chaîne de caractères dénotant un fichier de contrôle issu de l'analyse d'un signal audio. Un entier dénote le suffixe d'un fichier *adsyn.m* ou *pvoc.m* ; une chaîne de caractères (entre doubles apostrophes) donne un nom de fichier, optionnellement un nom de chemin complet. S'il ne s'agit pas d'un chemin complet, le fichier est d'abord recherché dans le répertoire courant, puis dans celui qui est indiqué par la variable d'environnement *SADIR* (si elle est définie). Le fichier de contrôle *adsyn* contient les valeurs des points charnière des enveloppes d'amplitude et de fréquence, tandis que le fichier de contrôle *pvoc* contient des données similaires organisées pour une resynthèse par tfr. L'utilisation de la mémoire dépend de la taille des fichiers impliqués, qui sont lus et maintenus entièrement en mémoire durant le calcul tout en étant partagés par les appels multiples (voir aussi *lpread*).

Exécution

kamod -- facteur d'amplitude des partiels additionnés.

kfmod -- facteur de fréquence des partiels additionnés. C'est un facteur de transposition au taux de contrôle : une valeur de 1 signifie pas de transposition, 1,5 transpose d'un quinte juste ascendante, et 0,5 d'une octave descendante.

ksmod -- facteur de vitesse des partiels additionnés.

adsyn synthétise des timbres dynamiques complexes par la méthode de synthèse additive. N'importe quel nombre de sinusoïdes, contrôlées individuellement en fréquence et en amplitude, peuvent être additionnées par une unité arithmétique très rapide pour produire un résultat de grande qualité.

Les composantes sinusoïdales sont décrites dans un fichier de contrôle qui contient des pistes d'amplitude et de fréquence définies par des points charnière. Les pistes sont des séquences de nombres entiers sur 16 bit :

-1, date, amp, date, amp,...
-2, date, fréq, date, fréq,...

telles que celles qui sont produites par l'analyse d'un fichier audio au moyen d'un filtre hétérodyne. (Pour des détails, voir *hetro*.) Les valeurs instantanées d'amplitude et de fréquence sont utilisées par un oscillateur interne en virgule fixe qui additionne chaque partiel actif dans un signal de sortie accumulé. Bien qu'il y ait une limite pratique (limite supprimée dans la version 3.47) du nombre de partiels mis à contribution, il n'y a aucune restriction quant à leur comportement dans le temps. Un son quelconque que l'on

peut décrire en termes d'évolution de sinusoïdes sera synthétisable par *adsyn* seul.

On peut aussi modifier un son décrit par un fichier de contrôle *adsyn* pendant la resynthèse. Les signaux *kamod*, *kfmod* et *ksmod* modifieront l'amplitude, la fréquence et la vitesse des partiels. Ce sont des facteurs multiplicatifs, avec *kfmod* modifiant la fréquence et *ksmod* modifiant la *vitesse* avec laquelle les segments en millisecondes définis par les points charnière sont parcourus. Ainsi, 0,7, 1,5 et 2 produiront un son plus doux, plus haut d'une quinte juste, mais deux fois moins long. Les valeurs 1, 1, 1 laisseront le son inchangé. Chacune de ces entrées peut être un signal de contrôle.

Exemples

Voici un exemple de l'opcode adsyn. Il utilise les fichiers *adsyn.csd* [examples/adsyn.csd] et *kickroll.het* [examples/kickroll.het]. Le fichier « kickroll.het » a été créé en utilisant l'utilitaire *hetro* avec le fichier audio *kickroll.wav* [examples/kickroll.wav].

Exemple 35. Exemple de l'opcode adsyn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adsyn.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; If the modulation amounts are set to 1, adsyn
  ; will not perform any special modulation.
  kamod init 1
  kfmod init 1
  ksmod init 1

  ; Re-synthesizes the file "kickroll.het".
  a1 adsyn kamod, kfmod, ksmod, "kickroll.het"

  out a1 * 32768
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Exemple écrit par Kevin Conder.

adsynt

adsynt — Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

Syntaxe

```
ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

Initialisation

iwfn -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

ifreqfn -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

iampfn -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

icnt -- nombre de partiels à générer.

iphs -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

Exécution

kamp -- amplitude de la note.

kcps -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

Exemples

Voici un exemple de l'opcode adsynt. Il utilise le fichier *adsynt.csd* [examples/adsynt.csd]. Ces deux instruments réalisent une synthèse additive. La sortie de chacun d'entre eux sonne comme un bol tibétain. Le premier est statique, car ses paramètres ne sont générés que pendant l'initialisation. Dans le second, les paramètres changent de façon continue.

Exemple 36. Exemple de l'opcode adsynt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adsynt.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1
; Generate two empty tables for adsynt.
gifrqs ftgen 2, 0, 32, 7, 0, 32, 0
; A table for freqency and amp parameters.
giamps ftgen 3, 0, 32, 7, 0, 32, 0

; Generates parameters at init time
instr 1
  ; Generate 10 voices.
  icnt = 10
  ; Init loop index.
  index = 0

; Loop only executed at init time.
loop:
  ; Define non-harmonic partials.
  ifreq pow index + 1, 1.5
  ; Define amplitudes.
  iamp = 1 / (index+1)
  ; Write to tables.
  tableiw ifreq, index, gifrqs
  ; Used by adsynt.
  tableiw iamp, index, giamps

  index = index + 1
  ; Do loop/
  if (index < icnt) igoto loop

  asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
  out asig
endin

; Generates parameters every k-cycle.
instr 2
  ; Generate 10 voices.
  icnt = 10
  ; Reset loop index.
  kindex = 0

; Loop executed every k-cycle.
loop:
  ; Generate lfo for frequencies.
  kspeed  pow kindex + 1, 1.6
  ; Individual phase for each voice.
  kphas phasorbnk kspeed * 0.7, kindex, icnt
  klfo table kphas, giwave, 1
  ; Arbitrary parameter twiddling...
  kdepth pow 1.4, kindex
  kfreq pow kindex + 1, 1.5
  kfreq = kfreq + klfo*0.006*kdepth

  ; Write freqs to table for adsynt.
  tablew kfreq, kindex, gifrqs
```

```
  ; Generate lfo for amplitudes.
  kspeed  pow kindex + 1, 0.8
  ; Individual phase for each voice.
  kphas phasorbnk kspeed*0.13, kindex, icnt, 2
  klfo table kphas, giwave, 1
  ; Arbitrary parameter twiddling...
  kamp pow 1 / (kindex + 1), 0.4
  kamp = kamp * (0.3+0.35*(klfo+1))

  ; Write amps to table for adsynt.
  tablew kamp, kindex, giamps

  kindex = kindex + 1
  ; Do loop.
  if (kindex < icnt) kgoto loop

  asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2.5 seconds.
i 1 0 2.5
; Play Instrument #2 for 2.5 seconds.
i 2 3 2.5
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Peter Neubäcker
Munich, Allemagne
Août 1999

Nouveau dans la version 3.58 de Csound

adsynt2

adsynt2 — Réalise une synthèse additive avec un nombre arbitraire de partiels - pas nécessairement harmoniques - avec interpolation.

Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques. (Voir *adsynt*)

Syntaxe

```
ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

Initialisation

iwfn -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

ifreqfn -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

iampfn -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

icnt -- nombre de partiels à générer.

iphs -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

Exécution

kamp -- amplitude de la note.

kcps -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

adsynt2 est identique à *adsynt* (by Peter Neubäcker), sauf qu'il réalise une interpolation linéaire pour les enveloppes d'amplitude de chaque partiel. Il est un peu plus lent que *adsynt*, mais l'interpolation améliore grandement la qualité du son dans les transitoires rapides des enveloppes d'amplitude lorsque *kr* < *sr* (c'est-à-dire quand *ksmps* > 1). Il n'y pas d'interpolation pour les enveloppes de hauteur, car dans ce cas la dégradation de la qualité sonore n'est pas aussi évidente même avec de grandes valeurs de *ksmps*. Il n'est pas recommandé quand *kr* = *sr* ; dans ce cas, *adsynt* est meilleur (car plus rapide).

Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

aexprand

aexprand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

aftouch

aftouch — Reçoit la valeur d'after-touch actuelle de ce canal.

Description

Reçoit la valeur d'after-touch actuelle de ce canal.

Syntaxe

```
kaft aftouch [imin] [, imax]
```

Initialisation

imin (facultatif, par défault 0) -- limite minimale des valeurs obtenues.

imax (facultatif, par défault 127) -- limite maximale des valeurs obtenues.

Exécution

Reçoit la valeur d'after-touch actuelle de ce canal.

Exemples

Voici un exemple de l'opcode aftouch. Il utilise le fichier *aftouch.csd* [examples/aftouch.csd].

Exemple 37. Exemple de l'opcode aftouch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o aftouch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  k1 aftouch

  printk2 k1
endin


</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

ampmidi, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Crédits

Auteurs : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Exemple écrit par Kevin Conder.

agauss

agauss — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

agogobel

agogobel — Obsolète.

Description

Nouveau dans la version 3.47

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *gogobel*.

alinrand

alinrand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

alpass

alpass — Réverbère un signal en entrée avec une réponse en fréquence plate.

Description

Réverbère un signal en entrée avec une réponse en fréquence plate.

Syntaxe

```
ares alpass asig, krvt, ilpt [, iskip] [, insmps]
```

Initialisation

ilpt -- durée de boucle en secondes, déterminant la « densité d'écho » de la réverbération. Cela caractérise aussi la « couleur » du filtre dont la courbe de réponse en fréquence contiendra *ilpt* * *sr*/2 sommets régulièrement espacés entre 0 et *sr*/2 (la fréquence de Nyquist). La durée de boucle n'est limitée que par la quantité de mémoire disponible. L'espace requis pour une boucle de *n* secondes est de 4*n***sr* octets. L'espace de retard est alloué et retourné comme dans *delay*.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

insmps (facultatif, par défaut 0) -- importance du retard, en nombre d'échantillons.

Exécution

krvt -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour faire décroître son amplitude à 1/1000, ou 60 dB de son amplitude originale).

Ce filtre réitère l'entrée avec une densité d'écho déterminée par la durée de boucle *ilpt*. La vitesse d'atténuation est indépendante et déterminée par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour faire décroître son amplitude à 1/1000, ou 60 dB de son amplitude originale). La sortie apparait sans retard.

Exemples

Voici un exemple de l'opcode alpass. Il utilise le fichier *alpass.csd* [examples/alpass.csd].

Exemple 38. Exemple de l'opcode alpass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o alpass.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the audio mixer.
gamix init 0

; Instrument #1.
instr 1
  ; Generate a source signal.
  a1 oscili 30000, cpspch(p4), 1
  ; Output the direct sound.
  out a1

  ; Add the source signal to the audio mixer.
  gamix = gamix + a1
endin

; Instrument #99 (highest instr number executed last)
instr 99
  krvt = 1.5
  ilpt = 0.1

  ; Filter the mixed signal.
  a99 alpass gamix, krvt, ilpt
  ; Output the result.
  out a99

  ; Empty the mixer for the next pass.
  gamix = 0
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=7.00
i 1 0 0.1 7.00
; Play Instrument #1 for a tenth of a second, p4=7.02
i 1 1 0.1 7.02
; Play Instrument #1 for a tenth of a second, p4=7.04
i 1 2 0.1 7.04
; Play Instrument #1 for a tenth of a second, p4=7.06
i 1 3 0.1 7.06

; Make sure the filter remains active.
i 99 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

comb, *reverb*, *valpass*, *vcomb*

Crédits

Auteur : William « Pete » Moss (*vcomb* et *valpass*)
Université du Texas à Austin
Austin, Texas USA
Janvier 2002

Exemple écrit par Kevin Conder.

ampdb

ampdb — Retourne l'amplitude équivalente à la valeur *x* donnée en décibel.

Description

Retourne l'amplitude équivalente à la valeur *x* donnée en décibel.

- 60 dB = 1000
- 66 dB = 1995.262
- 72 dB = 3891.07
- 78 dB = 7943.279
- 84 dB = 15848.926
- 90 dB = 31622.764

Syntaxe

```
ampdb(x)  (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode ampdb. Il utilise le fichier *ampdb.csd* [examples/ampdb.csd].

Exemple 39. Exemple de l'opcode ampdb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ampdb.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  idb = 90
  iamp = ampdb(idb)

  print iamp
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 31622.764
```

Voir Aussi

ampdbfs, *db*, *dbamp*, *dbfsamp*

Crédits

Exemple écrit par Kevin Conder.

ampdbfs

ampdbfs — Retourne l'amplitude équivalente (sur une échelle d'entiers signés sur 16 bit) à la valeur *x* de l'amplitude maximale (dB FS).

Description

Retourne l'amplitude équivalente à la valeur *x* de l'amplitude maximale (dB FS). Les valeurs logarithmiques de l'échelle en décibels sont converties en valeurs linéaires entières sur 16 bit allant de -32768 à +32767.

Syntaxe

```
ampdbfs(x)  (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode ampdbfs. Il utilise le fichier *ampdbfs.csd* [examples/ampdbfs.csd].

Exemple 40. Exemple de l'opcode ampdbfs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ampdbfs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  idb = -1
  iamp = ampdbfs(idb)

  print iamp
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 29203.621
```

Voir Aussi

ampdb, *dbamp*, *dbfsamp*, *0dbfs*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.10 de Csound.

ampmidi

ampmidi — Retourne la vélocité de l'évènement MIDI en cours.

Description

Retourne la vélocité de l'évènement MIDI en cours.

Syntaxe

```
iamp ampmidi iscal [, ifn]
```

Initialisation

iscal -- facteur de pondération de taux-i

ifn (facultatif, par défaut 0) -- numéro d'une table de fonction contenant un tableau de conversion normalisé, grâce auquel la valeur entrante est interprétée. La valeur par défaut est 0, ce qui signifie pas de conversion.

Exécution

Réçoit la vélocité de l'évènement MIDI en cours, la modifie éventuellement grâce à une table de conversion normalisée, et retourne une valeur d'amplitude dans l'intervalle 0 - *iscal*.

Exemples

Voici un exemple de l'opcode ampmidi. Il utilise le fichier *ampmidi.csd* [examples/ampmidi.csd].

Exemple 41. Exemple de l'opcode ampmidi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o ampmidi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Scale the amplitude between 0 and 1.
  ; This example expects MIDI note inputs on channel 1
  i1 ampmidi 1
```

```
  print i1
endin


</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

aftouch, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Crédits

Auteurs : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Exemple écrit par Kevin Conder.

apcauchy

apcauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

apoisson

apoisson — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

apow

apow — Obsolète.

Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

areson

areson — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode reson.

Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode reson.

Syntaxe

```
ares areson asig, kcf, kbw [, iscl] [, iskip]
```

Initialisation

iscl (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

ares -- le signal de sortie au taux audio.

asig -- le signal d'entrée au taux audio.

kcf -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

kbw -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

areson est un filtre dont les fonctions de transfert sont complémentaires de celles de *reson*. Ainsi *areson* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *areson* mais reste le complément réel de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *reson* et *areson* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant une banque de filtres adéquats en série. (La réponse résultante est le produit des différentes réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être restaurée au moyen de *balance*.

Exemples

Voici un exemple de l'opcode areson. Il utilise le fichier *areson.csd* [examples/areson.csd].

Exemple 42. Exemple de l'opcode areson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o areson.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
  ; Generate a white noise signal.
  asig rand 20000

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; Generate a white noise signal.
  asig rand 20000

  ; Filter it using the areson opcode.
  kcf init 1000
  kbw init 100
  afilt areson asig, kcf, kbw

  ; Clip the filtered signal's amplitude to 85 dB.
  a1 clip afilt, 2, ampdb(85)
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

aresonk, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

aresonk

aresonk — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode reson.

Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode reson.

Syntaxe

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]
```

Initialisation

iscl (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

kcf -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

kbw -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

aresonk est un filtre dont les fonctions de transfert sont complémentaires de celles de *resonk*. Ainsi *aresonk* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *aresonk* mais reste le complément réel de l'unité correspondante.

Voir aussi

areson, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

atone

atone — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

Syntaxe

```
ares atone asig, khp [, iskip]
```

Initialisation

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

ares -- le signal de sortie au taux audio.

asig -- le signal d'entrée au taux audio.

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

atone est un filtre dont les fonctions de transfert sont complémentaires de celles de *tone*. Ainsi *atone* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atone* mais reste le complément réel de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *tone* et *atone* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant une banque de filtres adéquats en série. (La réponse résultante est le produit des différentes réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être restaurée au moyen de *balance*.

Exemples

Voici un exemple de l'opcode atone. Il utilise le fichier *atone.csd* [examples/atone.csd].

Exemple 43. Exemple de l'opcode atone.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o atone.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
  ; Generate a white noise signal.
  asig rand 20000

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; Generate a white noise signal.
  asig rand 20000

  ; Filter it using the atone opcode.
  khp init 2000
  afilt atone asig, khp

  ; Clip the filtered signal's amplitude to 85 dB.
  a1 clip afilt, 2, ampdb(85)
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

areson, *aresonk*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

atonek

atonek — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

Syntaxe

```
kres atonek ksig, khp [, iskip]
```

Initialisation

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

atonek est un filtre dont les fonctions de transfert sont complémentaires de celles de *tonek*. Ainsi *atonek* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atonek* mais reste le complément réel de l'unité correspondante.

Voir Aussi

areson, *aresonk*, *atone*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

atonex

atonex — Emule une série de filtres utilisant l'opcode *atone*.

Description

atonex est équivalent à un filtre constitué de plusieurs couches de filtres *atone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de k cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

Syntaxe

```
ares atonex asig, khp [, inumlayer] [, iskip]
```

Initialisation

inumlayer (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

asig -- signal d'entrée

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

Voir Aussi

resonx, *tonex*

Crédits

Auteur : Gabriel Maldonado (adapté par John ffitch)
Italie

Nouveau dans la version 3.49 de Csound

atrirand

atrirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

ATSadd

ATSadd — uses the data from an ATS analysis file to perform additive synthesis.

Description

ATSadd reads from an ATS analysis file and uses the data to perform additive synthesis using an internal array of interpolating oscillators.

Syntax

```
ar ATSadd ktimepnt, kfmod, iatsfile, ifn, ipartials[, ipartialoffset, \
        ipartialincr, igatefn]
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ifn – table number of a stored function containing a sine wave for *ATSadd* and a cosine for *ATSaddnz* (see examples below for more info)

ipartials – number of partials that will be used in the resynthesis (the noise has a maximum of 25 bands)

ipartialoffset (optional) – is the first partial used (defaults to 0).

ipartialincr (optional) – sets an increment by which these synthesis opcodes counts up from *ipartialoffset* for ibins components in the re-synthesis (defaults to 1).

igatefn (optional) – is the number of a stored function which will be applied to the amplitudes of the analysis bins before resynthesis takes place. If *igatefn* is greater than 0 the amplitudes of each bin will be scaled by *igatefn* through a simple mapping process. First, the amplitudes of all of the bins in all of the frames in the entire analysis file are compared to determine the maximum amplitude value. This value is then used create normalized amplitudes as indices into the stored function *igatefn*. The maximum amplitude will map to the last point in the function. An amplitude of 0 will map to the first point in the function. Values between 0 and 1 will map accordingly to points along the function table. See the examples below.

Performance

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSadd* exactly the same as for *pvoc*.

ATSadd and *ATSaddnz* are based on *pvadd* by Richard Karpen and use files created by Juan Pampin's ATS (*Analysis - Transformation - Synthesis* [http://www-ccrma.stanford.edu/~juan/ATS.html]).

kfmod – A control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave. Used for *ATSadd* exactly the same as for *pvoc*.

ATSadd reads from an ATS analysis file and uses the data to perform additive synthesis using an internal array of interpolating oscillators. The user supplies the wave table (usually one period of a sine wave), and can choose which analysis partials will be used in the re-synthesis.

Examples

```
ktime line  0, p3, 2.5
asig  atsadd ktime, 1, "clarinet.ats", 1, 20, 2
```

In the example above, *ipartials* is 20 and ipartialoffset is 2. This will synthesize the 3rd thru 22nd partials in the "clarinet.ats" analysis file. kmod is 1 so there will be no pitch transformation. Since the *ktimepnt* envelope moves from 0 to 2.5 over the duration of the note, the analysis file will be read from 0 to 2.5 seconds of the original duration of the analysis over the duration of the csound note, this way we can change the duration independent of the pitch.

```
ktime line  0, p3, 2.5
asig  atsadd ktime, 1.0125, "clarinet.ats", 1, 20, 0, 2
```

In the above example we synthesize 20 partials as in example 1 except this time we're using a *ipartialoffset* of 0 and *ipartialincr* of 2, which means that we'll start from the first partial and synthesize 20 partials total, skipping every other one (ie. partial 1, 3, 5,...). We've also increased the pitch of the result (kfmod is set to 1.0125).

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSaddnz*, *ATSsinnoi*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSaddnz

ATSaddnz — uses the data from an ATS analysis file to perform noise resynthesis.

Description

ATSaddnz reads from an ATS analysis file and uses the data to perform additive synthesis using a modified randi function.

Syntax

```
ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ibands – number of noise bands that will be used in the resynthesis (the noise has a maximum of 25 bands)

ibandoffset (optional) – is the first noise band used (defaults to 0).

ibandincr (optional) – sets an increment by which these synthesis opcodes counts up from *ibandoffset* for ibins components in the re-synthesis (defaults to 1).

Performance

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSaddnz* exactly the same as for *pvoc* and *ATSadd*.

ATSaddnz and *ATSadd* are based on *pvadd* by Richard Karpen and use files created by Juan Pampin's ATS (*Analysis - Transformation - Synthesis* [http://www-ccrma.stanford.edu/~juan/ATS.html]).

ATSaddnz also reads from an ATS file but it resynthesizes the noise from noise energy data contained in the ATS file. It uses a modified randi function to create band limited noise and modulates that with a cosine wave, to synthesize a user specified selection of frequency bands. Modulating the noise is required to put the band limited noise in the correct place in the frequency spectrum.

Examples

```
ktime line  0, p3, 2.5
asig  atsaddnz ktime, "clarinet.ats", 25
```

In the example above we're synthesizing all 25 noise bands from the data contained in the ATS analysis file called "clarinet.ats".

```
ktime line  2.5, p3, 0
asig atsaddnz ktime, 1, "clarinet.ats", 1, 24
```

Here we synthesize only the 25th noise band (*ibandoffset* of 24 and *ibands* of 1). Also our time pointer is

going from 2.5 to 0 over the duration of the note so we're reading backwards from 2.5 seconds in the analysis file.

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSaddnz*, *ATSsinnoi*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSbufread

ATSbufread — reads data from and ATS data file and stores it in an internal data table of frequency, amplitude pairs.

Description

ATSbufread reads data from and ATS data file and stores it in an internal data table of frequency, amplitude pairs.

Syntax

```
ATSbufread ktimepnt, kfmod, iatsfile, ipartials[, ipartialoffset, \
          ipartialincr]
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ipartials – number of partials that will be used in the resynthesis (the noise has a maximum of 25 bands)

ipartialoffset (optional) – is the first partial used (defaults to 0).

ipartialincr (optional) – sets an increment by which these synthesis opcodes counts up from *ipartialoffset* for ibins components in the re-synthesis (defaults to 1).

Performance

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSbufread* exactly the same as for *pvoc*.

kfmod – an input for performing pitch transposition or frequency modulation on all of the synthesized partials, if no fm or pitch change is desired then use a 1 for this value.

ATSbufread is based on pvbufread by Richard Karpen. *ATScross*, *ATSinterpread* and *ATSpartialtap* are all dependent on *ATSbufread* just as *pvcross* and *pvinterp* are on *pvbufread*. *ATSbufread* reads data from and ATS data file and stores it in an internal data table of frequency, amplitude pairs. The data stored by an *ATSbufread* can only be accessed by other unit generators, and therefore, due to the architecture of Csound, an *ATSbufread* must come before (but not necessarily directly) any dependent unit generator. Besides the fact that *ATSbufread* doesn't output any data directly, it works almost exactly as *ATSadd*. The ugen uses a time pointer (*ktimepnt*) to index the data in time, *ipartials*, *ipartialoffset* and *ipartialincr* to select which partials to store in the table and *kfmod* to scale partials in frequency.

Examples

See the examples for *ATScross*, *ATSinterpread* and *ATSpartialtap*

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATScross

ATScross — perform cross synthesis from ATS analysis files.

Description

ATScross uses data from an ATS analysis file and data from an *ATSbufread* to perform cross synthesis.

Syntax

```
ar ATScross ktimepnt, kfmod, iatsfile, ifn, kmylev, kbuflev, ipartials \
          [, ipartialoffset, ipartialincr]
```

Initialization

iatsfile – integer or character-string denoting a control-file derived from ATS analysis of an audio signal. An integer denotes the suffix of a file ATS.m; a character-string (in double quotes) gives a filename, optionally a full pathname. If not full-path, the file is sought first in the current directory, then in the one given by the environment variable SADIR (if defined).

ifn – table number of a stored function containing a sine wave.

ipartials – number of partials that will be used in the resynthesis

ipartialoffset (optional) – is the first partial used (defaults to 0).

ipartialincr (optional) – sets an increment by which these synthesis opcodes counts up from *ipartialoffset* for ibins components in the re-synthesis (defaults to 1).

Performance

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATScross* exactly the same as for *pvoc*.

kfmod – an input for performing pitch transposition or frequency modulation on all of the synthesized partials, if no fm or pitch change is desired then use a 1 for this value.

kmylev - scales the *ATScross* component of the frequency spectrum applied to the partials from the ATS file indicated by the atscross opcode. The frequency spectrum information comes from the atscross ATS file. A value of 1 (and 0 for *kbuflev*) gives the same results as *ATSadd*.

kbuflev - scales the *ATSbufread* component of the frequency spectrum applied to the partials from the ATS file indicated by the *ATScross* opcode. The frequency spectrum information comes from the *ATSbufread* ATS file. A value of 1 (and 0 for *kmylev*) results in partials that have frequency information from the ATS file given by the *ATScross*, but amplitudes imposed by data from the ATS file given by *ATSbufread*.

ATScross uses data from an ATS analysis file (indicated by *iatsfile*) and data from an *ATSbufread* to perform cross synthesis. *ATScross* uses *ktimepnt*, *kfmod*, *ipartials*, *ipartialoffset* and *ipartialincr* just like *ATSadd*. *ATScross* synthesizes a sine-wave for each partial selected by the user and uses the frequency of that partial (after scaling in frequency by *kfmod*) to index the table created by *ATSbufread*. Interpolation is used to get in-between values. *ATScross* uses the sum of the amplitude data from its ATS file (scaled by *kmylev*) and the amplitude data gained from an *ATSbufread* (scaled by *kbuflev*) to scale the

amplitude of each partial it synthesizes. Setting *kmylev* to one and *kbuflev* to zero will make *ATScross* act exactly like *ATSadd*. Setting *kmylev* to zero and *kbuflev* to one will produce a sound that has all the partials selected by the *ATScross* ugen, but with amplitudes taken from an *ATSbufread*. The time pointers of the *ATSbufread* and *ATScross* do not need to be the same.

Examples

```
ktime line 0, p3, 2.4
ktime2 line 0, p3, .5
kline expseg 0.001, .9, 1, p3-.9, 1
kline2 expseg .001, p3, 1
atsbufread  ktime2, 1, "crt.ats", 20
aout atscross ktime, 1, "cl.ats", 1, kline, .001* (1 - kline2), 42
```

This example performs cross synthesis using two ATS files, "crt.ats" and "cl.ats". The result of this will be a sound that starts out with the shape (in frequency) of crt.ats, and ends with the shape of cl.ats. All the sine-wave frequencies come from cl.ats. The *kbuflev* value is scaled because the energy produced by applying crt.ats's frequency spectrum to cl.ats's partials is very large. Notice also that the time pointers of the atsbufread (crt.ats) and atscross (cl.ats) need not have the same value, this way you can read through the two ATS files at different rates.

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSinfo

ATSinfo — reads data out of the header of an ATS file.

Description

atsinfo reads data out of the header of an ATS file.

Syntax

```
idata ATSinfo iatsfile, ilocation
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ilocation – indicates which location in the header file to return. The data in the header gives information about the data contained in the rest of the ATS file. The possible values for ilocation are given in the following list:

0 - Sample rate (Hz)

1 - Frame Size (samples)

2 - Window Size (samples)

3 - Number of Partials

4 - Number of Frames

5 - Maximum Amplitude

6 - Maximum Frequency (Hz)

7 - Duration (seconds)

8 - ATS file Type

Performance

Macros can really improve the legibility of your csound code, I've provided my Macro Definitions below:

```
#define ATS_SAMP_RATE #0#
#define ATS_FRAME_SZ #1#
#define ATS_WIN_SZ #2#
#define ATS_N_PARTIALS #3#
#define ATS_N_FRAMES #4#
#define ATS_AMP_MAX #5#
#define ATS_FREQ_MAX #6#
#define ATS_DUR #7#
#define ATS_TYPE #8#
```

ATSinfo can be useful for writing generic instruments that will work with many ATS files, even if they have different lengths and different numbers of partials etc. Example 2 is a simple application of this.

Examples

1.

```
imax_freq atsinfo "cl.ats", $ATS_FREQ_MAX
```

In the example above we get the maximum frequency value from the ATS file "cl.ats" and store it in imax_freq. We use at Csound Macro (defined above) $ATS_FREQ_MAX, which is equivalent to the number 6.

2.

```
i_npartials atsinfo p4, $ATS_N_PARTIALS
i_dur         atsinfo p4, $ATS_DUR
ktimepnt line 0, p3, i_dur
aout         atsadd ktimepnt, 1, p4, 1, i_npartials
```

In the example above we use *ATSinfo* to retrieve the duration and number of partials in the ATS file indicated by p4. With this info we synthesize the partials using atsadd. Since the duration and number of partials are not "hard-coded" we can use this code with any ats file.

See also

ATSread, *ATSreadnz*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSsinnoi*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSinterpread

ATSinterpread — allows a user to determine the frequency envelope of any *ATSbufread*.

Description

ATSinterpread allows a user to determine the frequency envelope of any *ATSbufread*.

Syntax

```
kamp ATSinterpread kfreq
```

Performance

kfreq - a frequency value (given in Hertz) used by *ATSinterpread* as in index into the table produced by an *ATSbufread*.

ATSinterpread takes a frequency value (kfreq in Hz). This frequency is used to index the data of an *ATSbufread*. The return value is an amplitude gained from the *ATSbufread* after interpolation. *ATSinterpread* allows a user to determine the frequency envelope of any *ATSbufread*. This data could be useful for an number of reasons, one might be performing cross synthesis of data from an ATS file and non ATS data.

Examples

```
ktime  line  0, p3, 2.4
atsbufread  ktime, 1, "cl.ats", 42
kamp  atsinterpread p4
aosc  oscili  kamp, p4, 1
```

This example shows how to use *ATSinterpread*. Here a frequency is given by the score (p4) and this frequency is given to an *ATSinterpread* (with a corresponding *ATSbufread*). The *ATSinterpread* uses this frequency to output a corresponding amplitude value, based on the atsfile given by the *ATSbufread* (cl.ats in this case). We then use that amplitude to scale a sine-wave that is synthesized with the same frequency (p4). You could extend this to include multiple sine-waves. This way you could synthesize any reasonable frequency (within the low and high frequencies of the indicated ATS file), and maintain the shape (in frequency) of the indicated atsfile (given by the *ATSbufread*).

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATScross*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSread

ATSread — reads data from an ATS file.

Description

ATSread returns the amplitude (*kamp*) and frequency (*kfreq*) information of a user specified partial contained in the ATS analysis file at the time indicated by the time pointer *ktimepnt*.

Syntax

```
kfreq, kamp ATSread ktimepnt, iatsfile, ipartial
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ipartial – the number of the analysis partial to return the frequency in Hz and amplitude.

Performance

kfreq, kamp - outputs of the *ATSread* unit. These values represent the frequency and amplitude of a specific partial selected by the user using ipartial. The partials' informations are derived from an ATS analysis. *ATSread* linearly interpolates the frequency and amplitude between frames in the ATS analysis file at k-rate. The output is dependent on the data in the analysis file and the pointer *ktimepnt*.

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSread* exactly the same as for *pvoc* and *ATSadd*.

Examples

```
ktime line  0, p3, 2.5
kfreq, kamp atsread ktime, "clarinet.ats", 2
aout oscili 1000000 * kamp, kfreq, 1
```

Here we're using *ATSread* to get the 2nd partial's frequency and amplitude data out of the 'clarinet.ats' ATS analysis file. We're using that data to drive an oscillator, but we could use it for anything else that can take a k-rate input, like the bandwidth and resonance of a filter etc.

See also

ATSreadnz, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSsinnoi*

Credits

Author: Alex Norman
Seattle,Washington

2004

ATSreadnz

ATSreadnz — reads data from an ATS file.

Description

ATSreadnz returns the energy (*kenergy*) of a user specified noise band (1-25 bands) at the time indicated by the time pointer *ktimepnt*.

Syntax

```
kenergy ATSreadnz ktimepnt, iatsfile, iband
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ibands – the number of the noise band to return the energy data.

Performance

kenergy outputs the linearly interpolated energy of the noise band indicated in *iband*. The output is dependent on the data in the analysis file and the *ktimepnt*.

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSreadnz* exactly the same as for *pvoc* and *ATSadd*.

ATSaddnz reads from an ATS file and resynthesizes the noise from noise energy data contained in the ATS file. It uses a modified randi function to create band limited noise and modulates that with a user supplied wave table (one period of a cosine wave), to synthesize a user specified selection of frequency bands. Modulating the noise is required to put the band limited noise in the correct place in the frequency spectrum.

An ATS analysis differs from a pvanal in that ATS tracks the partials and computes the noise energy of the sound being analyzed. For more info about ATS analysis read Juan Pampin's description on the the *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html] web-page.

Examples

```
ktime line 2.5, p3, 0
kenergy atsreadnz ktime, "clarinet.ats", 5
```

Here we are extracting the noise energy from band 5 in the 'clarinet.ats' ATS analysis file. We're actually reading backwards from 2.5 seconds to the beginning of the analysis file. We could use this to synthesize noise like this:

```
anoise randi sqrt(kenergy), 55
          aout        oscili  40000000000000000000000000, 455, 2
aout = aout * anoise
```

Function table 2 used in the oscillator is a cosine, which is needed to shift the band limited noise into the correct place in the frequency spectrum. The randi function creates a band of noise centered about 0 Hz that has a bandwidth of about 110 Hz; multiplying it by a cosine will shift it to be centered at 455 Hz, which is the center frequency of the 5th critical noise band. This is only an example, for synthesizing the noise you'd be better off just using *ATSaddnz* unless you want to use your own noise synthesis algorithm. Maybe you could use the noise energy for something else like applying a small amount of jitter to specific partials or for controlling something totally unrelated to the source sound?

See also

ATSread, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSsinnoi*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSpartialtap

ATSpartialtap — returns a frequency, amplitude pair from an *ATSbufread* opcode.

Description

ATSpartialtap takes a partial number and returns a frequency, amplitude pair. The frequency and amplitude data comes from an atsbufread *ATSbufread* opcode.

Syntax

```
kfrq, kamp ATSpartialtap ipartialnum
```

Initialization

ipartialnum - indicates the partial that the *ATSpartialtap* opcode should read from an *ATSbufread*.

Performance

kfrq - returns the frequency value for the requested partial.

kamp - returns the amplitude value for the requested partial.

ATSpartialtap takes a partial number and returns a frequency, amplitude pair. The frequency and amplitude data comes from an *ATSbufread* opcode. This is more restricted version of *ATSread*, since each *ATSread* opcode has its own independent time pointer, and *ATSpartialtap* is restricted to the data given by an *ATSbufread*. Its simplicity is its attractive feature.

Examples

```
ktime line 0, p3, 2.4
atsbufread  ktime, 1, "crt.ats", 20
kfreq1, kamp1 atspartialtap 1
kfreq2, kamp2 atspartialtap 10
kfreq3, kamp3 atspartialtap 20
```

This example here uses an *ATSpartialtap*, and an *ATSbufread* to read partials 1, 10 and 20 from 'crt.ats'. These amplitudes and frequencies could be used to re-synthesize those partials, or something all together different.

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSadd*, *ATSaddnz*

Credits

Author: Alex Norman
Seattle,Washington
2004

ATSsinnoi

ATSsinnoi — uses the data from an ATS analysis file to perform resynthesis.

Description

ATSsinnoi reads data from an ATS data file and uses the information to synthesize sines and noise together.

Syntax

```
ar ATSsinnoi ktimepnt, ksinlev, knzlev, kfmod, iatsfile, ipartials \
          [, ipartialoffset, ipartialincr]
```

Initialization

iatsfile – the ATS number (n in ats.n) or the name in quotes of the analysis file made using *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html].

ipartials – number of partials that will be used in the resynthesis (the noise has a maximum of 25 bands)

ipartialoffset (optional) – is the first partial used (defaults to 0).

ipartialincr (optional) – sets an increment by which these synthesis opcodes counts up from *ipartialoffset* for ibins components in the re-synthesis (defaults to 1).

Performance

ktimepnt – The time pointer in seconds used to index the ATS file. Used for *ATSsinnoi* exactly the same as for *pvoc*.

ksinlev - controls the level of the sines in the *ATSsinnoi* ugen. A value of 1 gives full volume sinewaves.

knzlev - controls the level of the noise components in the *ATSsinnoi* ugen. A value of 1 gives full volume noise.

kfmod – an input for performing pitch transposition or frequency modulation on all of the synthesized partials, if no fm or pitch change is desired then use a 1 for this value.

ATSsinnoi reads data from an ATS data file and uses the information to synthesize sines and noise together. The noise energy for each band is distributed equally among each partial that falls in that band. Each partial is then synthesized, along with that partial's noise component. Each noise component is then modulated by the corresponding partial to be put in the correct place in the frequency spectrum. The level of the noise and the partials are individually controllable. See the *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html] webpage for more info about the sinnoi synthesis. An ATS analysis differs from a pvanal in that ATS tracks the partials and computes the noise energy of the sound being analyzed. For more info about ATS analysis read Juan Pampin's description on the the *ATS* [http://www-ccrma.stanford.edu/~juan/ATS.html] web-page.

Examples

```
ktime line  0, p3, 2.5
asig atssinnoi ktime, 1, 1, 1, "clarinet.ats", 42
```

Here we synthesize both the noise and the sinewaves (all 42 partials) contained in "clarinet.ats" together. The relative volumes of the noise and the partials are unaltered (each set to 1).

```
ktime line  0, p3, 2.5
knzfade expon  0.001, p3, 2.5
asig atssinnoi ktime, 1, knzfade, 1, "clarinet.ats", 42
```

This example here is like example 5 except that we use an envelope to control *knzlev* (the noise level). The result of this will be a clarinet sound that has its noise component fade in over the duration of the note.

See also

ATSread, *ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *AT-Saddnz*

Credits

Author: Alex Norman
Seattle,Washington
2004

aunirand

aunirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

aweibull

aweibull — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

babo

babo — A physical model reverberator.

Description

babo stands for *ba*ll-within-the-*bo*x. It is a physical model reverberator based on the paper by Davide Rocchesso "The Ball within the Box: a sound-processing metaphor", Computer Music Journal, Vol 19, N.4, pp.45-47, Winter 1995.

The resonator geometry can be defined, along with some response characteristics, the position of the listener within the resonator, and the position of the sound source.

Syntax

```
a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]
```

Initialization

irx, iry, irz -- the coordinates of the geometry of the resonator (length of the edges in meters)

idiff -- is the coefficient of diffusion at the walls, which regulates the amount of diffusion (0-1, where 0 = no diffusion, 1 = maximum diffusion - default: 1)

ifno -- expert values function: a function number that holds all the additional parameters of the resonator. This is typically a GEN2--type function used in non-rescaling mode. They are as follows:

- *decay* -- main decay of the resonator (default: 0.99)
- *hydecay* -- high frequency decay of the resonator (default: 0.1)
- *rcvx, rcvy, rcvz* -- the coordinates of the position of the receiver (the listener) (in meters; 0,0,0 is the resonator center)
- *rdistance* -- the distance in meters between the two pickups (your ears, for example - default: 0.3)
- *direct* -- the attenuation of the direct signal (0-1, default: 0.5)
- *early_diff* -- the attenuation coefficient of the early reflections (0-1, default: 0.8)

Performance

asig -- the input signal

ksrcx, ksrcy, ksrcz -- the virtual coordinates of the source of sound (the input signal). These are allowed to move at k-rate and provide all the necessary variations in terms of response of the resonator.

Examples

Here is a simple example of the babo opcode. It uses the file *babo.csd* [examples/babo.csd], and *beats.wav* [examples/beats.wav].

Exemple 44. A simple example of the babo opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o babo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; minimal babo instrument
;
instr 1
       ix     = p4  ; x position of source
       iy     = p5  ; y position of source
       iz     = p6  ; z position of source
       ixsize = p7  ; width  of the resonator
       iysize = p8  ; depth  of the resonator
       izsize = p9  ; height of the resonator

ainput soundin "beats.wav"

al,ar  babo    ainput*0.7, ix, iy, iz, ixsize, iysize, izsize

       outs    al,ar
endin


</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; simple babo usage:
;
;p4      : x position of source
;p5      : y position of source
;p6      : z position of source
;p7      : width   of the resonator
;p8      : depth   of the resonator
;p9      : height  of the resonator
;
i  1   0   10 6   4   3    14.39 11.86 10
;            ^^^^^^^    ^^^^^^^^^^^^^^
;            |||||||    +++++++++++++: optimal room dims according to
;            |||||||                   Milner and Bernard JASA 85(2), 1989
;            +++++++++: source position
e


</CsScore>
</CsoundSynthesizer>
```

Here is an advanced example of the babo opcode. It uses the file *babo_expert.csd* [examples/babo_expert.csd], and *beats.wav* [examples/beats.wav].

Exemple 45. An advanced example of the babo opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o babo_expert.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; full blown babo instrument with movement
;
instr 2
  ixstart = p4   ; start x position of source (left-right)
  ixend   = p7   ; end   x position of source
  iystart = p5   ; start y position of source (front-back)
  iyend   = p8   ; end   y position of source
  izstart = p6   ; start z position of source (up-down)
  izend   = p9  ; end   z position of source
  ixsize  = p10  ; width  of the resonator
  iysize  = p11  ; depth  of the resonator
  izsize  = p12  ; height of the resonator
  idiff   = p13  ; diffusion coefficient
  iexpert = p14  ; power user values stored in this function

ainput    soundin "beats.wav"
ksource_x line    ixstart, p3, ixend
ksource_y line    iystart, p3, iyend
ksource_z line    izstart, p3, izend

al,ar     babo    ainput*0.7, ksource_x, ksource_y, ksource_z, ixsize, iysize, izsize, idiff, iexpert

          outs    al,ar
endin


</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; full blown instrument
;p4         : start x position of source (left-right)
;p5         : end   x position of source
;p6         : start y position of source (front-back)
;p7         : end   y position of source
;p8         : start z position of source (up-down)
;p9         : end   z position of source
;p10        : width  of the resonator
;p11        : depth  of the resonator
;p12        : height of the resonator
;p13        : diffusion coefficient
;p14        : power user values stored in this function

;         decay  hidecay rx ry rz rdistance direct early_diff
f1  0 8 -2  0.95   0.95    0  0  0    0.3     0.5      0.8  ; brighter
f2  0 8 -2  0.95   0.5     0  0  0    0.3     0.5      0.8  ; default (to be set as)
f3  0 8 -2  0.95   0.01    0  0  0    0.3     0.5      0.8  ; darker
f4  0 8 -2  0.95   0.7     0  0  0    0.3     0.1      0.4  ; to hear the effect of diffusion
f5  0 8 -2  0.9    0.5     0  0  0    0.3     2.0      0.98 ; to hear the movement
f6  0 8 -2  0.99   0.1     0  0  0    0.3     0.5      0.8  ; default vals
;         ^
;         ----- gen. number: negative to avoid rescaling


i2 0 10   6   4   3   6   4  3  14.39  11.86  10    1  6 ; defaults
i2 +   4  6   4   3   6   4  3  14.39  11.86  10    1  1 ; hear brightness 1
i2 +   4  6   4   3  -6  -4  3  14.39  11.86  10    1  2 ; hear brightness 2
i2 +   4  6   4   3  -6  -4  3  14.39  11.86  10    1  3 ; hear brightness 3
i2 +   3 .6 .4 .3 -.6 -.4 .3  1.439  1.186  1.0 0.0  4 ; hear diffusion 1
i2 +   3 .6 .4 .3 -.6 -.4 .3  1.439  1.186  1.0 1.0  4 ; hear diffusion 2
i2 +   4 12   4   3 -12  -4 -3  24.39  21.86  20    1  5 ; hear movement
;
```

```
i2 +   4  6  4  3   6   4  3  14.39  11.86   10   1  1 ; hear brightness 1
i2 +   4  6  4  3  -6  -4  3  14.39  11.86   10   1  2 ; hear brightness 2
i2 +   4  6  4  3  -6  -4  3  14.39  11.86   10   1  3 ; hear brightness 3
i2 +   3 .6 .4 .3 -.6 -.4 .3  1.439  1.186  1.0 0.0  4 ; hear diffusion 1
i2 +   3 .6 .4 .3 -.6 -.4 .3  1.439  1.186  1.0 1.0  4 ; hear diffusion 2
i2 +   4 12  4  3 -12  -4 -3  24.39  21.86   20   1  5 ; hear movement
;        ^^^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^   ^  ^
;        |||||||||||||||||||  |||||||||||||||||   |   --: expert values function
;        |||||||||||||||||||  |||||||||||||||||   +--: diffusion
;        |||||||||||||||||||  -----------------: optimal room dims according to Milner and Bernard JASA 8
;        |||||||||||||||||||
;        -------------------: source position start and end
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Paolo Filippi
Padova, Italy
1999

Nicola Bernardini
Rome, Italy
2000

New in Csound version 4.09

balance

balance — Adjust one audio signal according to the values of another.

Description

The rms power of asig can be interrogated, set, or adjusted to match that of a comparator signal.

Syntax

```
ares balance asig, acomp [, ihp] [, iskip]
```

Initialization

ihp (optional) -- half-power point (in Hz) of a special internal low-pass filter. The default value is 10.

iskip (optional, default=0) -- initial disposition of internal data space (see *reson*). The default value is 0.

Performance

asig -- input audio signal

acomp -- the comparator signal

balance outputs a version of *asig*, amplitude-modified so that its rms power is equal to that of a comparator signal *acomp*. Thus a signal that has suffered loss of power (eg., in passing through a filter bank) can be restored by matching it with, for instance, its own source. It should be noted that *gain* and *balance* provide amplitude modification only - output signals are not altered in any other respect.

Examples

Here is an example of the balance opcode. It uses the file *balance.csd* [examples/balance.csd].

Exemple 46. Example of the balance opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o balance.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a band-limited pulse train.
  asrc buzz 30000, 440, sr/440, 1

  ; Send the source signal through 2 filters.
  a1 reson asrc, 1000, 100
  a2 reson a1, 3000, 500

  ; Balance the filtered signal with the source.
  afin balance a2, asrc

  out afin
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

gain, *rms*

bamboo

bamboo — Modèle semi-physique d'un son de bambou.

Description

bamboo est un modèle semi-physique d'un son de bambou. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]
```

Initialisation

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 1,25.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,9999 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,9999 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,05.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

ifreq (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2800.

ifreq1 (facultatif) -- la première fréquence de résonance. La valeur par défaut est 2240.

ifreq2 (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 3360.

Exécution

kamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

Exemples

Voici un exemple de l'opcode bamboo. Il utilise le fichier *bamboo.csd* [examples/bamboo.csd].

Exemple 47. Exemple de l'opcode bamboo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bamboo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 01  ;example of bamboo
a1  bamboo p4, 0.01
    out a1
    endin


</CsInstruments>
<CsScore>

i1 0 1 20000
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

dripwater, *guiro*, *sleighbells*, *tambourine*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

barmodel

barmodel — Crée un timbre similaire à une barre de métal frappée.

Description

La sortie audio est un timbre semblable à celui d'une barre de métal frappée, mettant en œuvre un modèle physique développé à partir de la résolution de l'équation différentielle. On contrôle les conditions aux limites ainsi que les caractéristiques de la barre.

Syntaxe

```
ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid
```

Initialisation

iK -- paramètre de raideur sans dimension. Si ce paramètre est négatif, l'initialisation est ignorée et l'état précédent de la barre est prolongé.

ib -- paramètre de perte des hautes fréquences (à garder petit).

iT30 -- temps de décroissance à 30 db en secondes.

ipos -- position le long de la barre où a lieu la frappe.

ivel -- vitesse de frappe normalisée.

iwid -- largeur spatiale de la frappe.

Exécution

Une note est jouée sur une barre métallique, avec les arguments suivants.

kbcL -- Condition aux limites à l'extémité gauche de la barre (1 fixée, 2 pivotante, 3 libre).

kbcR -- Condition aux limites à l'extémité droite de la barre (1 fixée, 2 pivotante, 3 libre).

kscan -- Taux de lecture de la position de sortie.

Noter que le changement des conditions aux limites pendant l'exécution peut provoquer des bruits parasites ; cette possibilité est offerte à titre expérimental. L'utilisation d'un *kscan* différent de zéro peut produire des réintroductions apparentes du son à cause de la modulation.

Exemples

Voici un exemple de l'opcode barmodel. Il utilise le fichier *barmodel.csd* [examples/barmodel.csd].

Exemple 48. Exemple de l'opcode barmodel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o barmodel.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
  sr        =           44100
  kr        =           4410
  ksmps     =           10
  nchnls    =           1

; Instrument #1.
instr 1
  aq        barmodel    1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
            out         aq
endin


</CsInstruments>
<CsScore>


i1 0.0 0.5  3 0.2 500  0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800  0.05
e
/* barmodel */

</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Stefan Bilbao
Université d'Edimbourg, UK
Auteur : John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 5.01 de Csound

bbcutm

bbcutm — Extrait des segments dans le style breakbeat à partir d'un flux audio mono.

Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

3+ 3R + 2

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

Syntaxe

```
a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \
      [, istutterspeed] [, istutterchance] [, ienvchoice ]
```

Initialisation

ibps -- Tempo pour les coupures, en pulsations par seconde.

isubdiv -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

ibarlength -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

iphrasebars -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

inumrepeats -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

istutterspeed -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

istutterchance -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

ienvchoice -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains

de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

Exécution

asource -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des évènements audio futurs.

Exemples

Voici un exemple de l'opcode bbcutm. Il utilise les fichiers *bbcutm.csd* [examples/bbcutm.csd] et *beats.wav* [examples/beats.wav].

Exemple 49. Un exemple simple de l'opcode bbcutm.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bbcutm.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Play an audio file normally.
instr 1
  asource soundin "beats.wav"
  out asource
endin

; Instrument #2 - Cut-up an audio file.
instr 2
  asource soundin "beats.wav"

  ibps = 4
  isubdiv = 8
  ibarlength = 4
  iphrasebars = 1
  inumrepeats = 2

  a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 3 2
e


</CsScore>
</CsoundSynthesizer>
```

Voici quelques exemples plus avancés ...

Exemple 50. Premiers pas - versions mono et stéréo

```
<CsoundSynthesizer>
<CsInstruments>
sr        =         44100
kr        =         4410
ksmps     =         10
nchnls    =         2

instr 1
    asource diskin "break7.wav",1,0,1 ; a source breakbeat sample, wraparound lest it stop!

    ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
    ; repeat in total (standard use) rare stuttering at 16 note speed,
    ; no enveloping
    asig bbcutm asource, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

    outs        asig,asig
endin

instr 2 ;stereo version
   asource1,asource2 diskin "break7stereo.wav", 1, 0, 1    ; a source breakbeat sample, wraparound lest

  ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
  ; repeat in total (standard use) rare stuttering at 16 note speed,
  ; no enveloping
  asig1,asig2 bbcuts asource1, asource2, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

  outs  asig1,asig2
endin

</CsInstruments>
<CsScore>
i1 0 10
i2 11 10
e
</CsScore>
</CsoundSynthesizer>
```

Exemple 51. Breaks multiples simultanés synchronisés

```
<CsoundSynthesizer>
<CsInstruments>
sr        =         44100
kr        =         4410
ksmps     =         10
nchnls    =         2

instr 1
  ibps    = 2.6937
  iplaybackspeed = ibps/p5
  asource diskin p4, iplaybackspeed, 0, 1

  asig bbcutm asource, 2.6937, p6, 4, 4, p7, 2, 0.1, 1

  out   asig
endin

</CsInstruments>
<CsScore>

;   source      bps cut repeats
i1 0 10 "break1.wav" 2.3 8   2  //2.3 is the source original tempo
i1 0 10 "break2.wav" 2.4 8   3
i1 0 10 "break3.wav" 2.5 16  4
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Exemple 52. Coupure de n'importe quelle source audio ancienne - des bruits bien plus intéressants que ceux-ci sont possibles !

```
<CsoundSynthesizer>
<CsInstruments>
sr        =        44100
kr        =        4410
ksmps     =        10
nchnls    =        2

instr 1
  asource oscil 20000, 70, 1
  ; ain, bps, subdiv, barlength, phrasebars, numrepeats,
  ;stutterspeed, stutterchance, envelopingon
  asig bbcutm asource, 2, 32, 1, 1, 2, 4, 0.6, 1
  outs  asig
endin

</CsInstruments>
<CsScore>
f1 0 256 10 1
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Exemple 53. Faux stuttering constant, impossible car on ne peut appliquer le stutter que dans la dernière demie-mesure, poourrait faire un paramètre optionnel supplémentaire de stuterring

```
<CsoundSynthesizer>
<CsInstruments>
sr        =        44100
kr        =        4410
ksmps     =        10
nchnls    =        2

instr 1
  asource diskin "break7.wav", 1, 0, 1

  ;16th note cuts- but cut size 2 over half a beat.
  ;each half beat will either survive intact or be turned into
  ;the first sixteenth played twice in succession

  asig bbcutm asource, 2.6937, 2, 0.5, 1, 2, 2, 1.0, 0
  outs  asig
endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

bbcuts

Crédits

Auteur : Nick Collins
Londres
Août 2001

Nouveau dans la version 4.13

bbcuts

bbcuts — Extrait des segments dans le style breakbeat à partir d'un flux audio stéréo.

Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

3+ 3R + 2

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

Syntaxe

```
a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
      inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]
```

Initialisation

ibps -- Tempo pour les coupures, en pulsations par seconde.

isubdiv -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

ibarlength -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

iphrasebars -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

inumrepeats -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

istutterspeed -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

istutterchance -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

ienvchoice -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains

de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

Exécution

asource -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des évènements audio futurs.

Exemples

Voir les exemples avancés dans la notice de l'opcode *bbcutm*.

Voir Aussi

bbcutm

Crédits

Auteur : Nick Collins
Londres
Août 2001

Nouveau dans la version 4.13

betarand

betarand — Générateur de nombres aléatoires de distribution beta (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution beta (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares betarand krange, kalpha, kbeta
```

```
ires betarand krange, kalpha, kbeta
```

```
kres betarand krange, kalpha, kbeta
```

Exécution

krange -- l'intervalle des nombres aléatoires (0 - *krange*).

kalpha -- valeur de alpha. Si *kalpha* est inférieur à un, ses petites valeurs favorisent les valeurs proches de 0.

kbeta -- valeur de beta. Si *kbeta* est inférieur à un, ses petites valeurs favorisent les valeurs proches de *krange*.

Si *kalpha* et *kbeta* sont tous deux égaux à un, nous obtenons une distribution uniforme. Si *kalpha* et *kbeta* sont tous deux supérieurs à un nous obtenons une sorte de distribution gaussienne. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode betarand. Il utilise le fichier *betarand.csd* [examples/betarand.csd].

Exemple 54. Exemple de l'opcode betarand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o betarand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a number between 0 and 1 with a
  ; uniform distribution.
  ; krange = 1
  ; kalpha = 1
  ; kbeta = 1

  i1 betarand 1, 1, 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 24583.412
```

Voir Aussi

seed, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Existait dans la 3.30

Exemple écrit par Kevin Conder.

bexprnd

bexprnd — Générateur de nombres aléatoires de distribution exponentielle.

Description

Générateur de nombres aléatoires de distribution exponentielle. C'est un générateur de bruit de classe x.

Syntaxe

```
ares bexprnd krange
```

```
ires bexprnd krange
```

```
kres bexprnd krange
```

Exécution

krange -- l'intervalle des nombres aléatoires (-*krange* à +*krange*)

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode bexprnd. Il utilise le fichier *bexprnd.csd* [examples/bexprnd.csd].

Exemple 55. Exemple de l'opcode bexprnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bexprnd.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Generate a random number between -1 and 1.
  ; krange = 1

  i1 bexprnd 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 1.141
```

Voir Aussi

seed, *betarand*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

bformenc

bformenc — Deprecated. Codes a signal into the ambisonic B format.

Description

Codes a signal into the ambisonic B format. Note that this opcode is deprecated as it is inaccurate, and is replaced by the much better opcode *bformenc1* which replicates all the important features.

Syntax

```
aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1
```

```
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \
      kord0, kord1 , kord2
```

```
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \
      asig, kalpha, kbeta, kord0, kord1, kord2, kord3
```

Performance

aw, ax, ay, ... -- output cells of the B format.

asig -- input signal.

kalpha –- azimuth angle in degrees (clockwise).

kbeta -- altitude angle in degrees.

kord0 -- linear gain of the zero order B format.

kord1 -- linear gain of the first order B format.

kord2 -- linear gain of the second order B format.

kord3 -- linear gain of the third order B format.

Example

Here is an example of the bformenc opcode. It uses the file *bformenc.csd* [examples/bformenc.csd].

Exemple 56. Example of the bformenc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
;-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
 -o bformenc.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

instr 1
        ; generate pink noise
        anoise pinkish 1000

        ; two full turns
        kalpha line 0, p3, 720
        kbeta = 0

        ; fade ambisonic order from 2nd to 0th during second turn
        kord0 = 1
        kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
        kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

        ; generate B format
        aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

        ; decode B format for 8 channel circle loudspeaker setup
        a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

        ; write audio out
        outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Samuel Groner
2005

New in version 5.07. Deprecated in 5.09.

bformenc1

bformenc1 — Codes a signal into the ambisonic B format.

Description

Codes a signal into the ambisonic B format

Syntax

```
aw, ax, ay, az bformenc1 asig, kalpha, kbeta
```

```
aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta
```

```
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \
      asig, kalpha, kbeta
```

Performance

aw, ax, ay, ... -- output cells of the B format.

asig -- input signal.

kalpha –- azimuth angle in degrees (anticlockwise).

kbeta -- altitude angle in degrees.

Example

Here is an example of the bformenc1 opcode. It uses the file *bformenc1.csd* [examples/bformenc1.csd].

Exemple 57. Example of the bformenc1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
;-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
 -o bformenc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

instr 1
        ; generate pink noise
        anoise pinkish 1000

        ; two full turns
        kalpha line 0, p3, 720
```

```
        kbeta = 0

        ; fade ambisonic order from 2nd to 0th during second turn
        kord0 = 1
        kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
        kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

        ; generate B format
        aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta, kord0, kord1, kord2

        ; decode B format for 8 channel circle loudspeaker setup
        a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

        ; write audio out
        outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e


</CsScore>
</CsoundSynthesizer>
```

See Also

bformdec1

Credits

Author: Richard Furse, Bruce Wiggins and Fons Adriaensen, following code by Samuel Groner 2008

New in version 5.09

bformdec

bformdec — Deprecated. Decodes an ambisonic B format signal.

Description

Decodes an ambisonic B format signal into loudspeaker specific signals. Note that this opcode is deprecated as it is inaccurate, and is replaced by the much better opcode *bformdec1* which replicates all the important features.

Syntax

```
ao1, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
      [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
      au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
      at, au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
      [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
```

Initialization

isetup –- loudspeaker setup. There are five supported setups: 1 denotes stereo setup. There must be two output cells with loudspeaker positions assumed to be (330/0, 30/0).

2 denotes quad setup. There must be four output cells. Loudspeaker positions assumed to be (45°/0), (135°/0), (225/0), (315/0).

3 is a 5.1 surround setup. There must be five output cells. LFE channel is not supported. Loudspeaker positions assumed to be (330/0), (30/0), (0/0), (250/0), (110/0).

4 denotes eight loudspeaker circle setup. There must be eight output cells. Loudspeaker positions assumed to be (22.5/0), (67.5/0), (112.5/0), (157.5/0), (202.5/0), (247.5/0), (292.5/0), (337.5/0).

5 means an eight loudspeaker cubic setup. There must be eight output cells. Loudspeaker positions assumed to be (45/0), (45/30), (135/0), (135/30), (225/0), (225/30), (315/0), (315/30).

Performance

aw, ax, ay, ... -- input signal in the B format.

ao1 .. *ao8* -– loudspeaker specific output signals.

Example

Here is an example of the bformdec opcode. It uses the file *bformenc.csd* [examples/bformenc.csd].

Exemple 58. Example of the bformdec opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
;-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
 -o bformenc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

instr 1
        ; generate pink noise
        anoise pinkish 1000

        ; two full turns
        kalpha line 0, p3, 720
        kbeta = 0

        ; fade ambisonic order from 2nd to 0th during second turn
        kord0 = 1
        kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
        kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

        ; generate B format
        aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

        ; decode B format for 8 channel circle loudspeaker setup
        a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

        ; write audio out
        outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Samuel Groner
2005

New in version 5.07. Deprecated in 5.09

bformdec1

bformdec1 — Decodes an ambisonic B format signal

Description

Decodes an ambisonic B format signal into loudspeaker specific signals.

Syntax

```
ao1, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
      [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
      au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
      at, au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
      [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
```

Initialization

Note that horizontal angles are measured anticlockwise in this description.

isetup –- loudspeaker setup. There are five supported setups:

- 1. Stereo - L(90), R(-90); this is an M+S style stereo decode.
- 2. Quad - FL(45), BL(135), BR(-135), FR(-45). This is a first-order `in-phase' decode.
- 3. 5.0 - L(30),R(-30),C(0),BL(110),BR(-110). Note that many people do not actually use the angles above for their speaker arrays and a good decode for DVD etc can be achieved using the Quad configuration to feed L, R, BL and BR (leaving C silent).
- 4. Octagon - FFL(22.5),FLL(67.5),BLL(112.5),BBL(157.5),BBR(-157.5),BRR(-112.5),FRR(-67.5),FFR(-22.5). This is a first-, second- or third-order `in-phase' decode, depending on the number of input channels.
- 5. Cube - FLD(45,-35.26),FLU(45,35.26),BLD(135,-35.26),BLU(135,35.26),BRD(-135,-35.26),BRU(-135,35.26),FRD(-45,-35.26),FRU(-45,35.26). This is a first-order `in-phase' decode.

Performance

aw, ax, ay, ... -- input signal in the B format.

ao1 .. ao8 -– loudspeaker specific output signals.

Example

Here is an example of the bformdec1 opcode. It uses the file *bformenc1.csd* [examples/bformenc1.csd].

Exemple 59. Example of the bformdec1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
;-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
 -o bformenc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

instr 1
        ; generate pink noise
        anoise pinkish 1000

        ; two full turns
        kalpha line 0, p3, 720
        kbeta = 0

        ; fade ambisonic order from 2nd to 0th during second turn
        kord0 = 1
        kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
        kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

        ; generate B format
        aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta, kord0, kord1, kord2

        ; decode B format for 8 channel circle loudspeaker setup
        a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

        ; write audio out
        outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e


</CsScore>
</CsoundSynthesizer>
```

See Also

bformenc1

Credits

Author: Richard Furse, Bruce Wiggins and Fons Adriaensen, following code by Samuel Groner 2008

New in version 5.09

binit

binit — PVS tracks to amplitude+frequency conversion.

Description

The binit opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials) and converts it into a equal-bandwidth bin-frame containing amplitude and frequency pairs (PVS_AMP_FREQ), suitable for overlap-add resynthesis (such as performed by pvsynth) or further PVS streaming phase vocoder signal transformations. For each frequency bin, it will look for a suitable track signal to fill it; if not found, the bin will be empty (0 amplitude). If more than one track fits a certain bin, the one with highest amplitude will be chosen. This means that not all of the input signal is actually 'binned', the operation is lossy. However, in many situations this loss is not perceptually relevant.

Syntax

```
fsig binit fin, isize
```

Performance

fsig -- output pv stream in PVS_AMP_FREQ format

fin -- input pv stream in TRACKS format

isize -- FFT size of output (N).

Examples

Exemple 60. Example

```
ain inch 1                    ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fbins binit  fst, 2048 ; convert it back to bins
      aout pvsynth fbins ; overlap-add resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal, conversion to bin frames and overlap-add resynthesis.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

biquad

biquad — A sweepable general purpose biquadratic digital filter.

Description

A sweepable general purpose biquadratic digital filter.

Syntax

```
ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]
```

Initialization

iskip (optional, default=0) -- if non-zero, intialization will be skipped. Default value 0. (New in Csound version 3.50)

Performance

asig -- input signal

biquad is a general purpose biquadratic digital filter of the form:

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

This filter has the following frequency response:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

This type of filter is often encountered in digital signal processing literature. It allows six user-defined k-rate coefficients.

Examples

Here is an example of the biquad opcode. It uses the file *biquad.csd* [examples/biquad.csd].

Exemple 61. Example of the biquad opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o biquad.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
  ; Get the values from the score.
  idur = p3
  iamp = p4
  icps = cpspch(p5)
  kfco = p6
  krez = p7

  ; Calculate the biquadratic filter's coefficients
  kfcon = 2*3.14159265*kfco/sr
  kalpha = 1-2*krez*cos(kfcon)*cos(kfcon)+krez*krez*cos(2*kfcon)
  kbeta = krez*krez*sin(2*kfcon)-2*krez*cos(kfcon)*sin(kfcon)
  kgama = 1+cos(kfcon)
  km1 = kalpha*kgama+kbeta*sin(kfcon)
  km2 = kalpha*kgama-kbeta*sin(kfcon)
  kden = sqrt(km1*km1+km2*km2)
  kb0 = 1.5*(kalpha*kalpha+kbeta*kbeta)/kden
  kb1 = kb0
  kb2 = 0
  ka0 = 1
  ka1 = -2*krez*cos(kfcon)
  ka2 = krez*krez

  ; Generate an input signal.
  axn vco 1, icps, 1

  ; Filter the input signal.
  ayn biquad axn, kb0, kb1, kb2, ka0, ka1, ka2
  outs ayn*iamp/2, ayn*iamp/2
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

;    Sta  Dur  Amp    Pitch Fco   Rez
i 1  0.0  1.0  20000  6.00  1000  .8
i 1  1.0  1.0  20000  6.03  2000  .95
e


</CsScore>
</CsoundSynthesizer>
```

Here is another example of the biquad opcode used for modal synthesis. It uses the file *biquad-2.csd* [examples/biquad-2.csd]. See the *Modal Frequency Ratios* appendix for other frequency ratios.

Exemple 62. Example of the biquad opcode for modal synthesis.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o biquad-2.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

/*  modal synthesis using biquad filters as oscillators
    Example by Scott Lindroth 2007 */


instr 1

    ipi = 3.1415926
    idenom = sr*0.5

    ipulseSpd = p4
    icps     = p5
    ipan = p6
    iamp    = p7
    iModes = p8

    apulse    mpulse iamp, 0

    icps    = cpspch( icps )

    ; filter gain

    iamp1 = 600
    iamp2 = 1000
    iamp3 = 1000
    iamp4 = 1000
    iamp5 = 1000
    iamp6 = 1000

    ; resonance

    irpole1 = 0.99999
    irpole2 = irpole12
    irpole3 = irpole13
    irpole4 = irpole14
    irpole5 = irpole15
    irpole6 = irpole16

    ; modal frequencies

    if (iModes == 1) goto modes1
    if (iModes == 2) goto modes2

    modes1:
    if1    = icps * 1            ;pot lid
    if2    = icps * 6.27
    if3    = icps * 3.2
    if4    = icps * 9.92
    if5    = icps * 14.15
    if6    = icps * 6.23
    goto nextPart

    modes2:
    if1     = icps * 1            ;uniform wood bar
    if2     = icps * 2.572
    if3     = icps * 4.644
    if4     = icps * 6.984
    if5     = icps * 9.723
    if6     = icps * 12.0
    goto nextPart

    nextPart:

    ; convert frequency to radian frequency

    itheta1 = (if1/idenom) * ipi
    itheta2 = (if2/idenom) * ipi
    itheta3 = (if3/idenom) * ipi
    itheta4 = (if4/idenom) * ipi
    itheta5 = (if5/idenom) * ipi
    itheta6 = (if6/idenom) * ipi

    ; calculate coefficients

    ib11 = -2 * irpole1 * cos(itheta1)
```

```
    ib21 = irpole1 * irpole1
    ib12 = -2 * irpole2 * cos(itheta2)
    ib22 = irpole2 * irpole2
    ib13 = -2 * irpole3 * cos(itheta3)
    ib23 = irpole3 * irpole3
    ib14 = -2 * irpole4 * cos(itheta4)
    ib24 = irpole4 * irpole4
    ib15 = -2 * irpole5 * cos(itheta5)
    ib25 = irpole5 * irpole5
    ib16 = -2 * irpole6 * cos(itheta6)
    ib26 = irpole6 * irpole6

    ;printk 1, ib 11
    ;printk 1, ib 21

    ;  also try setting the -1 coeff. to 0, but be sure to scale down the amplitude!

    asin1     biquad  apulse * iamp1, 1, 0, -1, 1, ib11, ib21
         asin2       biquad  apulse * iamp2, 1, 0, -1, 1, ib12, ib22
         asin3       biquad  apulse * iamp3, 1, 0, -1, 1, ib13, ib23
         asin4       biquad  apulse * iamp4, 1, 0, -1, 1, ib14, ib24
         asin5       biquad  apulse * iamp5, 1, 0, -1, 1, ib15, ib25
         asin6       biquad  apulse * iamp6, 1, 0, -1, 1, ib16, ib26


    afin    =    (asin1 + asin2 + asin3 + asin4 + asin5 + asin6)

    outs        afin * sqrt(p6), afin*sqrt(1-p6)

endin
</CsInstruments>
<CsScore>
;ins      st    dur   pulseSpd    pch     pan     amp     Modes
i1         0    12     0         7.089    0       0.7     2
i1         .    .      .         7.09     1       .       .
i1         .    .      .         7.091    0.5     .       .

i1         0    12     0         8.039    0       0.7     2
i1         0    12     0         8.04     1       0.7     2
i1         0    12     0         8.041    0.5     0.7     2

i1         9    .      .         7.089    0       .       2
i1         .    .      .         7.09     1       .       .
i1         .    .      .         7.091    0.5     .       .

i1         9    12     0         8.019    0       0.7     2
i1         9    12     0         8.02     1       0.7     2
i1         9    12     0         8.021    0.5     0.7     2
e
</CsScore>
</CsoundSynthesizer>
```

See Also

biquada, *moogvcf*, *rezzy*

Credits

Author: Hans Mikelson
October 1998

New in Csound version 3.49

biquada

biquada — A sweepable general purpose biquadratic digital filter with a-rate parameters.

Description

A sweepable general purpose biquadratic digital filter.

Syntax

```
ares biquada asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]
```

Initialization

iskip (optional, default=0) -- if non-zero, intialization will be skipped. Default value 0. (New in Csound version 3.50)

Performance

asig -- input signal

biquada is a general purpose biquadratic digital filter of the form:

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

This filter has the following frequency response:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

This type of filter is often encountered in digital signal processing literature. It allows six user-defined a-rate coefficients.

See Also

biquad

Credits

Author: Hans Mikelson
October 1998

New in Csound version 3.49

birnd

birnd — Retourne un nombre aléatoire dans un intervalle bipolaire.

Description

Retourne un nombre aléatoire dans un intervalle bipolaire.

Syntaxe

```
birnd(x) (taux-i ou -k seulement)
```

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

Exécution

Retourne un nombre aléatoire dans l'intervalle bipolaire allant de -*x* à *x*. *rnd* et *birnd* obtiennent leurs valeurs d'un générateur de nombres pseudo-aléatoires global, puis les mettent à l'échelle de l'intervalle demandé. Le générateur global unique distribuera ainsi sa séquence à ces unités durant toute l'exécution, quelque soit l'ordre d'arrivée de ces demandes.

Exemples

Voici un exemple de l'opcode birnd. Il utilise le fichier *birnd.csd* [examples/birnd.csd].

Exemple 63. Exemple de l'opcode birnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o birnd.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number from -1 to 1.
  i1 = birnd(1)
  print i1
endin


</CsInstruments>
```

```
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  i1 = 0.947
instr 1:  i1 = -0.721
```

Voir Aussi

rnd

Crédits

Auteur: Barry L. Vercoe
MIT
Cambridge, Massachussetts
1997

Etendu dans la version 3.47 au taux-x par John ffitch.

Exemple écrit par Kevin Conder.

bqrez

bqrez — A second-order multi-mode filter.

Description

A second-order multi-mode filter.

Syntax

```
ares bqrez asig, xfco, xres [, imode] [, iskip]
```

Initialization

imode (optional, default=0) -- The mode of the filter. Choose from one of the following:

- 0 = low-pass (default)
- 1 = high-pass
- 2 = band-pass
- 3 = band-reject
- 4 = all-pass

iskip (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

Performance

ares -- output audio signal.

asig -- input audio signal.

xfco -- filter cut-off frequency in Hz. May be i-time, k-rate, a-rate.

xres -- amount of resonance. Values of 1 to 100 are typical. Resonance should be one or greater. A value of 100 gives a 20dB gain at the cutoff frequency. May be i-time, k-rate, a-rate.

All filter modes can be frequency modulated as well as the resonance can also be frequency modulated.

bqrez is a resonant low-pass filter created using the Laplace s-domain equations for low-pass, high-pass, and band-pass filters normalized to a frequency. The bi-linear transform was used which contains a frequency transform constant from s-domain to z-domain to exactly match the frequencies together. Alot of trigonometric identities where used to simplify the calculation. It is very stable across the working frequency range up to the Nyquist frequency.

Examples

Here is an example of the bqrez opcode. It uses the file *bqrez.csd* [examples/bqrez.csd].

Exemple 64. Example of the bqrez opcode borrowed from the « rezzy » opcode in Kevin Conder's manual.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bqrez.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Gerassimof from example by Kevin Conder */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 16000, 220, 1

  ; Vary the filter-cutoff frequency from .2 to 2 KHz.
  kfco line 200, p3, 2000

  ; Set the resonance amount.
  kres init 0.99

  a1 bqrez asig, kfco, kres

  out a1
endin


</CsInstruments>
<CsScore>

/* Written by Matt Gerassimof from example by Kevin Conder */
; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

biquad, *moogvcf*, *rezzy*

Credits

Author: Matt Gerassimoff
New in version 4.32
Written in November 2002.

butbp

butbp — Same as the butterbp opcode.

Description

Same as the *butterbp* opcode.

Syntax

```
ares butbp asig, kfreq, kband [, iskip]
```

butbr

butbr — Same as the butterbr opcode.

Description

Same as the *butterbr* opcode.

Syntax

```
ares butbr asig, kfreq, kband [, iskip]
```

buthp

buthp — Same as the butterhp opcode.

Description

Same as the *butterhp* opcode.

Syntax

```
ares buthp asig, kfreq [, iskip]
```

butlp

butlp — Same as the butterlp opcode.

Description

Same as the *butterlp* opcode.

Syntax

```
ares butlp asig, kfreq [, iskip]
```

butterbp

butterbp — A band-pass Butterworth filter.

Description

Implementation of a second-order band-pass Butterworth filter. This opcode can also be written as *butbp*.

Syntax

```
ares butterbp asig, kfreq, kband [, iskip]
```

Initialization

iskip (optional, default=0) -- Skip initialization if present and non-zero.

Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

asig -- Input signal to be filtered.

kfreq -- Cutoff or center frequency for each of the filters.

kband -- Bandwidth of the bandpass and bandreject filters.

Examples

Here is an example of the butterbp opcode. It uses the file *butterbp.csd* [examples/butterbp.csd].

Exemple 65. Example of the butterbp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o butterbp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
```

```
  ; White noise signal
  asig rand 22050

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Filter it, passing only 1950 to 2050 Hz.
  abp butterbp asig, 2000, 100

  out abp
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

butterbr, *butterhp*, *butterlp*

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

Existed in 3.30

butterbr

butterbr — A band-reject Butterworth filter.

Description

Implementation of a second-order band-reject Butterworth filter. This opcode can also be written as *butbr*.

Syntax

```
ares butterbr asig, kfreq, kband [, iskip]
```

Initialization

iskip (optional, default=0) -- Skip initialization if present and non-zero.

Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

asig -- Input signal to be filtered.

kfreq -- Cutoff or center frequency for each of the filters.

kband -- Bandwidth of the bandpass and bandreject filters.

Examples

Here is an example of the butterbr opcode. It uses the file *butterbr.csd* [examples/butterbr.csd].

Exemple 66. Example of the butterbr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o butterbr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
```

```
  ; White noise signal
  asig rand 22050

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Filter it, cutting 2000 to 6000 Hz.
  abr butterbr asig, 4000, 2000

  out abr
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

butterbp, *butterhp*, *butterlp*

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

Existed in 3.30

butterhp

butterhp — A high-pass Butterworth filter.

Description

Implementation of second-order high-pass Butterworth filter. This opcode can also be written as *buthp*.

Syntax

```
ares butterhp asig, kfreq [, iskip]
```

Initialization

iskip (optional, default=0) -- Skip initialization if present and non-zero.

Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

asig -- Input signal to be filtered.

kfreq -- Cutoff or center frequency for each of the filters.

Examples

Here is an example of the butterhp opcode. It uses the file *butterhp.csd* [examples/butterhp.csd].

Exemple 67. Example of the butterhp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent
-odac           -idac     -d     ;;;realtime output
; For Non-realtime ouput leave only the line below:
; -o butterhp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
  ; White noise signal
  asig rand 22050

  out asig
```

```
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Filter it, passing frequencies above 250 Hz.
  ahp butterhp asig, 250

  out ahp
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

butterbp, *butterbr*, *butterlp*

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

Existed in 3.30

butterlp

butterlp — A low-pass Butterworth filter.

Description

Implementation of a second-order low-pass Butterworth filter. This opcode can also be written as *butlp*.

Syntax

```
ares butterlp asig, kfreq [, iskip]
```

Initialization

iskip (optional, default=0) -- Skip initialization if present and non-zero.

Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

asig -- Input signal to be filtered.

kfreq -- Cutoff or center frequency for each of the filters.

Examples

Here is an example of the butterlp opcode. It uses the file *butterlp.csd* [examples/butterlp.csd].

Exemple 68. Example of the butterlp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o butterlp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
  ; White noise signal
  asig rand 22050

  out asig
```

```
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Filter it, cutting frequencies above 1 KHz.
  alp butterlp asig, 1000

  out alp
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

butterbp, *butterbr*, *butterhp*

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

Existed in 3.30

button

button — Contrôles sur l'écran.

Description

Contrôles sur l'écran. Nécessite Winsound ou TCL/TK.

Syntaxe

```
kres button knum
```

Exécution

kres -- valeur du contrôle bouton. Si le bouton a été enfoncé depuis la dernière k-période, retourne 1, sinon 0.

knum -- le numéro du bouton. S'il n'existe pas, il apparaît sur l'écran à l'initialisation.

Voir Aussi

checkbox

Credits

Auteur : John ffitch
Université de Bath, Codemist. Ltd.
Bath, UK
Septembre 2000

Nouveau dans la version 4.08 de Csound

buzz

buzz — La sortie est un ensemble de partiels sinus en relation harmonique.

Description

La sortie est un ensemble de partiels sinus en relation harmonique.

Syntaxe

```
ares buzz xamp, xcps, knh, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table d'une fonction stockée contenant une onde sinus. Une grande table d'au moins 8192 points est recommandée.

iphs (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

Exécution

xamp -- amplitude

xcps -- fréquence en cycles par seconde

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

knh -- nombre total d'harmoniques demandés. Nouveau dans la version 3.57 de Csound, *knh* vaut un par défaut. Si *knh* est négatif, sa valeur absolue est utilisée.

buzz et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel *klh* = *kmul* = 1 ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire *knh* = int (sr / 2 / fréq. fondamentale), le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* durant l'exécution, sa valeur interne est nécessairement un entier ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. *buzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GEN11*, dans lequel le même ensemble de cosinus peut être stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul, le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

Exemples

Voici un exemple de l'opcode buzz. Il utilise le fichier *buzz.csd* [examples/buzz.csd].

Exemple 69. Exemple de l'opcode buzz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o buzz.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  knh = 3
  ifn = 1

  a1 buzz kamp, kcps, knh, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

gbuzz

Crédits

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre *kmul*.

Exemple écrit par Kevin Conder.

cabasa

cabasa — Modèle semi-physique d'un son de cabasa.

Description

cabasa est un modèle semi-physique d'un son de cabasa. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]
```

Initialisation

iamp -- Amplitude de la sortie. Note : commes ces instruments sont de type stochastique, ce n'est qu'une approximation.

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 512.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0.998 + (idamp * 0.002)

La valeur par défaut de *damping_amount* est 0,997 ce qui signifie que la valeur par défaut de *idamp* est -0,5. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

Exemples

Voici un exemple de l'opdcode cabasa. Il utilise le fichier *cabasa.csd* [examples/cabasa.csd].

Exemple 70. Exemple de l'opdcode cabasa.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o cabasa.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra ---------------

  sr =           44100
  kr =            4410
  ksmps =              10
  nchnls =               1

          instr 01                  ;an example of a cabasa
a1      cabasa p4, 0.01
          out a1
          endin


</CsInstruments>
<CsScore>

;score -------------------

   i1 0 1 26000
   e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

crunch, *sandpaper*, *sekere*, *stix*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

cauchy

cauchy — Générateur de nombres aléatoires de distribution de Cauchy.

Description

Générateur de nombres aléatoires de distribution de Cauchy. C'est un générateur de bruit de classe x.

Syntaxe

```
ares cauchy kalpha
```

```
ires cauchy kalpha
```

```
kres cauchy kalpha
```

Exécution

kalpha -- contrôle la dispersion centrée sur zéro (un grand *kalpha* = une grande dispersion). Donne en sortie des nombres positifs et négatifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode cauchy. Il utilise le fichier *cauchy.csd* [examples/cauchy.csd].

Exemple 71. Exemple de l'opcode cauchy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cauchy.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number, spread from 10.
  ; kalpha = 10

  i1 cauchy 10

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = -0.106
```

Voir Aussi

seed, *betarand*, *bexprnd*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Existait dans la 3.30

Exemple écrit par Kevin Conder.

ceil

ceil — Retourne le plus petit entier supérieur ou égal à *x*.

Description

Retourne le plus petit entier supérieur ou égal à *x*.

Syntaxe

```
ceil(x) (argument au taux d'initialisation, de contrôle ou audio)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Voir Aussi

abs, *exp*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Auteur : Istvan Varga
Nouveau dans Csound 5
2005

cent

cent — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

Syntaxe

```
cent(x)
```

Cette fonction travaille aux taux-i, -k et -a.

Initialisation

x -- une valeur exprimée en cents.

Exécution

La valeur retournée par la fonction *cent* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de cents spécifié.

Exemples

Voici un exemple de l'opcode cent. Il utilise le fichier *cent.csd* [examples/cent.csd].

Exemple 72. Exemple de l'opcode cent.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cent.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; The root note is A above middle-C (440 Hz)
  iroot = 440

  ; Raise the root note by 300 cents to C.
  icents = 300

  ; Calculate the new note.
```

```
  ifactor = cent(icents)
  inew = iroot * ifactor

  ; Print out of all of the values.
  print iroot
  print ifactor
  print inew
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1:  iroot = 440.000
instr 1:  ifactor = 1.189
instr 1:  inew = 523.229
```

Voir Aussi

db, *octave*, *semitone*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

cggoto

cggoto — Conditionally transfer control on every pass.

Description

Transfer control to *label* on every pass. (Combination of *cigoto* and *ckgoto*)

Syntax

```
cggoto condition, label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the cggoto opcode. It uses the file *cggoto.csd* [examples/cggoto.csd].

Exemple 73. Example of the cggoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cggoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = p4

  ; If i1 is equal to one, play a high note.
  ; Otherwise play a low note.
  cggoto (i1 == 1), highnote

lownote:
  a1 oscil 10000, 220, 1
  goto playit

highnote:
  a1 oscil 10000, 440, 1
  goto playit

playit:
  out a1
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play lownote for one second.
i 1 0 1 1
; Play highnote for one second.
i 1 0 1 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

cigoto, *ckgoto*, *cngoto*, *if*, *igoto*, *kgoto*, *tigoto*, *timout*

Credits

Added a note by Jim Aikin.

Example written by Kevin Conder.

chanctrl

chanctrl — Prend la valeur actuelle d'un contrôleur d'un canal MIDI.

Description

Prend la valeur actuelle d'un contrôleur et le configure optionnellement dans un intervalle spécifié.

Syntaxe

```
ival chanctrl ichnl, ictlno [, ilow] [, ihigh]
```

```
kval chanctrl ichnl, ictlno [, ilow] [, ihigh]
```

Initialisation

ichnl -- le canal MIDI (1-16).

ictlno -- le numéro du contrôleur MIDI (0-127).

ilow, *ihigh* -- Limites inférieure et supérieure de la configuration

Crédits

Auteur : Mike Berry
Collège Mills
Mai, 1997

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

changed

changed — k-rate signal change detector.

Description

This opcode outputs a trigger signal that informs when any one of its k-rate arguments has changed. Useful with valuator widgets or MIDI controllers.

Syntax

```
ktrig changed kvar1 [, kvar2,..., kvarN]
```

Performance

ktrig - Outputs a value of 1 when any of the k-rate signals has changed, otherwise outputs 0.

kvar1 [, kvar2,..., kvarN] - k-rate variables to watch for changes.

Examples

Here is an example of the changed opcode. It uses the file *changed.csd* [examples/changed.csd].

Exemple 74. Example of the changed opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =        44100
kr      =        100
ksmps   =        441
nchnls  =        2

       instr    1
ksig oscil 2,0.5,1
kint = int(ksig)
ktrig changed kint
printk 0.2, kint
printk2 ktrig
        endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 20


</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example written by Andrés Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

chani

chani — Reads data from the software bus

Description

Reads data from a channel of the inward software bus.

Syntax

```
kval chani kchan
```

```
aval chani kchan
```

Initialization

Performance

kchan -- a positive integer that indicates which channel of the software bus to read

Note that the inward and outward software busses are independent, and are not mixer buses. The last value remains until a new value is written. There is no imposed limit to the number of busses but they use memory so small numbers are to be preferred.

Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values

```
sr = 44100
ksmps = 100
nchnls = 1


instr   1
   kc   chani      1
   a1   oscil      p4, p5, 100
   a2   lowpass2  a1, kc, 200
        out        a2
endin
```

Credits

Author: John ffitch
2005

New in Csound5.00

chano

chano — Send data to the outwards software bus

Description

Send data to a channel of the outward software bus.

Syntax

```
chano kval, kchan
```

```
chano aval, kchan
```

Initialization

Performance

xval --- value to transmit

kchan -- a positive integer that indicates which channel of the software bus to write

Note that the inward and outward software busses are independent, and are not mixer buses. The last value remains until a new value is written. There is no imposed limit to the number of busses but they use memory so small numbers are to be preferred.

Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values

```
sr = 44100
ksmps = 100
nchnls = 1


instr   1
   a1   oscil     p4, p5, 100
        chano     1, a1
endin
```

Credits

Author: John ffitch
2005

New in Csound5.00

chebyshevpoly

chebyshevpoly — Efficiently evaluates the sum of Chebyshev polynomials of arbitrary order.

Description

The *chebyshevpoly* opcode calculates the value of a polynomial expression with a single a-rate input variable that is made up of a linear combination of the first N Chebyshev polynomials of the first kind. Each Chebyshev polynomial, Tn(x), is weighted by a k-rate coefficient, *kn*, so that the opcode is calculating a sum of any number of terms in the form *kn*Tn(x)*. Thus, the *chebyshevpoly* opcode allows for the waveshaping of an audio signal with a *dynamic* transfer function that gives precise control over the harmonic content of the output.

Syntax

```
aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]
```

Performance

ain -- the input signal used as the independent variable of the Chebyshev polynomials ("x").

aout -- the output signal ("y").

k0, k1, k2, ... -- k-rate multipliers for each Chebyshev polynomial.

This opcode is very useful for dynamic waveshaping of an audio signal. Traditional waveshaping techniques utilize a lookup table for the transfer function -- usually a sum of Chebyshev polynomials. When a sine wave at full-scale amplitude is used as an index to read the table, the precise harmonic spectrum as defined by the weights of the Chebyshev polynomials is produced. A dynamic spectrum is acheived by varying the amplitude of the input sine wave, but this produces a non-linear change in the spectrum.

By directly calculating the Chebyshev polynomials, the *chebyshevpoly* opcode allows more control over the spectrum and the number of harmonic partials added to the input can be varied with time. The value of each *kn* coefficient directly controls the amplitude of the nth harmonic partial if the input *ain* is a sine wave with amplitude = 1.0. This makes *chebyshevpoly* an efficient additive synthesis engine for N partials that requires only one oscillator instead of N oscillators. The amplitude or waveform of the input signal can also be changed for different waveshaping effects.

If we consider the input parameter *ain* to be "x" and the output *aout* to be "y", then the *chebyshevpoly* opcode calculates the following equation:

y = k0*T0(x) + k1*T1(x) + k2*T2(x) + k3*T3(x) + ...

where the Tn(x) are defined by the recurrence relation

T0(x) = 1,
T1(x) = x,
Tn(x) = 2x*T[n-1](x) - T[n-2](x)

More information about Chebyshev polynomials can be found on Wikipedia at *http://en.wikipedia.org/wiki/Chebyshev_polynomial* [http://en.wikipedia.org/wiki/Chebyshev_polynomial]

See Also

polynomial, *mac maca*

Examples

Here is an example of the chebyshevpoly opcode. It uses the file *chebyshevpoly.csd* [examples/chebyshevpoly.csd].

Exemple 75. Example of the chebyshevpoly opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441

; time-varying mixture of first six harmonics
instr 1
        ; According to the GEN13 manual entry,
        ; the pattern + - - + + - - for the signs of
        ; the chebyshev coefficients has nice properties.

        ; these six lines control the relative powers of the harmonics
        k1          line           1.0, p3, 0.0
        k2          line           -0.5, p3, 0.0
        k3          line           -0.333, p3, -1.0
        k4          line           0.0, p3, 0.5
        k5          line           0.0, p3, 0.7
        k6          line           0.0, p3, -1.0

        ; play the sine wave at a frequency of 256 Hz
        ax          oscili         1.0, 256, 1

        ; waveshape it
        ay          chebyshevpoly  ax, 0, k1, k2, k3, k4, k5, k6

        ; avoid clicks, scale final amplitude, and output
        adeclick    linseg         0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0
                    out            ay * adeclick * 10000
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1 ; a sine wave

i1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

checkbox

checkbox — Sense on-screen controls.

Description

Sense on-screen controls. Requires Winsound or TCL/TK.

Syntax

```
kres checkbox knum
```

Performance

kres -- value of the checkbox control. If the checkbox is set (pushed) then return 1, if not, return 0.

knum -- the number of the checkbox. If it does not exist, it is made on-screen at initialization.

Examples

Here is a simple example of the checkbox opcode. It uses the file *checkbox.csd* [examples/checkbox.csd].

Exemple 76. Simple example of the checkbox opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o checkbox.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
  ; Get the value from the checkbox.
  k1 checkbox 1

  ; If the checkbox is selected then k2=440, otherwise k2=880.
  k2 = (k1 == 0 ? 440 : 880)

  a1 oscil 10000, k2, 1
  out a1
endin


</CsInstruments>
<CsScore>
```

```
; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See Also

button

Credits

Author: John ffitch
University of Bath, Codemist. Ltd.
Bath, UK
September, 2000

Example written by Kevin Conder.

New in Csound version 4.08

chn

chn — Declare a channel of the named software bus.

Description

Declare a channel of the named software bus, with setting optional parameters in the case of a control channel. If the channel does not exist yet, it is created, with an inital value of zero or empty string. Otherwise, the type (control, audio, or string) of the existing channel must match the declaration, or an init error occurs. The input/output mode of an existing channel is updated so that it becomes the bitwise OR of the previous and the newly specified value.

Syntax

```
chn_k Sname, imode[, itype, idflt, imin, imax]
```

```
chn_a Sname, imode
```

```
chn_S Sname, imode
```

Initialization

imode -- sum of at least one of 1 for input and 2 for output.

itype (optional, defaults to 0) -- channel subtype for control channels only. Possible values are:

- 0: default/unspecified (idflt, imin, and imax are ignored)
- 1: integer values only
- 2: linear scale
- 3: exponential scale

idflt (optional, defaults to 0) -- default value, for control channels with non-zero itype only. Must be greater than or equal to imin, and less than or equal to imax.

imin (optional, defaults to 0) -- minimum value, for control channels with non-zero itype only. Must be non-zero for exponential scale (itype = 3).

imax (optional, defaults to 0) -- maximum value, for control channels with non-zero itype only. Must be greater than imin. In the case of exponential scale, it should also match the sign of imin.

Notes

The channel parameters (imode, itype, idflt, imin, and imax) are only hints for the host application or external software accessing the bus through the API, and do not actually restrict reading from or writing to the channel in any way. Also, the initial value of a newly created control channel is zero, regardless of the setting of idflt.

For communication with external software, using chnexport may be preferred, as it allows direct access

to orchestra variables exported as channels of the bus, eliminating the need for using chnset and chnget to send or receive data.

Performance

chn_k, **chn_a**, and **chn_S** declare a control, audio, or string channel, respectively.

Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values.

```
sr = 44100
ksmps = 100
nchnls = 1

chn_k "cutoff", 1, 3, 1000, 500, 2000

instr   1
   kc   chnget    "cutoff"
   a1   oscil     p4, p5, 100
   a2   lowpass2  a1, kc, 200
        out       a2
endin
```

Credits

Author: Istvan Varga
2005

chnclear

chnclear — Clears an audio output channel of the named software bus.

Description

Clears an audio channel of the named software bus to zero. Implies declaring the channel with imode=2 (see also chn_a).

Syntax

```
chnclear Sname
```

Initialization

Sname -- a string that indicates which named channel of the software bus to write.

Credits

Author: Istvan Varga
2006

chnexport

chnexport — Export a global variable as a channel of the bus.

Description

Export a global variable as a channel of the bus; the channel should not already exist, otherwise an init error occurs. This opcode is normally called from the orchestra header, and allows the host application to read or write orchestra variables directly, without having to use chnget or chnset to copy data.

Syntax

```
gival chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gaval chnexport Sname, imode
```

```
gSval chnexport Sname, imode
```

Initialization

imode -- sum of at least one of 1 for input and 2 for output.

itype (optional, defaults to 0) -- channel subtype for control channels only. Possible values are:

- 0: default/unspecified (idflt, imin, and imax are ignored)
- 1: integer values only
- 2: linear scale
- 3: exponential scale

idflt (optional, defaults to 0) -- default value, for control channels with non-zero itype only. Must be greater than or equal to imin, and less than or equal to imax.

imin (optional, defaults to 0) -- minimum value, for control channels with non-zero itype only. Must be non-zero for exponential scale (itype = 3).

imax (optional, defaults to 0) -- maximum value, for control channels with non-zero itype only. Must be greater than imin. In the case of exponential scale, it should also match the sign of imin.

Notes

The channel parameters (imode, itype, idflt, imin, and imax) are only hints for the host application or external software accessing the bus through the API, and do not actually restrict reading from or writing to the channel in any way.

While the global variable is used as output argument, chnexport does not actually change it, and always runs at i-time only. If the variable is not previously declared, it is created by Csound with an initial value

of zero or empty string.

Performance

Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values.

```
sr = 44100
ksmps = 100
nchnls = 1

gkc init 1000   ; set default value
gkc chnexport "cutoff", 1, 3, i(gkc), 500, 2000

instr   1
   a1   oscil     p4, p5, 100
   a2   lowpass2  a1, gkc, 200
        out       a2
endin
```

Credits

Author: Istvan Varga
2005

chnget

chnget — Reads data from the software bus.

Description

Reads data from a channel of the inward named software bus. Implies declaring the channel with imode=1 (see also chn_k, chn_a, and chn_S).

Syntax

```
ival chnget Sname
```

```
kval chnget Sname
```

```
aval chnget Sname
```

```
Sval chnget Sname
```

Initialization

Sname -- a string that identifies a channel of the named software bus to read.

Performance

ival -- the control value read at i-time.

kval -- the control value read at performance time.

aval -- the audio signal read at performance time.

Sval -- the string value read at i-time.

Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values.

```
sr = 44100
ksmps = 100
nchnls = 1

instr   1
   kc   chnget    "cutoff"
   a1   oscil     p4, p5, 100
   a2   lowpass2  a1, kc, 200
        out       a2
endin
```

Credits

Author: Istvan Varga
2005

chnmix

chnmix — Writes audio data to the named software bus, mixing to the previous output.

Description

Adds an audio signal to a channel of the named software bus. Implies declaring the channel with imode=2 (see also chn_a).

Syntax

```
chnmix aval, Sname
```

Initialization

Sname -- a string that indicates which named channel of the software bus to write.

Performance

aval -- the audio signal to write at performance time.

Credits

Author: Istvan Varga
2006

chnparams

chnparams — Query parameters of a channel.

Description

Query parameters of a channel (if it does not exist, all returned values are zero).

Syntax

```
itype, imode, ictltype, idflt, imin, imax chnparams
```

Initialization

itype -- channel data type (1: control, 2: audio, 3: string)

imode -- sum of 1 for input and 2 for output

ictltype -- special parameter for control channel only; if not available, set to zero.

idflt -- special parameter for control channel only; if not available, set to zero.

imin -- special parameter for control channel only; if not available, set to zero.

imax -- special parameter for control channel only; if not available, set to zero.

Performance

Example

Credits

Author: Istvan Varga
2005

chnset

chnset — Writes data to the named software bus.

Description

Write to a channel of the named software bus. Implies declaring the channel with imode=2 (see also chn_k, chn_a, and chn_S).

Syntax

```
chnset ival, Sname
```

```
chnset kval, Sname
```

```
chnset aval, Sname
```

```
chnset Sval, Sname
```

Initialization

Sname -- a string that indicates which named channel of the software bus to write.

Performance

ival -- the control value to write at i-time.

kval -- the control value to write at performance time.

aval -- the audio signal to write at performance time.

Sval -- the string value to write at i-time.

Example

The example shows the software bus being used to write pitch information to a controlling program.

```
sr = 44100
ksmps = 100
nchnls = 1

instr   1
   a1    in
   kp,ka pitchamdf a1
         chnset   kp, "pitch"
endin
```

Credits

Author: Istvan Varga
2005

chuap

chuap — Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

Description

Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

Syntaxe

```
aI3, aV2, aV1 chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step
```

Initialisation

iI3 -- Courant initial dans G

iV2 -- Tension initiale aux bornes de C2

iV1 -- Tension initiale aux bornes de C1

Exécution

kL -- Inductance L

kR0 -- Résistance R0

kC1 -- Capacité C1

kG -- Résistance G

kGa -- Résistance V (terme non linéaire)

kGb -- Résistance V (terme non linéaire)

kGb -- Résistance V (terme non linéaire)

ktime_step -- Pas temporel de l'équation aux différences, permet de contrôler plus ou moins la hauteur.

L'oscillateur de Chua est un simple oscillateur RLC avec une résistance active. L'oscillateur peut être amené à une bifurcation de période, et ainsi vers le chaos, à cause de la réponse non linéaire de la résistance active.

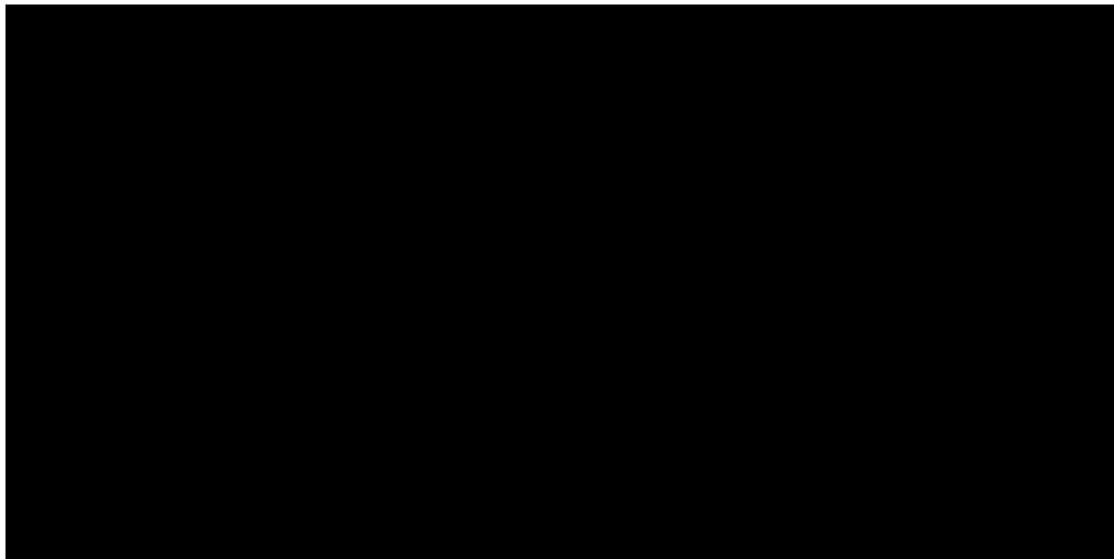


Diagramme du Circuit de l'Oscillateur de Chua

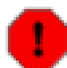
Le circuit est décrit par un ensemble de trois équations différentielles ordinaires appelées équations de Chua :

```
dI3      R0      1
--- =  - -- I3 - - V2
dt       L       L

dV2    1       G
--- = -- I3 - -- (V2 - V1)
dt    C2      C2

dV1    G              1
--- = -- (V2 - V1) - -- f(V1)
dt    C1             C1
```

où f() est une fonction dsicontinue par morceaux simulant la résistance active :

```
f(V1) = Gb V1 + - (Ga - Gb)(|V1 + E| - |V1 - E|)
```

Une solution (I3,V2,V1)(t) de ces équations partant d'un état initial (I3,V2,V1)(0) est appelée une trajectoire de l'oscillateur de Chua. L'implémentation dans Csound est une simulation de l'oscillateur de Chua par une équation aux différences avec intégration de Runge-Kutta.



Avertissement

Attention ! Certains jeux de paramètres produiront des pics d'amplitude ou une rétroaction positive pouvant endommager vos haut-parleurs.

Exemples

Voici un exemple de l'opcode chuap. Il utilise le fichier *chuap.csd* [examples/chuap.csd].

Exemple 77. Exemple de l'opcode chuap.

```
<CsoundSynthesizer>
<CsOptions>
csound -RWfo chuas_oscillator.wav
</CsOptions>
<CsInstruments>
sr              =           44100
ksmps           =           100
nchnls          =           2
0dbfs           =           10000

gibuzztable     ftgen       1, 0, 16384, 10, 1

                instr 1
                ; sys_variables = system_vars(5:12); % L,R0,C2,G,Ga,Gb,E,C1 or p8:p15
                ; integ_variables = [system_vars(14:16),system_vars(1:2)]; % x0,y0,z0,dataset_size,step
istep_size      =           p5
iL              =           p8
iR0             =           p9
iC2             =           p10
iG              =           p11
iGa             =           p12
iGb             =           p13
iE              =           p14
iC1             =           p15
iI3             =           p17
iV2             =           p18
iV1             =           p19
iattack         =           0.02
isustain        =           p3
irelease        =           0.02
p3              =           iattack + isustain + irelease
iscale          =           1.0
adamping        linseg      0.0, iattack, iscale, isustain, iscale, irelease, 0.0
aguide          buzz        5000, 440, sr/440, gibuzztable
aI3, aV2, aV1   chuap       iL, iR0, iC2, iG, iGa, iGb, iE, iC1, iI3, iV2, iV1, istep_size
asignal         balance     aV2, aguide
                outs        adamping * asignal, adamping * asignal
                endin
</CsInstruments>
<CsScore>
;         Adapted from ABC++ MATLAB example data.
i 1 0 20 1500 .1   -1 -1 -0.00707925 0.00001647 100  1 -.99955324 -1.00028375 1 -.00222159 204.8 -2.362
i 1 + 20 1500 .425  0 -1  1.3506168  0              -4.50746268737 -1 2.4924 .93 1 1 0 -22.28662665 .00
i 1 + 20 1024 .05  -1 -1  0.00667    0.000651    10 -1 .856 1.1 1 .06 51.2 -20.200590133667 .1725393235
i 1 + 20 1024 0.05 -1 -1 0.00667 0.000651 10 -1 0.856 1.1 1 0.1 153.6 21.12496758 0.03001749 0.51582866
</CsScore>
</CsoundSynthesizer>
```

Crédits

Inventeur de l'oscillateur de Chua : *Leon O. Chua* [http://www.eecs.berkeley.edu/~chua]
Auteur de la simulation dans MATLAB : James Patrick McEvoy *MATLAB Adventures in Bifurcations and Chaos (ABC++)* [http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3541]
Auteur du portage dans Csound : Michael Gogins

Nouveau dans la version 5.09 de Csound

cigoto

cigoto — Conditionally transfer control during the i-time pass.

Description

During the i-time pass only, unconditionally transfer control to the statement labeled by *label*.

Syntax

```
cigoto condition, label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the cigoto opcode. It uses the file *cigoto.csd* [examples/cigoto.csd].

Exemple 78. Example of the cigoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Get the value of the 4th p-field from the score.
  iparam = p4

  ; If iparam is 1 then play the high note.
  ; If not then play the low note.
  cigoto (iparam ==1), highnote
    igoto lownote

highnote:
  ifreq = 880
  goto playit

lownote:
  ifreq = 440
  goto playit

playit:
  ; Print the values of iparam and ifreq.
  print iparam
```

```
  print ifreq

  a1 oscil 10000, ifreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
instr 1:  iparam = 0.000
instr 1:  ifreq = 440.000
instr 1:  iparam = 1.000
instr 1:  ifreq = 880.000
```

See Also

cggoto, *ckgoto*, *cngoto*, *goto*, *if*, *kgoto*, *rigoto*, *tigoto*, *timout*

Credits

Added a note by Jim Aikin.

Example written by Kevin Conder.

ckgoto

ckgoto — Conditionally transfer control during the p-time passes.

Description

During the p-time passes only, unconditionally transfer control to the statement labeled by *label*.

Syntax

```
ckgoto condition, label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the ckgoto opcode. It uses the file *ckgoto.csd* [examples/ckgoto.csd].

Exemple 79. Example of the ckgoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ckgoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Change kval linearly from 0 to 2 over
  ; the period set by the third p-field.
  kval line 0, p3, 2

  ; If kval is greater than or equal to 1 then play the high note.
  ; If not then play the low note.
  ckgoto (kval >= 1), highnote
    kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit

playit:
  ; Print the values of kval and kfreq.
```

```
  printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

  a1 oscil 10000, kfreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

See Also

cggoto, *cigoto*, *cngoto*, *goto*, *if*, *igoto*, *tigoto*, *timout*

Credits

Added a note by Jim Aikin.

Example written by Kevin Conder.

clear

clear — Zeroes a list of audio signals.

Description

clear zeroes a list of audio signals.

Syntax

```
clear avar1 [, avar2] [, avar3] [...]
```

Performance

avar1, avar2, avar3, ... -- signals to be zeroed

clear sets every sample of each of the given audio signals to zero when it is performed. This is equivalent to writing *avarN = 0* in the orchestra for each of the specified variables. Typically, *clear* is used with global variables that combine multiple signals from different sources and change with each k-pass (performance loop) through all of the active instrument instances. After the final usage of such a variable and before the next k-pass, it is necessary to clear the variable so that it does not add the next cycle's signals to the previous result. *clear* is especially useful in combination with *vincr* (variable increment) and they are intended to be used together with file output opcodes such as *fout*.

Examples

See the *fout* opcode for an example.

See Also

vincr

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

clfilt

clfilt — Implements low-pass and high-pass filters of different styles.

Description

Implements the classical standard analog filter types: low-pass and high-pass. They are implemented with the four classical kinds of filters: Butterworth, Chebyshev Type I, Chebyshev Type II, and Elliptical. The number of poles may be any even number from 2 to 80.

Syntax

```
ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
```

Initialization

itype -- 0 for low-pass, 1 for high-pass.

inpol -- The number of poles in the filter. It must be an even number from 2 to 80.

ikind (optional) -- 0 for Butterworth, 1 for Chebyshev Type I, 2 for Chebyshev Type II, 3 for Elliptical. Defaults to 0 (Butterworth)

ipbr (optional) -- The pass-band ripple in dB. Must be greater than 0. It is ignored by Butterworth and Chebyshev Type II. The default is 1 dB.

isba (optional) -- The stop-band attenuation in dB. Must be less than 0. It is ignored by Butterworth and Chebyshev Type I. The default is -60 dB.

iskip (optional) -- 0 initializes all filter internal states to 0. 1 skips initialization. The default is 0.

Performance

asig -- The input audio signal.

kfreq -- The corner frequency for low-pass or high-pass.

Examples

Here is an example of the clfilt opcode as a low-pass filter. It uses the file *clfilt_lowpass.csd* [examples/clfilt_lowpass.csd].

Exemple 80. Example of the clfilt opcode as a low-pass filter.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
```

```
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o clfilt_lowpass.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
  ; White noise signal
  asig rand 22050

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Lowpass filter signal asig with a
  ; 10-pole Butterworth at 500 Hz.
  a1 clfilt asig, 500, 0, 10

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Here is an example of the clfilt opcode as a high-pass filter. It uses the file *clfilt_highpass.csd* [examples/clfilt_highpass.csd].

Exemple 81. Example of the clfilt opcode as a high-pass filter.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o clfilt_highpass.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
```

```
  ; White noise signal
  asig rand 22050

  out asig
endin


; Instrument #2 - a filtered noise waveform.
instr 2
  ; White noise signal
  asig rand 22050

  ; Highpass filter signal asig with a 6-pole Chebyshev
  ; Type I at 20 Hz with 3 dB of passband ripple.
  a1 clfilt asig, 20, 1, 6, 1, 3

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Erik Spjut

New in version 4.20

clip

clip — Clips a signal to a predefined limit.

Description

Clips an a-rate signal to a predefined limit, in a « soft » manner, using one of three methods.

Syntax

```
ares clip asig, imeth, ilimit [, iarg]
```

Initialization

imeth -- selects the clipping method. The default is 0. The methods are:

- 0 = Bram de Jong method (default)
- 1 = sine clipping
- 2 = tanh clipping

ilimit -- limiting value

iarg (optional, default=0.5) -- when *imeth* = 0, indicates the point at which clipping starts, in the range 0 - 1. Not used when *imeth* = 1 or *imeth* = 2. Default is 0.5.

Performance

asig -- a-rate input signal

The Bram de Jong method (*imeth* = 0) applies the algorithm:

```
|x| > a:     f(x) = sin(x) * (a+(x-a)/(1+((x-a)/(1-a))^2 |x| > 1: f(x) = sin(x) * (a+1)/2
```

This method requires that *asig* be normalized to 1.

The second method (*imeth* = 1) is the sine clip:

```
|x| < limit: f(x) = limit * sin(#*x/(2*limit)) f(x) = limit * sin(x)
```

The third method (imeth = 3) is the tanh clip:

|*x*| < *limit*: f(*x*) = *limit* * tanh(*x*/*limit*)/tanh(1) f(*x*) = *limit* * sin(*x*)

Note

Method 1 appears to be non-functional at release of Csound version 4.07.

Examples

Here is an example of the clip opcode. It uses the file *clip.csd* [examples/clip.csd].

Exemple 82. Example of the clip opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o clip.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a noisy waveform.
  arnd rand 44100
  ; Clip the noisy waveform's amplitude to 20,000
  a1 clip arnd, 2, 20000

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch

University of Bath, Codemist Ltd.
Bath, UK
August, 2000

New in Csound version 4.07

clock

clock — Obsolète.

Description

Obsolète. Utiliser plutôt l'opcode *rtclock*.

clockoff

clockoff — Stops one of a number of internal clocks.

Description

Stops one of a number of internal clocks.

Syntax

```
clockoff inum
```

Initialization

inum -- the number of a clock. There are 32 clocks numbered 0 through 31. All other values are mapped to clock number 32.

Performance

Between a *clockon* and a *clockoff* opcode, the CPU time used is accumulated in the clock. The precision is machine dependent but is the millisecond range on UNIX and Windows systems. The *readclock* opcode reads the current value of a clock at initialization time.

Examples

See the *readclock* opcode for an example.

See Also

clockon, *readclock*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
July, 1999

New in Csound version 3.56

clockon

clockon — Starts one of a number of internal clocks.

Description

Starts one of a number of internal clocks.

Syntax

```
clockon inum
```

Initialization

inum -- the number of a clock. There are 32 clocks numbered 0 through 31. All other values are mapped to clock number 32.

Performance

Between a *clockon* and a *clockoff* opcode, the CPU time used is accumulated in the clock. The precision is machine dependent but is the millisecond range on UNIX and Windows systems. The *readclock* opcode reads the current value of a clock at initialization time.

Examples

See the *readclock* opcode for an example.

See Also

clockoff, *readclock*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
July, 1999

New in Csound version 3.56

cngoto

cngoto — Transfers control on every pass when a condition is not true.

Description

Transfers control on every pass when the condition is *not* true.

Syntax

```
cngoto condition, label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the cngoto opcode. It uses the file *cngoto.csd* [examples/cngoto.csd].

Exemple 83. Example of the cngoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cngoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Change kval linearly from 0 to 2 over
  ; the period set by the third p-field.
  kval line 0, p3, 2

  ; If kval *is not* greater than or equal to 1 then play
  ; the high note. Otherwise, play the low note.
  cngoto (kval >= 1), highnote
    kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit

playit:
  ; Print the values of kval and kfreq.
```

```
  printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

  a1 oscil 10000, kfreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
kval = 0.000000, kfreq = 880.000000
kval = 0.999732, kfreq = 880.000000
kval = 1.999639, kfreq = 440.000000
```

See Also

cggoto, *cigoto*, *ckgoto*, *goto*, *if*, *igoto*, *tigoto*, *timout*

Credits

Example written by Kevin Conder.

New in version 4.21

comb

comb — Reverberates an input signal with a « colored » frequency response.

Description

Reverberates an input signal with a « colored » frequency response.

Syntax

```
ares comb asig, krvt, ilpt [, iskip] [, insmps]
```

Initialization

ilpt -- loop time in seconds, which determines the « echo density » of the reverberation. This in turn characterizes the « color » of the *comb* filter whose frequency response curve will contain *ilpt* * *sr*/2 peaks spaced evenly between 0 and *sr*/2 (the Nyquist frequency). Loop time can be as large as available memory will permit. The space required for an *n* second loop is 4*n*sr* bytes. Delay space is allocated and returned as in *delay*.

iskip (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

insmps (optional, default=0) -- delay amount, as a number of samples.

Performance

krvt -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Output from a comb filter will appear only after *ilpt* seconds.

Examples

Here is an example of the comb opcode. It uses the file *comb.csd* [examples/comb.csd].

Exemple 84. Example of the comb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o comb.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the audio mixer.
gamix init 0

; Instrument #1.
instr 1
  ; Generate a source signal.
  a1 oscili 30000, cpspch(p4), 1
  ; Output the direct sound.
  out a1

  ; Add the source signal to the audio mixer.
  gamix = gamix + a1
endin

; Instrument #99 (highest instr number executed last)
instr 99
  krvt = 1.5
  ilpt = 0.1

  ; Comb-filter the mixed signal.
  a99 comb gamix, krvt, ilpt
  ; Output the result.
  out a99

  ; Empty the mixer for the next pass.
  gamix = 0
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=7.00
i 1 0 0.1 7.00
; Play Instrument #1 for a tenth of a second, p4=7.02
i 1 1 0.1 7.02
; Play Instrument #1 for a tenth of a second, p4=7.04
i 1 2 0.1 7.04
; Play Instrument #1 for a tenth of a second, p4=7.06
i 1 3 0.1 7.06

; Make sure the comb-filter remains active.
i 99 0 5
e


</CsScore>
</CsoundSynthesizer>
```

See Also

alpass, *reverb*, *valpass*, *vcomb*

Credits

Author: William « Pete » Moss (*vcomb* and *valpass*)
University of Texas at Austin
Austin, Texas USA
January 2002

Example written by Kevin Conder.

compress

compress — Compress, limit, expand, duck or gate an audio signal.

Description

This unit functions as an audio compressor, limiter, expander, or noise gate, using either soft-knee or hard-knee mapping, and with dynamically variable performance characteristics. It takes two audio input signals, aasig and acsig, the first of which is modified by a running analysis of the second. Both signals can be the same, or the first can be modified by a different controlling signal.

compress first examines the controlling acsig by performing envelope detection. This is directed by two control values katt and krel, defining the attack and release time constants (in seconds) of the detector. The detector rides the peaks (not the RMS) of the control signal. Typical values are .01 and .1, the latter usually being similar to ilook.

The running envelope is next converted to decibels, then passed through a mapping function to determine what compresser action (if any) should be taken. The mapping function is defined by four decibel control values. These are given as positive values, where 0 db corresponds to an amplitude of 1, and 90 db corresponds to an amplitude of 32768.

Syntax

```
ar compress aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook
```

Initialization

ilook -- lookahead time in seconds, by which an internal envelope release can sense what is coming. This induces a delay between input and output, but a small amount of lookahead improves the performance of the envelope detector. Typical value is .05 seconds, sufficient to sense the peaks of the lowest frequency in acsig.

Performance

kthresh -- sets the lowest decibel level that will be allowed through. Normally 0 or less, but if higher the threshold will begin removing low-level signal energy such as background noise.

kloknee, *khiknee* -- decibel break-points denoting where compression or expansion will begin. These set the boundaries of a soft-knee curve joining the low-amplitude 1:1 line and the higher-amplitude compression ratio line. Typical values are 48 and 60 db. If the two breakpoints are equal, a hard-knee (angled) map will result.

kratio -- ratio of compression when the signal level is above the knee. The value 2 will advance the output just one decibel for every input gain of two; 3 will advance just one in three; 20 just one in twenty, etc. Inverse ratios will cause signal expansion: .5 gives two for one, .25 four for one, etc. The value 1 will result in no change.

The actions of compress will depend on the parameter settings given. A hard-knee compressor-limiter, for instance, is obtained from a near-zero attack time, equal-value break-points, and a very high ratio (say 100). A noise-gate plus expander is obtained from some positive threshold, and a fractional ratio above the knee. A voice-activated music compressor (ducker) will result from feeding the music into aasig and the speech into acsig. A voice de-esser will result from feeding the voice into both, with the acsig version being preceded by a band-pass filter that emphasizes the sibilants. Each application will re-

quire some experimentation to find the best parameter settings; these have been made k-variable to make this practical.

Examples

```
aout compress amus, avoc, 0, 40, 60, 3, .1, .5, .02 ; voice-activated compressor
                                                   ; with low-level sensitivity
```

Credits

Written by Barry L. Vercoe for Extended Csound and released in csound5.02.

control

control — Configurable slider controls for realtime user input.

Description

Configurable slider controls for realtime user input. Requires Winsound or TCL/TK. *control* reads a slider's value.

Syntax

```
kres control knum
```

Performance

knum -- number of the slider to be read.

Calling *control* will create a new slider on the screen. There is no theoretical limit to the number of sliders. Windows and TCL/TK use only integers for slider values, so the values may need rescaling. GUIs usually pass values at a fairly slow rate, so it may be advisable to pass the output of control through *port*.

Examples

See the *setctrl* opcode for an example.

See Also

setctrl

Credits

Author: John ffitch
University of Bath, Codemist. Ltd.
Bath, UK
May, 2000

New in Csound version 4.06

convle

convle — Same as the convolve opcode.

Description

Same as the *convolve* opcode.

convolve

convolve — Convolves a signal and an impulse response.

Description

Output is the convolution of signal *ain* and the impulse response contained in *ifilcod*. If more than one output signal is supplied, each will be convolved with the same impulse response. Note that it is considerably more efficient to use one instance of the operator when processing a mono input to create stereo, or quad, outputs.

Note: this opcode can also be written as *convle*.

Syntax

```
ar1 [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]
```

Initialization

ifilcod -- integer or character-string denoting an impulse response data file. An integer denotes the suffix of a file *convolve.m*; a character string (in double quotes) gives a filename, optionally a full pathname. If not a fullpath, the file is sought first in the current directory, then in the one given by the environment variable SADIR (if defined). The data file contains the Fourier transform of an impulse response. Memory usage depends on the size of the data file, which is read and held entirely in memory during computation, but which is shared by multiple calls.

ichannel (optional) -- which channel to use from the impulse response data file.

Performance

ain -- input audio signal.

convolve implements Fast Convolution. The output of this operator is delayed with respect to the input. The following formulas should be used to calculate the delay:

```
For (1/kr) <= IRdur:
        Delay = ceil(IRdur * kr) / kr
For (1/kr)  IRdur:
        Delay = IRdur * ceil(1/(kr*IRdur))
Where:
        kr  = Csound control rate
        IRdur = duration, in seconds, of impulse response
        ceil(n) = smallest integer not smaller than n
```

One should be careful to also take into account the initial delay, if any, of the impulse response. For example, if an impulse response is created from a recording, the soundfile may not have the initial delay included. Thus, one should either ensure that the soundfile has the correct amount of zero padding at the start, or, preferably, compensate for this delay in the orchestra. (the latter method is more efficient). To compensate for the delay in the orchestra, subtract the initial delay from the result calculated using the above formula(s), when calculating the required delay to introduce into the 'dry' audio path.

For typical applications, such as reverb, the delay will be in the order of 0.5 to 1.5 seconds, or even longer. This renders the current implementation unsuitable for real time applications. It could conceivably be used for real time filtering however, if the number of taps is small enough.

The author intends to create a higher-level operator at some stage, that would mix the wet & dry signals, using the correct amount of delay automatically.

Examples

Create frequency domain impulse response file using the *cvanal utility*:

```
csound -Ucvanal l1_44.wav l1_44.cv
```

Determine duration of impulse response. For high accuracy, determine the number of sample frames in the impulse response soundfile, and then compute the duration with:

duration = (sample frames)/(sample rate of soundfile)

This is due to the fact that the *sndinfo utility* only reports the duration to the nearest 10ms. If you have a utility that reports the duration to the required accuracy, then you can simply use the reported value directly.

```
sndinfo l1_44.wav
```

length = 60822 samples, sample rate = 44100

Duration = 60822/44100 = 1.379s.

Determine initial delay, if any, of impulse response. If the impulse response has not had the initial delay removed, then you can skip this step. If it has been removed, then the only way you will know the initial delay is if the information has been provided separately. For this example, let's assume that the initial delay is 60ms. (0.06s)

Determine the required delay to apply to the dry signal, to align it with the convolved signal:

```
If kr = 441:
     1/kr = 0.0023, which is <= IRdur (1.379s), so:
     Delay1  = ceil(IRdur * kr) / kr
             = ceil(608.14) / 441
             = 609/441
             = 1.38s
```

```
Accounting for the initial delay:
     Delay2  = 0.06s
Total delay   = delay1 - delay2
```

= 1.38 - 0.06
= 1.32s

Create .orc file, e.g.:

```
 ; Simple demonstration of CONVOLVE operator, to apply reverb.
        sr = 44100
        kr = 441
        ksmps = 100
        nchnls = 2
        instr   1
imix = 0.22 ; Wet/dry mix. Vary as desired.
            ; NB: 'Small' reverbs often require a much higher
            ; percentage of wet signal to sound interesting. 'Large'
            ; reverbs seem require less. Experiment! The wet/dry mix is
            ; very important - a small change can make a large difference.
ivol = 0.9  ; Overall volume level of reverb. May need to adjust
            ; when wet/dry mix is changed, to avoid clipping.
idel = 1.32 ; Required delay to align dry audio with output of convolve.
            ; This can be automatically calculated within the orc file,
            ; if desired.
adry            soundin "anechoic.wav"      ; input (dry) audio
awet1,awet2     convolve adry,"l1_44.cv"    ; stereo convolved (wet) audio
adrydel         delay   (1-imix)*adry,idel  ; Delay dry signal, to align it with
                                           ; convolved signal. Apply level
                                           ; adjustment here too.
                outs    ivol*(adrydel+imix*awet1),ivol*(adrydel+imix*awet2)
                                           ; Mix wet & dry signals, and output
        endin
```

See also

pconvolve, *dconv*, *cvanal*.

Credits

Author: Greg Sullivan

1996

New in version 3.28

cos

cos — Calcule une fonction cosinus.

Description

Retourne cosinus de *x* (*x* en radians).

Syntaxe

```
cos(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode cos. Il utilise le fichier *cos.csd* [examples/cos.csd].

Exemple 85. Exemple de l'opcode cos.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cos.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = cos(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.991
```

Voir Aussi

cosh, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv*

Crédits

Exemple écrit par Kevin Conder.

cosh

cosh — Calcule une fonction cosinus hyperbolique.

Description

Retourne cosinus hyperbolique de *x* (*x* en radians).

Syntaxe

```
cosh(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode cosh. Il utilise le fichier *cosh.csd* [examples/cosh.csd].

Exemple 86. Exemple de l'opcode cosh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = cosh(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.543
```

Voir Aussi

cos, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

cosinv

cosinv — Calcule une fonction arccosinus.

Description

Retourne arccosinus de *x* (*x* en radians).

Syntaxe

```
cosinv(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode cosinv. Il utilise le fichier *cosinv.csd* [examples/cosinv.csd].

Exemple 87. Exemple de l'opcode cosinv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosinv.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = cosinv(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.047
```

Voir Aussi

cos, *cosh*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

cps2pch

cps2pch — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

Syntaxe

```
icps cps2pch ipch, iequal
```

Initialisation

ipch -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

iequal -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'octave. Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.

Note

1. Les lignes suivantes sont équivalentes

```
ia  =  cpspch(8.02)
ib     cps2pch  8.02, 12
ic     cpsxpch  8.02, 12, 2, 1.02197503906
```

2. C'est un opcode, pas une fonction.

3. Des valeurs négatives pour *ipch* sont permises.

Exemples

Voici un exemple de l'opcode cps2pch. Il utilise le fichier *cps2pch.csd* [examples/cps2pch.csd].

Exemple 88. Exemple de l'opcode cps2pch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o cps2pch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a normal twelve-tone scale.
  ipch = 8.02
  iequal = 12

  icps cps2pch ipch, iequal

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 293.666
```

Voici un exemple de l'opcode cps2pch qui utilise une table de multiplicateurs de fréquence. Il utilise le fichier *cps2pch_ftable.csd* [examples/cps2pch_ftable.csd].

Exemple 89. Exemple de l'opcode cps2pch qui utilise une table de multiplicateurs de fréquence.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cps2pch_ftable.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ipch = 8.02

  ; Use Table #1, a table of frequency multipliers.
  icps cps2pch ipch, -1

  print icps
endin


</CsInstruments>
<CsScore>
```

```
; Table #1: a table of frequency multipliers.
; Creates a 10-note scale of unequal divisions.
f 1 0 16 -2 1 1.1 1.2 1.3 1.4 1.6 1.7 1.8 1.9

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 313.951
```

Voici un exemple de l'opcode cps2pch qui utilise une échelle tempérée égale avec 19 divisions de l'octave. Il utilise le fichier *cps2pch_19et.csd* [examples/cps2pch_19et.csd].

Exemple 90. Exemple de l'opcode cps2pch qui utilise une échelle tempérée égale avec 19 divisions de l'octave.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cps2pch_19et.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use 19ET scale.
  ipch = 8.02
  iequal = 19

  icps cps2pch ipch, iequal

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 281.429
```

Voir Aussi

cpspch, *cpsxpch*

Crédits

Auteur : John ffitch
Université de Bath/Codemist Ltd.
Bath, UK
1997

Nouveau dans la version 3.492 de Csound

cpsmidi

cpsmidi — Get the note number of the current MIDI event, expressed in cycles-per-second.

Description

Get the note number of the current MIDI event, expressed in cycles-per-second.

Syntax

```
icps cpsmidi
```

Performance

Get the note number of the current MIDI event, expressed in cycles-per-second units, for local processing.

cpsmidi vs. cpsmidinn

The *cpsmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *cpsmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *cpsmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *cpsmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

Examples

Here is an example of the cpsmidi opcode. It uses the file *cpsmidi.csd* [examples/cpsmidi.csd].

Exemple 91. Example of the cpsmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpsmidi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  i1 cpsmidi

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidib*, *cpstmid*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

cpsmidib

cpsmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in cycles-per-second.

Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in cycles-per-second.

Syntax

```
icps cpsmidib [irange]
```

```
kcps cpsmidib [irange]
```

Initialization

irange (optional) -- the pitch bend range in semitones.

Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in cycles-per-second units. Available as an i-time value or as a continuous k-rate value.

Examples

Here is an example of the cpsmidib opcode. It uses the file *cpsmidib.csd* [examples/cpsmidib.csd].

Exemple 92. Example of the cpsmidib opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpsmidib.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 cpsmidib

  print i1
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidi*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

cpsmidinn

cpsmidinn — Convertit un numéro de note Midi en cycles par seconde.

Description

Convertit un numéro de note Midi en cycles par seconde.

Syntaxe

```
cpsmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

cpsmidinn est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de fréquence équivalente en cycles par seconde (Hertz). Cette conversion suppose que le do médian est la note Midi numéro 60 et que le la du diapason est accordé à 440 Hz. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.

cpsmidinn vs. cpsmidi

L'opcode *cpsmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un évènement Midi, un évènement de partition, un évènement en ligne, ou depuis un autre instrument. La valeur d'entrée de *cpsmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *cpsmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *cpsmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

cpsmidinn et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 7. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode cpsmidinn. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

Exemple 93. Exemple de l'opcode cpsmidinn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
  ; i-time loop to print conversion table
  imidiNN =   0
  loop1:
    icps   = cpsmidinn(imidiNN)
    ioct   = octmidinn(imidiNN)
    ipch   = pchmidinn(imidiNN)

    print   imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
  if (imidiNN < 128) igoto loop1
```

```
endin

instr 2
  ; test k-rate converters
  kMiddleC  =   60
  kcps  = cpsmidinn(kMiddleC)
  koct  = octmidinn(kMiddleC)
  kpch  = pchmidinn(kMiddleC)

  printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

octmidinn, *pchmidinn*, *cpsmidi*, *notnum*, *cpspch*, *cpsoct*, *octcps*, *octpch*, *pchoct*

Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

cpsoct

cpsoct — Convertit une valeur octave-point-partie-décimale en cycles par seconde.

Description

Convertit une valeur octave-point-partie-décimale en cycles par seconde.

Syntaxe

```
cpsoct (oct)  (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression.

Exécution

cpsoct et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 8. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici une exemple de l'opcode cpsoct. Il utilise le fichier *cpsoct.csd* [examples/cpsoct.csd].

Exemple 94. Exemple de l'opcode cpsoct.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsoct.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Convert an octave-point-decimal value into a
  ; cycles-per-second value.
  ioct = 8.75
  icps = cpsoct(ioct)

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 440.000
```

Voir Aussi

cpspch, *octcps*, *octpch*, *pchoct*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Crédits

Exemple écrit par Kevin Conder.

cpspch

cpspch — Convertit une valeur de classe de hauteur en cycles par seconde.

Description

Convertit une valeur de classe de hauteur en cycles par seconde.

Syntaxe

```
cpspch (pch)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

cpsoct et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 9. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode cpspch. Il utilise le fichier *cpspch.csd* [examples/cpspch.csd].

Exemple 95. Exemple de l'opcode cpspch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpspch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Convert a pitch-class value into a
  ; cycles-per-second value.
  ipch = 8.09
  icps = cpspch(ipch)

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 440.000
```

Voir Aussi

cps2pch, *cpsoct*, *cpsxpch*, *octcps*, *octpch*, *pchoct*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Crédits

Exemple écrit par Kevin Conder.

cpstmid

cpstmid — Get a MIDI note number (allows customized micro-tuning scales).

Description

This unit is similar to *cpsmidi*, but allows fully customized micro-tuning scales.

Syntax

```
icps cpstmid ifn
```

Initialization

ifn -- function table containing the parameters (*numgrades*, *interval*, *basefreq*, *basekeymidi*) and the tuning ratios.

Performance

Init-rate only

cpsmid requires five parameters, the first, *ifn*, is the function table number of the tuning ratios, and the other parameters must be stored in the function table itself. The function table *ifn* should be generated by *GEN02*, with normalization inhibited. The first four values stored in this function are:

1. *numgrades* -- the number of grades of the micro-tuning scale
2. *interval* -- the frequency range covered before repeating the grade ratios, for example 2 for one octave, 1.5 for a fifth etc.
3. *basefreq* -- the base frequency of the scale in Hz
4. *basekeymidi* -- the MIDI note number to which *basefreq* is assigned unmodified

After these four values, the user can begin to insert the tuning ratios. For example, for a standard 12 note scale with the base frequency of 261 Hz assigned to the key number 60, the corresponding f-statement in the score to generate the table should be:

```
;         numgrades interval  basefreq basekeymidi tuning ratios (equal temp)
f1 0 64 -2   12       2       261       60        1  1.059463094359  1.122462048309  1.189207115003 ..etc...
```

Another example with a 24 note scale with a base frequency of 440 assigned to the key number 48, and a repetition interval of 1.5:

```
;          numgrades interval  basefreq basekeymidi tuning-ratios (equal temp)
f1 0 64 -2   24       1.5      440       48        1   1.01  1.02  1.03   ..etc...
```

Examples

Here is an example of the cpstmid opcode. It uses the file *cpstmid.csd* [examples/cpstmid.csd].

Exemple 96. Example of the cpstmid opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpstmid.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
             1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
             1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
  ; Use Table #1.
  ifn = 1
  i1 cpstmid ifn

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

See Also

cpsmidi, *GEN02*

Credits

Author: Gabriel Maldonado
Italy
1998

Example written by Kevin Conder.

New in Csound version 3.492

cpstun

cpstun — Retourne des valeurs d'échelle microtonale au taux-k.

Description

Retourne des valeurs d'échelle microtonale au taux-k.

Syntaxe

```
kcps cpstun ktrig, kindex, kfn
```

Exécution

kcps -- Valeur de retour en cycles par seconde.

ktrig -- Un signal utilisé pour déclencher l'évaluation.

kindex -- Un nombre entier servant d'indice dans l'échelle.

kfn -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

cpstun travaille au taux-k. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *kindex* reçoit des nombres entiers indiquant quel degré de l'échelle donnée doit être converti en Hz. Dans *cpstun*, une nouvelle valeur ne sera évaluée que lorsque *ktrig* contiendra une valeur non nulle. La table de fonction *kfn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;           numgrades    basefreq     tuning-ratios (eq.temp) .......
;                  interval    basekey
f1 0 64 -2  12      2      261    60     1    1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```
;                  numgrades       basefreq      tuning-ratios .......
;                          interval       basekey
f1 0 64 -2         24      1.5      440     48      1    1.01   1.02   1.03    ..etc...
```

Exemples

Voici un exemple de l'opcode cpstun. Il utilise le fichier *cpstun.csd* [examples/cpstun.csd].

Exemple 97. Exemple de l'opcode cpstun.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstun.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
             1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
             1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
  ; Set the trigger.
  ktrig init 1

  ; Use Table #1.
  kfn init 1

  ; If the base key (note #60) is C, then 9 notes
  ; above it (note #60 + 9 = note #69) should be A.
  kindex init 69

  k1 cpstun ktrig, kindex, kfn

  printk2 k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
i1   440.11044
```

Voir Aussi

cpstmid, *cpstuni*, *GEN02*

Crédits

Exemple écrit par Kevin Conder.

cpstuni

cpstuni — Retourne des valeurs d'échelle microtonale au taux-i.

Description

Retourne des valeurs d'échelle microtonale au taux-i.

Syntaxe

```
icps cpstuni index, ifn
```

Initialisation

icps -- Valeur de retour en cycles par seconde.

index -- Un nombre entier servant d'indice dans l'échelle.

ifn -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

Exécution

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

cpstuni travaille au taux-i. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *index* reçoit un nombre entier indiquant quel degré de l'échelle donnée doit être converti en Hz. La table de fonction *ifn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;           numgrades    basefreq      tuning-ratios (eq.temp) .......
;                  interval    basekey
f1 0 64 -2  12      2      261    60      1   1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```
;                  numgrades       basefreq      tuning-ratios .......
;                          interval       basekey
f1 0 64 -2         24      1.5     440    48     1   1.01  1.02  1.03   ..etc...
```

Exemples

Voici un exemple de l'opcode cpstuni. Il utilise le fichier *cpstuni.csd* [examples/cpstuni.csd].

Exemple 98. Exemple de l'opcode cpstuni.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstuni.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
             1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
             1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
  ; Use Table #1.
  ifn = 1

  ; If the base key (note #60) is C, then 9 notes
  ; above it (note #60 + 9 = note #69) should be A.
  index = 69

  i1 cpstuni index, ifn

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 440.110
```

Voir Aussi

cpstmid, *cpstun*, *GEN02*

Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Kevin Conder.

cpsxpch

cpsxpch — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle.

Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle. Le nombre de divisions ne doit pas dépasser 100.

Syntaxe

```
icps cpsxpch ipch, iequal, irepeat, ibase
```

Initialisation

ipch -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

iequal -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'« octave ». Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.

irepeat -- Nombre indiquant l'intervalle qui est l'« octave ». Le nombre 2 est utilisé pour des divisions de l'octave, 3 pour une douzième, 4 pour deux octaves, ainsi de suite. Ce nombre ne doit pas forcément être un entier, mais il doit être positif.

ibase -- La fréquence qui correspond à la hauteur 0.0

Note

1. Les lignes suivantes sont équivalentes

   ```
   ia  =  cpspch(8.02)
   ib     cps2pch  8.02, 12
   ic     cpsxpch  8.02, 12, 2, 1.02197503906
   ```

2. C'est un opcode, pas une fonction.

3. Des valeurs négatives sont permises pour *ipch*, mais pas pour *irepeat*, ni pour *iequal* ou *ibase*.

Exemples

Voici un exemple de l'opcode cpsxpch. Il utilise le fichier *cpsxpch.csd* [examples/cpsxpch.csd].

Exemple 99. Exemple de l'opcode cpsxpch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a normal twelve-tone scale.
  ipch = 8.02
  iequal = 12
  irepeat = 2
  ibase = 1.02197503906

  icps cpsxpch ipch, iequal, irepeat, ibase

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 293.666
```

Voici un exemple de l'ocpode cpsxpch qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave. Il utilise le fichier *cpsxpch_105et.csd* [examples/cpsxpch_105et.csd].

Exemple 100. Exemple de l'ocpode cpsxpch qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_105et.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
  ; Use a 10.5ET scale.
  ipch = 4.02
  iequal = 21
  irepeat = 4
  ibase = 16.35160062496

  icps cpsxpch ipch, iequal, irepeat, ibase

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 4776.824
```

Voici un exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian. Il utilise le fichier *cpsxpch_pierce.csd* [examples/cpsxpch_pierce.csd].

Exemple 101. Exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_pierce.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a Pierce scale centered on middle A.
  ipch = 2.02
  iequal = 12
  irepeat = 3
  ibase = 261.62561

  icps cpsxpch ipch, iequal, irepeat, ibase

  print icps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 2827.762
```

Voir Aussi

cpspch, *cps2pch*

Crédits

Auteur : John ffitch
Université de Bath/Codemist Ltd.
Bath, UK
1997

Nouveau dans la version 3.492 de Csound

cpuprc

cpuprc — Control allocation of cpu resources on a per-instrument basis, to optimize realtime output.

Description

Control allocation of cpu resources on a per-instrument basis, to optimize realtime output.

Syntax

```
cpuprc insnum, ipercent
```

Initialization

insnum -- instrument number

ipercent -- percent of cpu processing-time to assign. Can also be expressed as a fractional value.

Performance

cpuprc sets the cpu processing-time percent usage of an instrument, in order to avoid buffer underrun in realtime performances, enabling a sort of polyphony theshold. The user must set *ipercent* value for each instrument to be activated in realtime. Assuming that the total theoretical processing time of the cpu of the computer is 100%, this percent value can only be defined empirically, because there are too many factors that contribute to limiting realtime polyphony in different computers.

For example, if *ipercent* is set to 5% for instrument 1, the maximum number of voices that can be allocated in realtime, is 20 (5% * 20 = 100%). If the user attempts to play a further note while the 20 previous notes are still playing, Csound inhibits the allocation of that note and will display the following warning message:

```
can't allocate last note because it exceeds 100% of cpu time
```

In order to avoid audio buffer underruns, it is suggested to set the maximum number of voices slightly lower than the real processing power of the computer. Sometimes an instrument can require more processing time than normal. If, for example, the instrument contains an oscillator which reads a table that doesn't fit in cache memory, it will be slower than normal. In addition, any program running concurrently in multitasking, can subtract processing power to varying degrees.

At the start, all instruments are set to a default value of *ipercent* = 0.0% (i.e. zero processing time or rather infinite cpu processing-speed). This setting is OK for deferred-time sessions.

All instances of *cpuprc* must be defined in the header section, not in the instrument body.

Examples

Here is an example of the cpuprc opcode. It uses the file *cpuprc.csd* [examples/cpuprc.csd].

Exemple 102. Example of the cpuprc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpuprc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Limit Instrument #1 to 5% of the CPU processing time.
cpuprc 1, 5

; Instrument #1
instr 1
  a1 oscil 10000, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

See Also

maxalloc, *prealloc*

Credits

Author: Gabriel Maldonado
Italy
July, 1999

Example written by Kevin Conder.

New in Csound version 3.57

cross2

cross2 — Cross synthesis using FFT's.

Description

This is an implementation of cross synthesis using FFT's.

Syntax

```
ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias
```

Initialization

isize -- This is the size of the FFT to be performed. The larger the size the better the frequency response but a sloppy time response.

ioverlap -- This is the overlap factor of the FFT's, must be a power of two. The best settings are 2 and 4. A big overlap takes a long time to compile.

iwin -- This is the function table that contains the window to be used in the analysis. One can use the *GEN20* routine to create this window.

Performance

ain1 -- The stimulus sound. Must have high frequencies for best results.

ain2 -- The modulating sound. Must have a moving frequency response (like speech) for best results.

kbias -- The amount of cross synthesis. 1 is the normal, 0 is no cross synthesis.

Examples

Here is an example of the cross2 opcode. It uses the file *cross2.csd* [examples/cross2.csd] and *beats.wav* [examples/beats.wav].

Exemple 103. Example of the cross2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cross2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Play an audio file.
instr 1
  ; Use the "beats.wav" audio file.
  aout soundin "beats.wav"
  out aout
endin

; Instrument #2 - Cross-synthesize!
instr 2
  ; Use the "ahh" sound stored in Table #1.
  ain1 loscil 30000, 1, 1, 1
  ; Use the "beats.wav" audio file.
  ain2 soundin "beats.wav"

  isize = 4096
  ioverlap = 2
  iwin = 2
  kbias init 1

  aout cross2 ain1, ain2, isize, ioverlap, iwin, kbias

  out aout
endin


</CsInstruments>
<CsScore>

; Table #1: An audio file.
f 1 0 128 1 "ahh.aiff" 0 4 0
; Table #2: A windowing function.
f 2 0 2048 20 2

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Paris Smaragdis
MIT, Cambridge
1997

Example written by Kevin Conder.

crunch

crunch — Modèle semi-physique d'un son de craquement.

Description

crunch est un modèle semi-physique d'un son de craquement. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntax

```
ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]
```

Initialisation

iamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 7.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,998 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,99806 ce qui signifie que la valeur par défaut de *idamp* est 0,03. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

Exemples

Voici un exemple de l'opcode crunch. Il utilise le fichier *crunch.csd* [examples/crunch.csd].

Exemple 104. Exemple de l'opcode crunch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o crunch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra ---------------

  sr =           44100
  kr =            4410
  ksmps =             10
  nchnls =              1

instr 01                  ;an example of a crunch
a1      crunch p4, 0.01
          out a1
          endin


</CsInstruments>
<CsScore>

;score -------------------

   i1 0 1 26000
   e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cabasa, *sandpaper*, *sekere*, *stix*

Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

ctrl14

ctrl14 — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest ctrl14 ichan, ictlno1, ictlno2, imin, imax [, ifn]
```

```
kdest ctrl14 ichan, ictlno1, ictlno2, kmin, kmax [, ifn]
```

Initialisation

idest -- signal de sortie

ichan -- numéro de canal MIDI (1-16)

ictln1o -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids faible (0-127)

imin -- la valeur décimale minimale de sortie définie par l'utilisateur

imax -- la valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

Exécution

kdest -- signal de sortie

kmin -- la valeur décimale minimale de sortie définie par l'utilisateur

kmax -- la valeur décimale maximale de sortie définie par l'utilisateur

ctrl14 (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table. Il nécessite deux contrôleurs MIDI en entrée.

ctrl14 est différent de *midic14* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl14*.

Voir aussi

ctrl7, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

ctrl21

ctrl21 — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest ctrl21 ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
```

```
kdest ctrl21 ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

Initialisation

idest -- signal de sortie

ichan -- numéro de canal MIDI (1-16)

ictlno1 -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids moyen (0-127)

ictlno3 -- numéro de contrôleur pour l'octet de poids faible (0-127)

imin -- la valeur décimale minimale de sortie définie par l'utilisateur

imax -- la valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

Exécution

kdest -- signal de sortie

kmin -- la valeur décimale minimale de sortie définie par l'utilisateur

kmax -- la valeur décimale maximale de sortie définie par l'utilisateur

ctrl21 (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser une indexation de table facultative. Il nécessite trois contrôleurs MIDI en entrée.

ctrl21 est différent de *midic21* parce qu'il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl21*.

Voir aussi

ctrl7, *ctrl14*, *initc7*, *initc14*, *initc21*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

ctrl7

ctrl7 — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest ctrl7 ichan, ictlno, imin, imax [, ifn]
```

```
kdest ctrl7 ichan, ictlno, kmin, kmax [, ifn]
```

```
adest ctrl7 ichan, ictlno, kmin, kmax [, ifn] [, icutoff]
```

Initialisation

idest -- signal de sortie

ichan -- canal MIDI (1-16)

ictlno -- numéro du contrôleur MIDI (0-127)

imin -- valeur décimale minimale de sortie définie par l'utilisateur

imax -- valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *iminx* et *imax*.

icutoff (facultatif) -- fréquence de coupure du filtre passe-bas pour lisser la sortie au taux-a.

Exécution

kdest, adest -- signal de sortie

kmin -- valeur décimale minimale de sortie définie par l'utilisateur

kmax -- valeur décimale maximale de sortie définie par l'utilisateur

ctrl7 (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Les valeurs minimale et maximale peuvent varier au taux-k.

ctrl7 diffère de *midic7* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a aussi besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur.

La version de *ctrl7* au taux-a fournit en sortie une variable de taux-a, qui est passée par un filtre passe-bas (lissée). Il y a un paramètre facultatif *icutoff*, pour établir la fréquence de coupure du filtre passe-bas.

Sa valeur par défaut est 5.

Voir aussi

ctrl14, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

La version de *ctrl7* au taux-a a été ajoutée dans la version 5.06

ctrlinit

ctrlinit — Initialise les valeurs pour un groupe de contrôleurs MIDI.

Description

Initialise les valeurs pour un groupe de contrôleurs MIDI.

Syntaxe

```
ctrlinit ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \
      [, ival3] [,...ival32]
```

Initialisation

ichnl -- numéro de canal MIDI (1-16)

ictlno1, *ictlno1*, etc. -- numéros de contrôleurs MIDI (0-127)

ival1, *ival2*, etc. -- valeur initiale pour le contrôleur MIDI de numéro correspondant

Exécution

Initialise les valeurs pour un groupe de contrôleurs MIDI.

Voir aussi

massign

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

cuserrnd

cuserrnd — Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

Description

Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

Syntaxe

```
aout cuserrnd kmin, kmax, ktableNum
```

```
iout cuserrnd imin, imax, itableNum
```

```
kout cuserrnd kmin, kmax, ktableNum
```

Initialisation

imin -- limite inférieure de l'intervalle

imax -- limite supérieure de l'intervalle

itableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

Exécution

ktableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

kmin -- limite inférieure de l'intervalle

kmax -- limite supérieure de l'intervalle

cuserrnd (Continuous USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire continue créée par l'utilisateur. Dans ce cas la forme de l'histogramme de la distribution peut être dessinée ou générée par n'importe quelle routine GEN. La table contenant la forme d'un tel histogramme doit être transformée ensuite en une fonction de distribution au moyen de GEN40 (voir GEN40 pour plus de détails). Cette fonction doit ensuite être allouée à l'argument *XtableNum* de *cuserrnd*. L'intervalle de sortie sera ensuite mis à l'échelle selon les arguments *Xmin* et *Xmax*. *cuserrnd* faisant une interpolation linéaire entre les éléments de la table, il n'est pas recommandé pour les distributions discrètes (GEN41 et GEN42).

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Voir Aussi

duserrnd, *urd*

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

dam

dam — A dynamic compressor/expander.

Description

This opcode dynamically modifies a gain value applied to the input sound *ain* by comparing its power level to a given threshold level. The signal will be compressed/expanded with different factors regarding that it is over or under the threshold.

Syntax

```
ares dam asig, kthreshold, icomp1, icomp2, irtime, iftime
```

Initialization

icomp1 -- compression ratio for upper zone.

icomp2 -- compression ratio for lower zone

irtime -- gain rise time in seconds. Time over which the gain factor is allowed to raise of one unit.

iftime -- gain fall time in seconds. Time over which the gain factor is allowed to decrease of one unit.

Performance

asig -- input signal to be modified

kthreshold -- level of input signal which acts as the threshold. Can be changed at k-time (e.g. for ducking)

Note on the compression factors: A compression ratio of one leaves the sound unchanged. Setting the ratio to a value smaller than one will compress the signal (reduce its volume) while setting the ratio to a value greater than one will expand the signal (augment its volume).

Examples

Because the results of the *dam* opcode can be subtle, I recommend looking at them in a graphical audio editor program like *audacity*. *audacity* is available for Linux, Windows, and the MacOS and may be downloaded from *http://audacity.sourceforge.net* [http://audacity.sourceforge.net/].

Here is an example of the dam opcode. It uses the file *dam.csd* [examples/dam.csd], and *beats.wav* [examples/beats.wav].

Exemple 105. An example of the dam opcode compressing an audio signal.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
;-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dam.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1, uncompressed signal.
instr 1
  ; Use the "beats.wav" audio file.
  asig soundin "beats.wav"

  out asig
endin

; Instrument #2, compressed signal.
instr 2
  ; Use the "beats.wav" audio file.
  asig soundin "beats.wav"

  ; Compress the audio signal.
  kthreshold = 25000
  icomp1 = 0.5
  icomp2 = 0.763
  irtime = 0.1
  iftime = 0.1
  a1 dam asig, kthreshold, icomp1, icomp2, irtime, iftime

  out a1
endin

; Instrument #3, compressed signal.
instr 3
  ; Use the "beats.wav" audio file.
  asig soundin "beats.wav"

  ; Compress the audio signal.
  kthreshold line 25000, p3, 4410000
  icomp1 = 0.5
  icomp2 = 0.763
  irtime = 0.1
  iftime = 0.1
  a1 dam asig, kthreshold, icomp1, icomp2, irtime, iftime

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
; Play Instrument #3 for 2 seconds.
i 3 4 2
e


</CsScore>
</CsoundSynthesizer>
```

This example compresses the audio file « beats.wav ». You should hear a drum pattern repeat twice. The second time, the sound should be quieter (compressed) than the first.

Here is another example of the dam opcode. It uses the file *dam_expanded.csd* [examples/dam_expanded.csd], and *mary.wav* [examples/mary.wav].

Exemple 106. An example of the dam opcode expanding an audio signal.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dam_expanded.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1, normal audio signal.
instr 1
  ; Use the "mary.wav" audio file.
  asig soundin "mary.wav"

  out asig
endin

; Instrument #2, expanded audio signal.
instr 2
  ; Use the "mary.wav" audio file.
  asig soundin "mary.wav"

  ; Expand the audio signal.
  kthreshold init 7500
  icomp1 = 2.25
  icomp2 = 2.25
  irtime = 0.1
  iftime = 0.6
  a1 dam asig, kthreshold, icomp1, icomp2, irtime, iftime

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1.
i 1 0.0 3.5
; Play Instrument #2.
i 2 3.5 3.5
e


</CsScore>
</CsoundSynthesizer>
```

This example expands the audio file « mary.wav ». You should hear a melody repeat twice. The second time, the sound should be louder (expanded) than the first.

Credits

Author: Marc Resibois
Belgium
1997

New in version 3.47

Examples written by Kevin Conder.

date

date — Returns the number seconds since 1 January 1970.

Description

Returns the number seconds since 1 January 1970, using the operating system's clock.

Syntax

```
ir date
```

Initialization

ir -- value at i-time, of the system clock in seconds since the start of the epoch.

Examples

Here is an example of the date opcode. It uses the file *date.csd* [examples/date.csd].

Exemple 107. Example of the date opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o date.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
      ii date
      print ii
      Sa dates ii
      prints Sa
      Ss dates -1
      prints Ss
      St dates 1
      prints St
endin

</CsInstruments>
<CsScore>
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
instr 1:  ii = 1165665152.000
Sat Dec  9 11:52:32 2006
Sat Dec  9 11:51:46 2006
Thu Jan  1 01:00:01 1970
```

See Also

dates

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
December 2006

New in Csound version 5.05

dates

dates — Returns as a string the date and time specified.

Description

Returns as a string the date and time specified.

Syntax

```
Sir dates [ itime]
```

Initialization

itime -- the time is seconds since teh start of the epoch. If omited or negative the current time is taken.

Sir -- the date and time as a sting.

Examples

Here is an example of the dates opcode. It uses the file *date.csd* [examples/date.csd].

Exemple 108. Example of the dates opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o date.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
      ii date
      print ii
      Sa dates ii
      prints Sa
      Ss dates -1
      prints Ss
      St dates 1
      prints St
endin

</CsInstruments>
<CsScore>
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
instr 1:  ii = 1165665152.000
Sat Dec  9 11:52:32 2006
Sat Dec  9 11:51:46 2006
Thu Jan  1 01:00:01 1970
```

See Also

date

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
December 2006

New in Csound version 5.05

db

db — Retourne l'amplitude équivalente pour une valeur donnée en décibels.

Description

Retourne l'amplitude équivalente pour une valeur donnée en décibels. Cet opcode est le même que *ampdb*.

Syntaxe

```
db(x)
```

Cette fonction fonctionne aux taux-i, -k et -a.

Initialisation

x -- une valeur exprimée en décibels.

Exécution

Retourne l'amplitude équivalente pour une valeur donnée en décibels.

Exemples

Voici un exemple de l'opcode db. Il utilise le fichier *db.csd* [examples/db.csd].

Exemple 109. Example of the db opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o db.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Calculate the amplitude of 40 decibels.
  idecibels = 40
  iamp = db(idecibels)

  print iamp
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 100.000
```

Voir Aussi

ampdb, *cent*, *octave*, *semitone*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

dbamp

dbamp — Retourne l'équivalent en décibel de l'amplitude *x*.

Description

Retourne l'équivalent en décibel de l'amplitude *x*.

Syntaxe

```
dbamp(x)  (arguments de taux-i ou -k seulement)
```

Exemples

Voici un exemple de l'opcode dbamp. Il utilise le fichier *dbamp.csd* [examples/dbamp.csd].

Exemple 110. Exemple de l'opcode dbamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dbamp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 30000
  idb = dbamp(iamp)

  print idb
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  idb = 89.542
```

Voir Aussi

ampdb, *ampdbfs*, *dbfsamp*

Crédits

Exemple écrit par Kevin Conder.

dbfsamp

dbfsamp — Retourne l'équivalent en décibel de l'amplitude *x*, relative à l'amplitude maximale.

Description

Retourne l'équivalent en décibel de l'amplitude *x*, relative à l'amplitude maximale. L'amplitude maximale est supposée être codée en 16 bit. Nouveau dans la version 4.10.

Syntaxe

```
dbfsamp(x)  (arguments de taux-i ou -k seulement)
```

Exemples

Voici un exemple de l'opcode dbfsamp. Il utilise le fichier *dbfsamp.csd* [examples/dbfsamp.csd].

Exemple 111. Exemple de l'opcode dbfsamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dbfsamp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 30000
  idb = dbfsamp(iamp)

  print idb
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  idb = -0.767
```

Voir Aussi

ampdb, *ampdbfs*, *dbamp*

Crédits

Exemple écrit par Kevin Conder.

dcblock

dcblock — A DC blocking filter.

Description

Implements the DC blocking filter

Y[i] = X[i] - X[i-1] + (igain * Y[i-1])

Based on work by Perry Cook.

Syntax

```
ares dcblock ain [, igain]
```

Initialization

igain -- the gain of the filter, which defaults to 0.99

Performance

ain -- audio signal input

Note

The new *dcblock2* opcode is an improved method of removing DC from an audio signal.

Examples

Here is an example of the dcblock opcode. It uses the file *dcblock.csd* [examples/dcblock.csd], and *beats.wav* [examples/beats.wav].

Exemple 112. Example of the dcblock opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dcblock.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- normal audio signal.
instr 1
  asig soundin "beats.wav"
  out asig
endin

; Instrument #2 -- dcblock-ed audio signal.
instr 2
  asig soundin "beats.wav"

  igain = 0.75
  a1 dcblock asig, igain

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See also

dcblock2

Credits

Author: John ffitch
University of Bath, Codemist Ltd.
Bath, UK

Example written by Kevin Conder.

New in Csound version 3.49

February 2003: Thanks to a note from Anders Andersson, corrected the formula.

dcblock2

dcblock2 — Un filtre bloqueur de composante continue.

Description

Implémente un filtre bloqueur de composante continue avec une atténuation améliorée de la composante continue.

Syntaxe

```
ares dcblock2 ain [, iorder] [, iskip]
```

Initialisation

iorder -- ordre du filtre, au minimum 4ème ordre, vaut par défaut 128.

iskip -- s'il vaut 1, l'initialisation est ignorée (0 par défaut).

Exécution

ares -- signal audio filtré

ain -- signal audio en entrée

Note

Avec l'utilisation d'une valeur inférieure à *ksmps* pour *iorder*, la réduction du décalage dû à la composante continue ne sera pas efficace.

Voir Aussi

dcblock

Crédits

Par Victor Lazzarini

Nouveau dans la version 5.09 de Csound

dconv

dconv — A direct convolution opcode.

Description

A direct convolution opcode.

Syntax

```
ares dconv asig, isize, ifn
```

Initialization

isize -- the size of the convolution buffer to use. if the buffer size is smaller than the size of ifn, then only the first isize values will be used from the table.

ifn -- table number of a stored function containing the impulse response for convolution.

Performance

Rather than the analysis/resynthesis method of the convolve opcode, *dconv* uses direct convolution to create the result. For small tables it can do this quite efficiently, however larger table require much more time to run. *dconv* does (isize * ksmps) multiplies on every k-cycle. Therefore, reverb and delay effects are best done with other opcodes (unless the times are short).

dconv was designed to be used with time varying tables to facilitate new realtime filtering capabilities.

Examples

Here is an example of the dconv opcode. It uses the file *dconv.csd* [examples/dconv.csd].

Exemple 113. Example of the dconv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dconv.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

#define RANDI(A) #kout  randi   1, kfq, $A*.001+iseed, 1
        tablew  kout, $A, itable#
```

```
instr 1
itable  init    1
iseed   init    .6
isize   init    ftlen(itable)
kfq     line    1, p3, 10

$RANDI(0)
$RANDI(1)
$RANDI(2)
$RANDI(3)
$RANDI(4)
$RANDI(5)
$RANDI(6)
$RANDI(7)
$RANDI(8)
$RANDI(9)
$RANDI(10)
$RANDI(11)
$RANDI(12)
$RANDI(13)
$RANDI(14)
$RANDI(15)

asig    rand    10000, .5, 1
asig    butlp   asig, 5000
asig    dconv   asig, isize, itable

        out     asig *.5
endin


</CsInstruments>
<CsScore>

f1 0 16 10 1
i1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See also

pconvolve, *convolve.*, *convolve*

Credits

Author: William « Pete » Moss
2001

New in version 4.12

delay

delay — Delays an input signal by some time interval.

Description

A signal can be read from or written into a delay path, or it can be automatically delayed by some time interval.

Syntax

```
ares delay asig, idlt [, iskip]
```

Initialization

idlt -- requested delay time in seconds. This can be as large as available memory will permit. The space required for n seconds of delay is 4n * *sr* bytes. It is allocated at the time the instrument is first initialized, and returned to the pool at the end of a score section.

iskip (optional, default=0) -- initial disposition of delay-loop data space (see *reson*). The default value is 0.

Performance

asig -- audio signal

delay is a composite of *delayr* and *delayw*, both reading from and writing into its own storage area. It can thus accomplish signal time-shift, although modified feedback is not possible. There is no minimum delay period.

Examples

Here is an example of the delay opcode. It uses the file *delay.csd* [examples/delay.csd].

Exemple 114. Example of the delay opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o delay.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
```

```
; Instrument #1 -- Delayed beeps.
instr 1
  ; Make a basic sound.
  abeep vco 20000, 440, 1

  ; Delay the beep by .1 seconds.
  idlt = 0.1
  adel delay abeep, idlt

  ; Send the beep to the left speaker and
  ; the delayed beep to the right speaker.
  outs abeep, adel
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Keep the score running for 2 seconds.
f 0 2

; Play Instrument #1.
i 1 0.0 0.2
i 1 0.5 0.2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

delay1, *delayr*, *delayw*

Credits

Example written by Kevin Conder.

delay1

delay1 — Delays an input signal by one sample.

Description

Delays an input signal by one sample.

Syntax

```
ares delay1 asig [, iskip]
```

Initialization

iskip (optional, default=0) -- initial disposition of delay-loop data space (see *reson*). The default value is 0.

Performance

delay1 is a special form of delay that serves to delay the audio signal *asig* by just one sample. It is thus functionally equivalent to the *delay* opcode but is more efficient in both time and space. This unit is particularly useful in the fabrication of generalized non-recursive filters.

See Also

delay, *delayr*, *delayw*

delayk

delayk — Delays an input signal by some time interval.

Description

k-rate delay opcodes

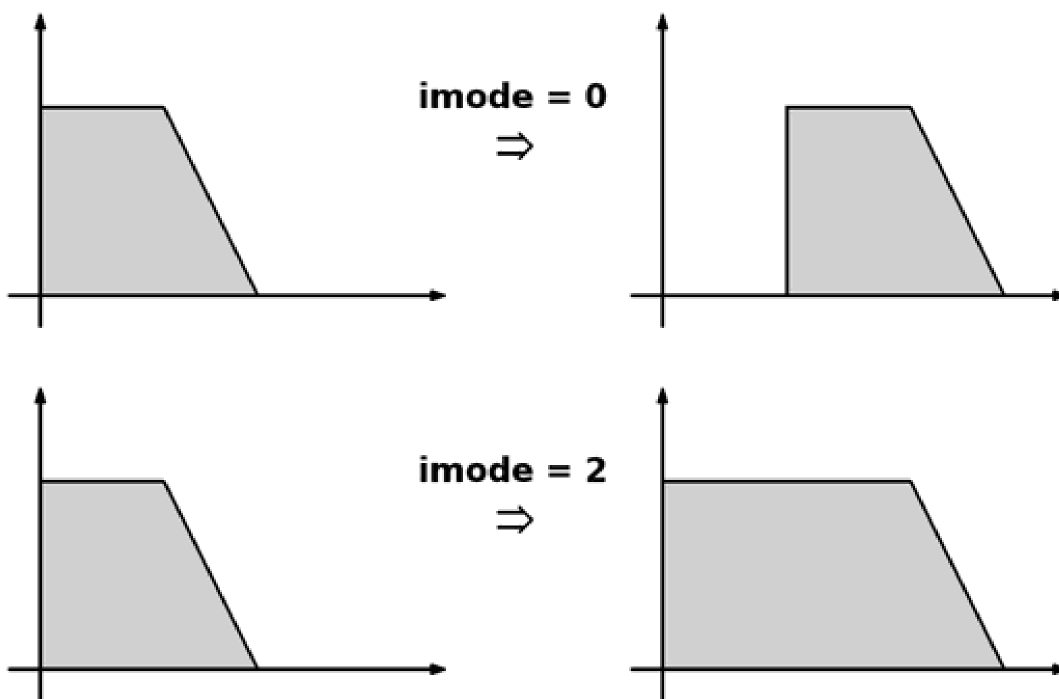
Syntax

```
kr delayk   ksig, idel[, imode]

kr vdel_k   ksig, kdel, imdel[, imode]
```

Initialization

idel -- delay time (in seconds) for delayk. It is rounded to the nearest integer multiple of a k-cycle (i.e. 1/kr).

imode -- sum of 1 for skipping initialization (e.g. in tied notes) and 2 for holding the first input value during the initial delay, instead of outputting zero. This is mainly of use when delaying envelopes that do not start at zero.



imdel -- maximum delay time for vdel_k, in seconds.

Performance

kr -- the output signal. Note: neither of the opcodes interpolate the output.

ksig -- the input signal.

kdel -- delay time (in seconds) for vdel_k. It is rounded to the nearest integer multiple of a k-cycle (i.e. 1/kr).

Credits

Istvan Varga.

delayr

delayr — Reads from an automatically established digital delay line.

Description

Reads from an automatically established digital delay line.

Syntax

```
ares delayr idlt [, iskip]
```

Initialization

idlt -- requested delay time in seconds. This can be as large as available memory will permit. The space required for n seconds of delay is 4n * *sr* bytes. It is allocated at the time the instrument is first initialized, and returned to the pool at the end of a score section.

iskip (optional, default=0) -- initial disposition of delay-loop data space (see *reson*). The default value is 0.

Performance

delayr reads from an automatically established digital delay line, in which the signal retrieved has been resident for *idlt* seconds. This unit must be paired with and precede an accompanying *delayw* unit. Any other Csound statements can intervene.

Examples

See the example for *delayw*.

See Also

delay, *delay1*, *delayw*

delayw

delayw — Writes the audio signal to a digital delay line.

Description

Writes the audio signal to a digital delay line.

Syntax

```
delayw asig
```

Performance

delayw writes *asig* into the delay area established by the preceding *delayr* unit. Viewed as a pair, these two units permit the formation of modified feedback loops, etc. However, there is a lower bound on the value of *idlt*, which must be at least 1 control period (or 1/*kr*).

Examples

Here is an example of the delayw opcode. It uses the file *delayw.csd* [examples/delayw.csd].

Exemple 115. Example of the delayw opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o delayw.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- Delayed beeps.
instr 1
  ; Make a basic sound.
  abeep vco 20000, 440, 1

  ; Set up a delay line.
  idlt = 0.1
  adel delayr idlt

  ; Write the beep to the delay line.
  delayw abeep

  ; Send the beep to the left speaker and
  ; the delayed beep to the right speaker.
  outs abeep, adel
endin
```

```
</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Keep the score running for 2 seconds.
f 0 2

; Play Instrument #1.
i 1 0.0 0.2
i 1 0.5 0.2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

delay, *delay1*, *delayr*

Credits

Example written by Kevin Conder.

deltap

deltap — Taps a delay line at variable offset times.

Description

Tap a delay line at variable offset times.

Syntax

```
ares deltap kdlt
```

Performance

kdlt -- specifies the tapped delay time in seconds. Each can range from 1 control period to the full delay time of the read/write pair; however, since there is no internal check for adherence to this range, the user is wholly responsible. Each argument can be a constant, a variable, or a time-varying signal.

deltap extracts sound by reading the stored samples directly.

This opcode can tap into a *delayr*/*delayw* pair, extracting delayed audio from the *idlt* seconds of stored sound. There can be any number of *deltap* and/or *deltapi* units between a read/write pair. Each receives an audio tap with no change of original amplitude.

This opcode can provide multiple delay taps for arbitrary delay path and feedback networks. They can deliver either constant-time or time-varying taps, and are useful for building chorus effects, harmonizers, and Doppler shifts. Constant-time delay taps (and some slowly changing ones) do not need interpolated readout; they are well served by *deltap*. Medium-paced or fast varying dlt's, however, will need the extra services of *deltapi*.

delayr/*delayw* pairs may be interleaved. To associate a delay tap unit with a specific *delayr* unit, it not only has to be located between that *delayr* and the appropriate *delayw* unit, but must also precede any following *delayr* units. See Example 2. (This feature added in Csound version 3.57 by Jens Groh and John ffitch).

N.B. k-rate delay times are not internally interpolated, but rather lay down stepped time-shifts of audio samples; this will be found quite adequate for slowly changing tap times. For medium to fast-paced changes, however, one should provide a higher resolution audio-rate timeshift as input.

Examples

Exemple 116. deltap example #1

```
asource  buzz      1, 440, 20, 1
atime    linseg    1, p3/2,.01, p3/2,1   ; trace a distance in secs
ampfac   =         1/atime/atime         ; and calc an amp factor
adump    delayr    1                     ; set maximum distance
amove    deltapi   atime                 ; move sound source past
         delayw    asource               ; the listener
         out       amove * ampfac
```

Exemple 117. deltap example #2

```
  ainput1 = .....
  ainput2 = .....
  kdlyt1  = .....
  kdlyt2  = .....

;Read delayed signal, first delayr instance:
  adump   delayr  4.0
  adly1   deltap  kdlyt1       ;associated with first delayr instance

;Read delayed signal, second delayr instance:
  adump   delayr  4.0
  adly2   deltap  kdlyt2       ; associated with second delayr instance

;Do some cross-coupled manipulation:
  afdbk1  =       0.7 * adly1 + 0.7 * adly2 + ainput1
  afdbk2  =       -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
          delayw  afdbk1

;Feed back signal, associated with second delayr instance:
          delayw  afdbk2
          outs    adly1, adly2
```

See Also

deltap3, *deltapi*, *deltapn*

deltap3

deltap — Taps a delay line at variable offset times, uses cubic interpolation.

Description

Taps a delay line at variable offset times, uses cubic interpolation.

Syntax

```
ares deltap3 xdlt
```

Performance

xdlt -- specifies the tapped delay time in seconds. Each can range from 1 control period to the full delay time of the read/write pair; however, since there is no internal check for adherence to this range, the user is wholly responsible. Each argument can be a constant, a variable, or a time-varying signal; the *xdlt* argument in *deltap3* implies that an audio-varying delay is permitted there.

deltap3 is experimental, and uses cubic interpolation. (New in Csound version 3.50.)

This opcode can tap into a *delayr*/*delayw* pair, extracting delayed audio from the *idlt* seconds of stored sound. There can be any number of *deltap* and/or *deltapi* units between a read/write pair. Each receives an audio tap with no change of original amplitude.

This opcode can provide multiple delay taps for arbitrary delay path and feedback networks. They can deliver either constant-time or time-varying taps, and are useful for building chorus effects, harmonizers, and Doppler shifts. Constant-time delay taps (and some slowly changing ones) do not need interpolated readout; they are well served by *deltap*. Medium-paced or fast varying dlt's, however, will need the extra services of *deltapi*.

delayr/*delayw* pairs may be interleaved. To associate a delay tap unit with a specific *delayr* unit, it not only has to be located between that *delayr* and the appropriate *delayw* unit, but must also precede any following *delayr* units. See Example 2. (This feature added in Csound version 3.57 by Jens Groh and John ffitch).

N.B. k-rate delay times are not internally interpolated, but rather lay down stepped time-shifts of audio samples; this will be found quite adequate for slowly changing tap times. For medium to fast-paced changes, however, one should provide a higher resolution audio-rate timeshift as input.

Examples

Exemple 118. deltap example #1

```
asource  buzz      1, 440, 20, 1
atime    linseg    1, p3/2,.01, p3/2,1   ; trace a distance in secs
ampfac   =         1/atime/atime         ; and calc an amp factor
adump    delayr    1                     ; set maximum distance
amove    deltapi   atime                 ; move sound source past
         delayw    asource               ; the listener
         out       amove * ampfac
```

Exemple 119. deltap example #2

```
  ainput1 = .....
  ainput2 = .....
  kdlyt1  = .....
  kdlyt2  = .....

;Read delayed signal, first delayr instance:
  adump   delayr  4.0
  adly1   deltap  kdlyt1       ;associated with first delayr instance

;Read delayed signal, second delayr instance:
  adump   delayr  4.0
  adly2   deltap  kdlyt2       ; associated with second delayr instance

;Do some cross-coupled manipulation:
  afdbk1  =       0.7 * adly1 + 0.7 * adly2 + ainput1
  afdbk2  =       -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
          delayw  afdbk1

;Feed back signal, associated with second delayr instance:
          delayw  afdbk2
          outs    adly1, adly2
```

See Also

deltap, *deltapi*, *deltapn*

deltapi

deltapi — Taps a delay line at variable offset times, uses interpolation.

Description

Taps a delay line at variable offset times, uses interpolation.

Syntax

```
ares deltapi xdlt
```

Performance

xdlt -- specifies the tapped delay time in seconds. Each can range from 1 control period to the full delay time of the read/write pair; however, since there is no internal check for adherence to this range, the user is wholly responsible. Each argument can be a constant, a variable, or a time-varying signal; the *xdlt* argument in *deltapi* implies that an audio-varying delay is permitted there.

deltapi extracts sound by interpolated readout. By interpolating between adjacent stored samples *deltapi* represents a particular delay time with more accuracy, but it will take about twice as long to run.

This opcode can tap into a *delayr*/*delayw* pair, extracting delayed audio from the *idlt* seconds of stored sound. There can be any number of *deltap* and/or *deltapi* units between a read/write pair. Each receives an audio tap with no change of original amplitude.

This opcode can provide multiple delay taps for arbitrary delay path and feedback networks. They can deliver either constant-time or time-varying taps, and are useful for building chorus effects, harmonizers, and Doppler shifts. Constant-time delay taps (and some slowly changing ones) do not need interpolated readout; they are well served by *deltap*. Medium-paced or fast varying dlt's, however, will need the extra services of *deltapi*.

delayr/*delayw* pairs may be interleaved. To associate a delay tap unit with a specific *delayr* unit, it not only has to be located between that *delayr* and the appropriate *delayw* unit, but must also precede any following *delayr* units. See Example 2. (This feature added in Csound version 3.57 by Jens Groh and John ffitch).

N.B. k-rate delay times are not internally interpolated, but rather lay down stepped time-shifts of audio samples; this will be found quite adequate for slowly changing tap times. For medium to fast-paced changes, however, one should provide a higher resolution audio-rate timeshift as input.

Examples

Exemple 120. deltap example #1

```
asource  buzz      1, 440, 20, 1
atime    linseg    1, p3/2,.01, p3/2,1    ; trace a distance in secs
ampfac   =         1/atime/atime          ; and calc an amp factor
adump    delayr    1                      ; set maximum distance
amove    deltapi   atime                  ; move sound source past
         delayw    asource                ; the listener
```

```
          out       amove * ampfac
```

Exemple 121. deltap example #2

```
  ainput1 = .....
  ainput2 = .....
  kdlyt1  = .....
  kdlyt2  = .....

;Read delayed signal, first delayr instance:
  adump   delayr  4.0
  adly1   deltap  kdlyt1       ;associated with first delayr instance

;Read delayed signal, second delayr instance:
  adump   delayr  4.0
  adly2   deltap  kdlyt2       ; associated with second delayr instance

;Do some cross-coupled manipulation:
  afdbk1  =       0.7 * adly1 + 0.7 * adly2 + ainput1
  afdbk2  =       -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
          delayw  afdbk1

;Feed back signal, associated with second delayr instance:
          delayw  afdbk2
          outs    adly1, adly2
```

See Also

deltap, *deltap3*, *deltapn*

deltapn

deltapn — Taps a delay line at variable offset times.

Description

Tap a delay line at variable offset times.

Syntax

```
ares deltapn xnumsamps
```

Performance

xnumsamps -- specifies the tapped delay time in number of samples. Each can range from 1 control period to the full delay time of the read/write pair; however, since there is no internal check for adherence to this range, the user is wholly responsible. Each argument can be a constant, a variable, or a time-varying signal.

deltapn is identical to *deltapi*, except delay time is specified in number of samples, instead of seconds (Hans Mikelson).

This opcode can tap into a *delayr*/*delayw* pair, extracting delayed audio from the *idlt* seconds of stored sound. There can be any number of *deltap* and/or *deltapi* units between a read/write pair. Each receives an audio tap with no change of original amplitude.

This opcode can provide multiple delay taps for arbitrary delay path and feedback networks. They can deliver either constant-time or time-varying taps, and are useful for building chorus effects, harmonizers, and Doppler shifts. Constant-time delay taps (and some slowly changing ones) do not need interpolated readout; they are well served by *deltap*. Medium-paced or fast varying dlt's, however, will need the extra services of *deltapi*.

delayr/*delayw* pairs may be interleaved. To associate a delay tap unit with a specific *delayr* unit, it not only has to be located between that *delayr* and the appropriate *delayw* unit, but must also precede any following *delayr* units. See Example 2. (This feature added in Csound version 3.57 by Jens Groh and John ffitch).

N.B. k-rate delay times are not internally interpolated, but rather lay down stepped time-shifts of audio samples; this will be found quite adequate for slowly changing tap times. For medium to fast-paced changes, however, one should provide a higher resolution audio-rate timeshift as input.

Examples

Exemple 122. deltap example #1

```
asource  buzz      1, 440, 20, 1
atime    linseg    1, p3/2,.01, p3/2,1    ; trace a distance in secs
ampfac   =         1/atime/atime          ; and calc an amp factor
adump    delayr    1                      ; set maximum distance
amove    deltapi   atime                  ; move sound source past
         delayw    asource                ; the listener
```

```
          out       amove * ampfac
```

Exemple 123. deltap example #2

```
  ainput1 = .....
  ainput2 = .....
  kdlyt1  = .....
  kdlyt2  = .....

;Read delayed signal, first delayr instance:
  adump   delayr  4.0
  adly1   deltap  kdlyt1       ;associated with first delayr instance

;Read delayed signal, second delayr instance:
  adump   delayr  4.0
  adly2   deltap  kdlyt2       ; associated with second delayr instance

;Do some cross-coupled manipulation:
  afdbk1  =       0.7 * adly1 + 0.7 * adly2 + ainput1
  afdbk2  =       -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
          delayw  afdbk1

;Feed back signal, associated with second delayr instance:
          delayw  afdbk2
          outs    adly1, adly2
```

See Also

deltap, *deltap3*, *deltapi*

deltapx

deltapx — Read to or write from a delay line with interpolation.

Description

deltapx is similar to *deltapi* or *deltap3*. However, it allows higher quality interpolation. This opcode can read from and write to a delayr/delayw delay line with interpolation.

Syntax

```
aout deltapx adel, iwsize
```

Initialization

iwsize -- interpolation window size in samples. Allowed values are integer multiplies of 4 in the range 4 to 1024. iwsize = 4 uses cubic interpolation. Increasing iwsize improves sound quality at the expense of CPU usage, and minimum delay time.

Performance

aout -- Output signal

adel -- Delay time in seconds.

```
a1      delayr idlr
        deltapxw a2, adl1, iws1
a3      deltapx adl2, iws2
        deltapxw a4, adl3, iws3
        delayw a5
```

Minimum and maximum delay times:

```
idlr >= 1/kr                               Delay line length

adl1 >= (iws1/2)/sr                        Write before read
adl1 <= idlr - (1 + iws1/2)/sr             (allows shorter delays)

adl2 >= 1/kr + (iws2/2)/sr                 Read time
adl2 <= idlr - (1 + iws2/2)/sr
adl2 >= adl1 + (iws1 + iws2) / (2*sr)
adl2 >= 1/kr + adl3 + (iws2 + iws3) / (2*sr)

adl3 >= (iws3/2)/sr                        Write after read
adl3 <= idlr - (1 + iws3/2)/sr             (allows feedback)
```

Note

Window sizes for opcodes other than deltapx are: deltap, deltapn: 1, deltapi: 2 (linear), del-

tap3: 4 (cubic)

Examples

```
a1      phasor 300.0
a1      =  a1 - 0.5
a_      delayr 1.0
adel    phasor 4.0
adel    =  sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
        deltapxw a1, adel, 32
adel    phasor 2.0
adel    =  sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
        deltapxw a1, adel, 32
adel    =  0.3
a2      deltapx adel, 32
a1      =  0
        delayw a1

        out a2 * 20000.0
```

See Also

deltapxw

Credits

Author: Istvan Varga
August 2001

New in version 4.13

deltapxw

deltapxw — Mixes the input signal to a delay line.

Description

deltapxw mixes the input signal to a delay line. This opcode can be mixed with reading units (*deltap*, *deltapn*, *deltapi*, *deltap3*, and *deltapx*) in any order; the actual delay time is the difference of the read and write time. This opcode can read from and write to a *delayr*/*delayw* delay line with interpolation.

Syntax

```
deltapxw ain, adel, iwsize
```

Initialization

iwsize -- interpolation window size in samples. Allowed values are integer multiplies of 4 in the range 4 to 1024. iwsize = 4 uses cubic interpolation. Increasing iwsize improves sound quality at the expense of CPU usage, and minimum delay time.

Performance

ain -- Input signal

adel -- Delay time in seconds.

```
a1      delayr idlr
        deltapxw a2, adl1, iws1
a3      deltapx adl2, iws2
        deltapxw a4, adl3, iws3
        delayw a5
```

Minimum and maximum delay times:

```
idlr >= 1/kr                                Delay line length

adl1 >= (iws1/2)/sr                         Write before read
adl1 <= idlr - (1 + iws1/2)/sr              (allows shorter delays)

adl2 >= 1/kr + (iws2/2)/sr                  Read time
adl2 <= idlr - (1 + iws2/2)/sr
adl2 >= adl1 + (iws1 + iws2) / (2*sr)
adl2 >= 1/kr + adl3 + (iws2 + iws3) / (2*sr)

adl3 >= (iws3/2)/sr                         Write after read
adl3 <= idlr - (1 + iws3/2)/sr              (allows feedback)
```

Note

Window sizes for opcodes other than deltapx are: deltap, deltapn: 1, deltapi: 2 (linear), deltap3: 4 (cubic)

Examples

```
a1      phasor 300.0
a1      =  a1 - 0.5
a_      delayr 1.0
adel    phasor 4.0
adel    =  sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
        deltapxw a1, adel, 32
adel    phasor 2.0
adel    =  sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
        deltapxw a1, adel, 32
adel    =  0.3
a2      deltapx adel, 32
a1      =  0
        delayw a1

        out a2 * 20000.0
```

See Also

deltapx

Credits

Author: Istvan Varga
August 2001

New in version 4.13

denorm

denorm — Mixes low level noise to a list of a-rate signals

Description

Mixes low level (~1e-20 for floats, and ~1e-56 for doubles) noise to a list of a-rate signals. Can be used before IIR filters and reverbs to avoid denormalized numbers which may otherwise result in significantly increased CPU usage.

Syntax

```
denorm a1[, a2[, a3[, ... ]]]
```

Performance

a1[, a2[, a3[, ...]]] -- signals to mix noise with

Some processor architectures (particularly Pentium IVs) are very slow at processing extremely small numbers. These small numbers can appear as a result of some decaying feedback process like reverb and IIR filters. Low level noise can be added so that very small numbers are never reached, and they are 'absorbed' by this 'noise floor'.

If CPU usage goes to 100% at the end of reverb tails, or you get audio glitches in processes that shouldn't use too much CPU, using *denorm* before the culprit opcode or process might solve the problem.

Credits

Author: Istvan Varga
2005

diff

diff — Modify a signal by differentiation.

Description

Modify a signal by differentiation.

Syntax

```
ares diff asig [, iskip]
```

```
kres diff ksig [, iskip]
```

Initialization

iskip (optional) -- initial disposition of internal save space (see *reson*). The default value is 0.

Performance

integ and *diff* perform integration and differentiation on an input control signal or audio signal. Each is the converse of the other, and applying both will reconstruct the original signal. Since these units are special cases of low-pass and high-pass filters, they produce a scaled (and phase shifted) output that is frequency-dependent. Thus *diff* of a sine produces a cosine, with amplitude *2 * sin(pi * Hz /* sr*)* that of the original (for each component partial); *integ* will inversely affect the magnitudes of its component inputs. With this understanding, these units can provide useful signal modification.

Examples

Here is an example of the diff opcode. It uses the file *diff.csd* [examples/diff.csd].

Exemple 124. Example of the diff opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o diff.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- a normal instrument.
instr 1
  ; Generate a band-limited pulse train.
```

```
  asrc buzz 20000, 440, 20,  1

  out asrc
endin

; Instrument #2 -- a differentiated instrument.
instr 2
  ; Generate a band-limited pulse train.
  asrc buzz 20000, 440, 20,  1

  ; Emphasize the highs.
  a1 diff asrc

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e


</CsScore>
</CsoundSynthesizer>
```

See Also

downsamp, *integ*, *interp*, *samphold*, *upsamp*

Credits

Example written by Kevin Conder.

diskgrain

diskgrain — Synthèse granulaire synchrone, utilisant un fichier son comme source.

Description

diskgrain implémente la synthèse granulaire synchrone. La source sonore des grains est obtenue en lisant un fichier son contenant les échantillons de la forme d'onde source.

Syntaxe

```
asig diskgrain Sfname, kamp, kfreq, kpitch, kgrsize, kprate, \
      ifun, iolaps[, ioffset, imaxgrsize]
```

Initialisation

Sfilename -- fichier son source.

ifun -- table de fonction de l'enveloppe de grain.

iolaps -- nombre maximum de chevauchements, max(*kfreq*)*max(*kgrsize*). Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

ioffset -- décalage initial en secondes à partir du début du fichier (par défaut 0).

imaxgrsize -- taille de grain maximale en secondes (par défaut 1.0).

Exécution

kamp -- pondération de l'amplitude

kfreq -- fréquence de génération des grains, ou densité, en grains/sec.

kpitch -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers)

kgrsize -- taille de grain en secondes.

kprate -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

Le générateur de grain contrôle complètement la fréquence (grains/sec), l'amplitude globale, la hauteur de grain (un incrément de l'échantillonage) et la taille de grain (en secondes), comme paramètres fixes ou variant dans le temps (signaux). La vitesse du pointeur de grain est un paramètre supplémentaire qui contrôle la position à laquelle le générateur commencera à lire les échantillons dans le fichier pour chaque grain successif. Elle est mesurée en fraction de la taille de grain, si bien qu'une valeur de 1 (par défaut) provoquera la lecture de chaque grain successif à partir de l'endroit où le grain précédent s'est terminé. Avec une valeur de 0.5 le grain suivant commencera à la position médiane entre le début et la fin du grain précédent, etc... Avec une valeur de 0 le générateur lit toujours à partir d'une position fixe (quelque soit l'endroit où il se trouvait précédemment). Une valeur négative décrémentera les positions

du pointeur. Ce contrôle donne plus de flexibilité pour créer des modifications de l'échelle temporelle pendant la resynthèse.

Diskgrain générera n'importe quel nombre de flux de grain parallèles (en fonction de la densité/fréquence de grain) borné par la valeur de *iolaps* (par défaut 100). Le nombre de flux (grains se chevauchant) est déterminé par *kgrsize*kfreq*. Plus il y aura de chevauchements, plus il y aura de calculs ce qui pourra empêcher la synthèse en temps réel (selon la puissance du processeur).

Diskgrain peut simuler une synthèse formantique à la FOF, si on emploie une forme adéquate comme enveloppe de grain et une forme d'onde sinus comme onde de grain. Pour cette utilisation, on peut choisir des tailles de grain d'environ 0.04 secondes. La fréquence centrale du formant est déterminée par la hauteur de grain. Comme celle-ci est exprimée en incrément d'échantillonnage, il faut pondérer cette valeur par *tablesize/sr* pour obtenir une fréquence en Hz. La fréquence de grain déterminera le fondamental.

Cet opcode est une variation sur l'opcode *syncgrain*.

Exemples

Voici un exemple de l'opcode diskgrain. Il utilise le fichier *diskgrain.csd* [examples/diskgrain.csd].

Exemple 125. Exemple de l'opcode diskgrain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 128

instr 1
iolaps = 2
igrsize = 0.04
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = p4  /* timescale */
ipitch = p5 /* pitchscale */

a1 diskgrain "mary.wav", 32000, ifreq, ipitch, igrsize, ips*istr, 1, iolaps

out   a1
endin

</CsInstruments>
<CsScore>
f 1 0 8192 20   1   1  ;Hamming function

;           timescale   pitchscale
i 1   0   5     1           1
i 1   +   5     2           1
i 1   +   5     1          0.75
i 1   +   5     1.5        1.5
i 1   +   5     0.5        1.5

e
</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Victor Lazzarini;
Mai 2007
Nouveau dans Csound 5.06

diskin

diskin — Reads audio data from an external device or stream and can alter its pitch.

Description

Reads audio data from an external device or stream and can alter its pitch.

Syntax

```
ar1 [, ar2 [, ar3 [, ... ar24]]] diskin ifilcod, kpitch [, iskiptim] \
      [, iwraparound] [, iformat] [, iskipinit]
```

Initialization

ifilcod -- integer or character-string denoting the source soundfile name. An integer denotes the file soundin.filcod ; a character-string (in double quotes, spaces permitted) gives the filename itself, optionally a full pathname. If not a full path, the named file is sought first in the current directory, then in that given by the environment variable SSDIR (if defined) then by SFDIR. See also *GEN01*.

iskptim (optional) -- time in seconds of input sound to be skipped. The default value is 0.

iformat (optional) -- specifies the audio data file format:

- 1 = 8-bit signed char (high-order 8 bits of a 16-bit integer)
- 2 = 8-bit A-law bytes
- 3 = 8-bit U-law bytes
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = 8-bit unsigned int (not available in Csound versions older than 5.00)
- 8 = 24-bit int (not available in Csound versions older than 5.00)
- 9 = 64-bit doubles (not available in Csound versions older than 5.00)

iwraparound -- 1 = on, 0 = off (wraps around to end of file either direction)

iskipinit switches off all initialisation if non zero (default =0). This was introduced in 4_23f13 and csound5.

If *iformat* = 0 it is taken from the soundfile header, and if no header from the Csound *-o* command-line flag. The default value is 0.

Performance

kpitch -- can be any real number. a negative number signifies backwards playback. The given number is a pitch ratio, where:

- *1* = normal pitch
- *2* = 1 octave higher
- *3* = 12th higher, etc.
- *.5* = 1 octave lower
- *.25* = 2 octaves lower, etc.
- *-1* = normal pitch backwards
- *-2* = 1 octave higher backwards, etc.

diskin is identical to *soundin* except that it can alter the pitch of the sound that is being read.

Note to Windows users

Windows users typically use back-slashes, « \ », when specifying the paths of their files. As an example, a Windows user might use the path « c:\music\samples\loop001.wav ». This is problematic because back-slashes are normally used to specify special characters.

To correctly specify this path in Csound, one may alternately:

- *Use forward slashes*: c:/music/samples/loop001.wav
- *Use back-slash special characters, « \\ »*: c:\\music\\samples\\loop001.wav

Examples

Here is an example of the diskin opcode. It uses the file *diskin.csd* [examples/diskin.csd], *beats.wav* [examples/beats.wav].

Exemple 126. Example of the diskin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o diskin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
```

```
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
  ; Play the audio file backwards.
  asig diskin "beats.wav", -1
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

in, *inh*, *ino*, *inq*, *ins*, *soundin* and *diskin2*

Credits

Authors: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

New in versin 3.46

Example written by Kevin Conder.

Warning to Windows users added by Kevin Conder, April 2002

diskin2

diskin2 — Reads audio data from a file, and can alter its pitch using one of several available interpolation types, as well as convert the sample rate to match the orchestra sr setting.

Description

Reads audio data from a file, and can alter its pitch using one of several available interpolation types, as well as convert the sample rate to match the orchestra sr setting. diskin2 can also read multichannel files with any number of channels in the range 1 to 24. diskin2 allows more control and higher sound quality than diskin, but there is also the disadvantage of higher CPU usage.

Syntax

```
a1[, a2[, ... a24]] diskin2 ifilcod, kpitch[, iskiptim \
      [, iwrap[, iformat [, iwsize[, ibufsize[, iskipinit]]]]]]
```

Initialization

ifilcod -- integer or character-string denoting the source soundfile name. An integer denotes the file soundin.ifilcod; a character-string (in double quotes, spaces permitted) gives the filename itself, optionally a full pathname. If not a full path, the named file is sought first in the current directory, then in those given by the environment variable SSDIR (if defined) then by SFDIR. See also GEN01. Note: files longer than 2^31-1 sample frames may not be played correctly on 32 bit platforms; this means a maximum length about 3 hours with a sample rate of 192000 Hz.

iskiptim (optional, defaults to zero) -- time in seconds of input sound to be skipped, assuming kpitch=1. Can be negative, to add -iskiptim/kpitch seconds of delay instead of skipping sound.

iwrap (optional, defaults to zero) -- if set to any non-zero value, read locations that are negative or are beyond the end of the file are wrapped to the duration of the sound file instead of assuming zero samples. Useful for playing a file in a loop.

Note

If iwrap is enabled, the file length should not be shorter than the interpolation window size (see below), otherwise there may be clicks in the sound output.

iformat (optional, defaults to zero) -- sample format, for raw (headerless) files only. This parameter is ignored if the file has a header. Allowed values are:

- 0: 16-bit short integers
- 1: 8-bit signed char (high-order 8 bits of a 16-bit integer)
- 2: 8-bit A-law bytes
- 3: 8-bit U-law bytes
- 4: 16-bit short integers
- 5: 32-bit long integers

- 6: 32-bit floats
- 7: 8-bit unsigned int
- 8: 24-bit int
- 9: 64-bit doubles

iwsize (optional, defaults to zero) -- interpolation window size, in samples. Can be one of the following:

- 1: round to nearest sample (no interpolation, for kpitch=1)
- 2: linear interpolation
- 4: cubic interpolation
- >= 8: iwsize point sinc interpolation with anti-aliasing (slow)

Zero or negative values select the default, which is cubic interpolation.

Note

If interpolation is used, kpitch is automatically scaled by the ratio of the sample rate of the sound file and the orchestra, so that the file will always be played at the original pitch if kpitch is 1. However, the sample rate conversion is disabled if iwsize is 1.

ibufsize (optional, defaults to 0) -- buffer size in mono samples (not sample frames). This is only the suggested value, the actual setting will be rounded so that the number of sample frames is an integer power of two and is in the range 128 (or iwsize if greater than 128) to 1048576. The default, which is 4096, and is enabled by zero or negative values, should be suitable for most uses, but for non-realtime mixing of many large sound files, a high buffer setting is recommended to improve the efficiency of disk reads. For real time audio output, reading the files from a fast RAM file system (on platforms where this option is available) with a small buffer size may be preferred.

iskipinit (optional, defaults to 0) -- skip initialization if set to any non-zero value.

Performance

a1 ... a24 -- output signals, in the range -0dbfs to 0dbfs. Any samples before the beginning (i.e. negative location) and after the end of the file are assumed to be zero, unless iwrap is non-zero. The number of output arguments must be the same as the number of sound file channels - which can be determined with the filenchnls opcode, otherwise an init error will occur.

Note

It is more efficient to read a single file with many channels, than many files with only a single channel, especially with high iwsize settings.

kpitch -- transpose the pitch of input sound by this factor (e.g. 0.5 means one octave lower, 2 is one octave higher, and 1 is the original pitch). Fractional and negative values are allowed (the latter results in playing the file backwards, however, in this case the skip time parameter should be set to some positive value, e.g. the length of the file, or iwrap should be non-zero, otherwise nothing would be played). If interpolation is enabled, and the sample rate of the file differs from the orchestra sample rate, the transpose ratio is automatically adjusted to make sure that kpitch=1 plays at the original pitch. Using a high

iwsize setting (40 or more) can significantly improve sound quality when transposing up, although at the expense of high CPU usage.

Example

```
<CsoundSynthesizer>
<CsOptions>
; set this to a directory where beats.wav can be found
--env:SSDIR+=/Csound/Documentation/manual/examples
</CsOptions>
<CsInstruments>
sr      =  48000
ksmps   =  32
nchnls  =  2

       instr 1

ktrans  linseg 1, 5, 2, 10, -2
a1      diskin2 "beats.wav", ktrans, 0, 1, 0, 32
       outs a1, a1
       endin

</CsInstruments>
<CsScore>

i 1 0 15
e

</CsScore>
</CsoundSynthesizer>
```

See Also

in, *inh*, *ino*, *inq*, *ins*, *soundin* and *diskin2*

Credits

Author: Istvan Varga
2005

New in version 5.00

dispfft

displayfft — Displays the Fourier Transform of an audio or control signal.

Description

These units will print orchestra init-values, or produce graphic display of orchestra control signals and audio signals. Uses X11 windows if enabled, else (or if *-g* flag is set) displays are approximated in ASCII characters.

Syntax

```
dispfft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg]
```

Initialization

iprd -- the period of display in seconds.

iwsiz -- size of the input window in samples. A window of *iwsiz* points will produce a Fourier transform of *iwsiz*/2 points, spread linearly in frequency from 0 to sr/2. *iwsiz* must be a power of 2, with a minimum of 16 and a maximum of 4096. The windows are permitted to overlap.

iwtyp (optional, default=0) -- window type. 0 = rectangular, 1 = Hanning. The default value is 0 (rectangular).

idbout (optional, default=0) -- units of output for the Fourier coefficients. 0 = magnitude, 1 = decibels. The default is 0 (magnitude).

iwtflg (optional, default=0) -- wait flag. If non-zero, each display is held until released by the user. The default value is 0 (no wait).

Performance

dispfft -- displays the Fourier Transform of an audio or control signal (*asig* or *ksig*) every *iprd* seconds using the Fast Fourier Transform method.

Examples

Here is an example of the dispfft opcode. It uses the file *dispfft.csd* [examples/dispfft.csd] and *beats.wav* [examples/beats.wav].

Exemple 127. Example of the dispfft opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o dispfft.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  asig soundin "beats.wav"
  dispfft asig, 1, 512
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

display, *print*

Credits

Comments about the *inprds* parameter contributed by Rasmus Ekman.

Example written by Kevin Conder.

display

display — Displays the audio or control signals as an amplitude vs. time graph.

Description

These units will print orchestra init-values, or produce graphic display of orchestra control signals and audio signals. Uses X11 windows if enabled, else (or if *-g* flag is set) displays are approximated in ASCII characters.

Syntax

```
display xsig, iprd [, inprds] [, iwtflg]
```

Initialization

iprd -- the period of display in seconds.

inprds (optional, default=1) -- Number of display periods retained in each display graph. A value of 2 or more will provide a larger perspective of the signal motion. The default value is 1 (each graph completely new).

inprds (optional, default=1) -- a scaling factor for the displayed waveform, controlling how many iprd-sized frames of samples are drawn in the window (the default and minimum value is 1.0). Higher inprds values are slower to draw (more points to draw) but will show the waveform scrolling through the window, which is useful with low iprd values.

iwtflg (optional, default=0) -- wait flag. If non-zero, each display is held until released by the user. The default value is 0 (no wait).

Performance

display -- displays the audio or control signal *xsig* every *iprd* seconds, as an amplitude vs. time graph.

Examples

Here is an example of the display opcode. It uses the file *display.csd* [examples/display.csd].

Exemple 128. Example of the display opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o display.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Go from 1000 to 0 linearly, over the period defined by p3.
  klin line 1000, p3, 0

  ; Create a new display each second, wait for the user.
  display klin, 1, 1, 1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

See Also

dispfft, *print*

Credits

Comments about the *inprds* parameter contributed by Rasmus Ekman.

Example written by Kevin Conder.

distort

distort — Distort an audio signal via waveshaping and optional clipping.

Description

Syntax

```
ar distort asig, kdist, ifn[, ihp, istor]
```

Initialization

ifn -- table number of a waveshaping function with extended guard point. The function can be of any shape, but it should pass through 0 with positive slope at the table mid-point. The table size need not be large, since it is read with interpolation.

ihp -- (optional) half-power point (in cps) of an internal low-pass filter. The default value is 10.

istor -- (optional) initial disposition of internal data space (see reson). The default value is 0.

Performance

asig -- Audio singal to be processed

kdist -- Amount of distortion (usually between 0 and 1)

This unit distorts an incoming signal using a waveshaping function *ifn* and a distortion index *kdist*. The input signal is first compressed using a running rms, then passed through a waveshaping function which may modify its shape and spectrum. Finally it is rescaled to approximately its original power.

The amount of distortion depends on the nature of the shaping function and on the value of *kdist*, which generally ranges from 0 to 1. For low values of *kdist*, we should like the shaping function to pass the signal almost unchanged. This will be the case if, at the mid-point of the table, the shaping function is near-linear and is passing through 0 with positive slope. A line function from -1 to +1 will satisfy this requirement; so too will a sigmoid (sinusoid from 270 to 90 degrees). As *kdist* is increased, the compressed signal is expanded to encounter more and more of the shaping function, and if this becomes non-linear the signal is increasingly *bent* on read-through to cause distortion.

When kdist becomes large enough, the read-through process will eventually hit the outer limits of the table. The table is not read with wrap-around, but will ´stick¡ at the end-points as the incoming signal exceeds them; this introduces clipping, an additional form of signal distortion. The point at which clipping begins will depend on the complexity (rms-to-peak value) of the input signal. For a pure sinusoid, clipping will begin only as kdist exceeds 0.7; for a more complex input, clipping might begin at a kdist of 0.5 or much less. kdist can exceed the clip point by any amount, and may be greater than 1.

The shaping function can be made arbitrarily complex for extra effect. It should generally be continuous, though this is not a requirement. It should also be well-behaved near the mid-point, and roughly balanced positive-negative overall, else some excessive DC offset may result. The user might experiment with more aggressive functions to suit the purpose. A generally positive slope allows the distorted signal to be mixed with the source without phase cancellation.

distort is useful as an effects process, and is usually combined with reverb and chorusing on effects

busses. However, it can alternatively be used to good effect within a single instrument.

Examples

```
gifn   ftgen         0,0, 257, 9, .5,1,270         ; define a sigmoid, or better
gifn   ftgen         0,0, 257, 9, .5,1,270,1.5,.33,90,2.5,.2,270,3.5,.143,90,4.5,.111,270

kdist   line         0, 10, 1.2                 ; and over 10 seconds
aout   distort          asig, kdist, gifn         ; gradually increase the distortion
```

Credits

Written by Barry L. Vercoe for Extended Csound and released in csound5.

distort1

distort1 — Modified hyperbolic tangent distortion.

Description

Implementation of modified hyperbolic tangent distortion. *distort1* can be used to generate wave shaping distortion based on a modification of the *tanh* function.

```
       exp(asig * (shape1 + pregain)) - exp(asig * (shape2 - pregain))
 aout = ---------------------------------------------------------------
       exp(asig * pregain)           + exp(-asig * pregain)
```

Syntax

```
ares distort1 asig, kpregain, kpostgain, kshape1, kshape2[, imode]
```

Initialization

imode (Csound version 5.00 and later only; optional, defaults to 0) -- scales kpregain, kpostgain, kshape1, and kshape2 for use with audio signals in the range -32768 to 32768 (imode=0), -0dbfs to 0dbfs (imode=1), or disables scaling of kpregain and kpostgain and scales kshape1 by kpregain and kshape2 by -kpregain (imode=2).

Performance

asig -- is the input signal.

kpregain -- determines the amount of gain applied to the signal before waveshaping. A value of 1 gives slight distortion.

kpostgain -- determines the amount of gain applied to the signal after waveshaping.

kshape1 -- determines the shape of the positive part of the curve. A value of 0 gives a flat clip, small positive values give sloped shaping.

kshape2 -- determines the shape of the negative part of the curve.

Examples

Here is an example of the distort1 opcode. It uses the file *distort1.csd* [examples/distort1.csd].

Exemple 129. Example of the distort1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o distort1.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gadist init 0

instr 1
  iamp = p4
  ifqc = cpspch(p5)
  asig pluck iamp, ifqc, ifqc, 0, 1
  gadist = gadist + asig
endin

instr 50
  kpre init p4
  kpost init p5
  kshap1 init p6
  kshap2 init p7
  aout distort1 gadist, kpre, kpost, kshap1, kshap2

  outs aout, aout

  gadist = 0
endin


</CsInstruments>
<CsScore>

;   Sta  Dur  Amp    Pitch
i1  0.0  3.0  10000  6.00
i1  0.5  2.5  10000  7.00
i1  1.0  2.0  10000  7.07
i1  1.5  1.5  10000  8.00

;   Sta  Dur  PreGain PostGain Shape1 Shape2
i50 0    3    2       1        0      0
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Hans Mikelson
December 1998

New in Csound version 3.50

divz

divz — Division protégée de deux nombres.

Syntaxe

```
ares divz xa, xb, ksubst
```

```
ires divz ia, ib, isubst
```

```
kres divz ka, kb, ksubst
```

Description

Division protégée de deux nombres.

Initialisation

Lorsque *b* est différent de zéro, le résultat reçoit la valeur de *a / b* ; si *b* est égal à zéro, le résultat prend la valeur de *subst*.

Exemples

Voici un exemple de l'opcode divz. Il utilise le fichier *divz.csd* [examples/divz.csd].

Exemple 130. Exemple de l'opcode divz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o divz.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Define the numbers to be divided.
  ka init 200
  ; Linearly change the value of kb from 200 to 0.
  kb line 0, p3, 200
  ; If a "divide by zero" error occurs, substitute -1.
  ksubst init -1

  ; Safely divide the numbers.
  kresults divz ka, kb, ksubst
```

```
  ; Print out the results.
  printks "%f / %f = %f\\n", 0.1, ka, kb, kresults
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
200.000000 / 0.000000 = -1.000000
200.000000 / 19.999887 = 10.000056
200.000000 / 40.000027 = 4.999997
```

Voir Aussi

=, *init*, *tival*

Crédits

Auteur : John ffitch d'après une idée de Barry L. Vercoe

Exemple écrit par Kevin Conder.

downsamp

downsamp — Modify a signal by down-sampling.

Description

Modify a signal by down-sampling.

Syntax

```
kres downsamp asig [, iwlen]
```

Initialization

iwlen (optional) -- window length in samples over which the audio signal is averaged to determine a downsampled value. Maximum length is *ksmps*; 0 and 1 imply no window averaging. The default value is 0.

Performance

downsamp converts an audio signal to a control signal by downsampling. It produces one kval for each audio control period. The optional window invokes a simple averaging process to suppress foldover.

Examples

Here is an example of the downsamp opcode. It uses the file *downsamp.csd* [examples/downsamp.csd].

Exemple 131. Example of the downsamp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o downsamp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create a noise signal at a-rate.
  anoise noise 20000, 0.2

  ; Downsample the noise signal to k-rate.
  knoise downsamp anoise
```

```
  ; Use the noise signal at k-rate.
  a1 oscil 30000, knoise, 1
  out anoise
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

See Also

diff, *integ*, *interp*, *samphold*, *upsamp*

Credits

Example written by Kevin Conder.

dripwater

dripwater — Modèle semi-physique d'une goutte d'eau.

Description

dripwater est un modèle semi-physique d'une goutte d'eau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntax

```
ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]
```

Initialisation

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 10.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,996 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,996 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 2,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale. Rasmus Ekman proposee un intervalle de 1,4 à 1,75. Il suggère aussi une valeur maximale de 1,9 au lieu de la limite théorique de 2,0.

imaxshake (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

ifreq (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 450.

ifreq1 (facultatif) -- la première fréquence de résonance. La valeur par défaut est 600.

ifreq2 (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 750.

Exécution

kamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

Exemples

Voici un exemple de l'opcode dripwater. Il utilise le fichier *dripwater.csd* [examples/dripwater.csd].

Exemple 132. Exemple de l'opcode dripwater.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dripwater.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 01  ;example of a water drip
a1  line 5, p3, 5  ;preset an amplitude boost
a2  dripwater p4, 0.01, 0, .9  ;dripwater needs a little amplitude help at these values
a3  product a1, a2 ;increase amplitude
    out a3
    endin


</CsInstruments>
<CsScore>

i1 0 1 20000
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

bamboo, *guiro*, *sleighbells*, *tambourine*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

dssiactivate

dssiactivate — Activates or deactivates a DSSI or LADSPA plugin.

Syntax

```
dssiactivate ihandle, ktoggle
```

Description

dssiactivate is used to activate or deactivate a DSSI or LADSPA plugin. It calles the plugin's activate() and deactivate() functions if they are provided.

Initialization

ihandle - the number which identifies the plugin, generated by *dssiinit*.

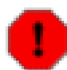
Performance

ktoggle - Selects between activation (ktoggle=1) and deactivation (ktoggle=0).

dssiactivate is used to turn on and off plugins if they provide this facility. This may help conserve CPU processing in some cases. For consistency, all plugins must be activated to produce sound. An inactive plugin produces silence.

Depending on the plugin's implementation, this may cause interruptions in the realtime audio process, so use with caution.

dssiactivate may cause audio stream breakups when used in realtime, so it is recommended to load all plugins to be used before playing.



Avertissement

Please note that even if activate() and deactivate() functions are not present in a plugin, *dssiactivate* must be called for the plugin to produce sound.

Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

dssiaudio

dssiaudio — Processes audio using a LADSPA or DSSI plugin.

Syntax

```
aout1 [, aout2, aout3, aout4] dssiaudio ihandle, ain1 [,ain2, ain3, ain4]
```

Description

dssiaudio generates audio by processing an input signal through a LADSPA plugin.

Initialization

ihandle - handle for the plugin returned by *dssiinit*

Performance

aout1, aout2, etc - Audio ouput generated by the plugin

ain1, ain2, etc - Audio provided to the plugin for processing

dssiaudio runs a plugin on the provided audio and produces audio output. Currently upto four inputs and outputs are provided. You should provide signal for all the plugins audio inputs, otherwise unpredictable results may occur. If the plugin doesn't have any input (e.g Noise generator) you must still provide at least one input variable, which will be ignored with a message.

Only one *dssiaudio* should be executed once per plugin, or strange results may occur.

Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

dssictls

dssictls — Send control information to a LADSPA or DSSI plugin.

Syntax

```
dssictls ihandle, iport, kvalue, ktrigger
```

Description

dssictls sends control values to a plugin's control port

Initialization

ihandle - handle for the plugin returned by *dssiinit*

iport - control port number

Performance

kvalue - value to be assigned to the port

ktrigger - determines whether the control information will be sent (ktrigger = 1) or not. This is useful for thinning control information, generating ktrigger with *metro*

dssictls sends control information to a LADSPA or DSSI plugin's control port. The valid control ports and ranges are given by *dssiinit* . Using values outside the ranges may produce unspecified behaviour.

Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

dssiinit

dssiinit — Loads a DSSI or LADSPA plugin.

Syntax

```
ihandle dssiinit ilibraryname, iplugindex [, iverbose]
```

Description

dssiinit is used to load a DSSI or LADSPA plugin into memory for use with the other dssi4cs opcodes. Both LADSPA effects and DSSI instruments can be used.

Initialization

ihandle - the number which identifies the plugin, to be passed to other dssi4cs opcodes.

ilibraryname - the name of the .so (shared object) file to load.

iplugindex - The index of the plugin to be used.

iverbose (optional) - show plugin information and parameters when loading. (default = 1)

dssiinit looks for *ilibraryname* on LADSPA_PATH and DSSI_PATH. One of these variables must be set, otherwise *dssiinit* will return an error. LADSPA and DSSI libraries may contain more than one plugin which must be referenced by its index. *dssiinit* then attempts to find plugin index *iplugindex* in the library and load the plugin into memory if it is found. To find out which plugins you have available and their index numbers you can use: *dssilist*.

If *iverbose* is not 0 (the default), information about the plugin detailing its characteristics and its ports will be shown. This information is important for opcodes like *dssictls*.

Plugins are set to inactive by default, so you **must** use *dssiactivate* to get the plugin to produce sound. This is required even if the plugin doesn't provide an activate() function.

dssiinit may cause audio stream breakups when used in realtime, so it is recommended to load all plugins to be used before playing.

Examples

Here is an example of the dssinit opcode. It uses the file *dssi4cs.csd* [examples/dssi4cs.csd].

Exemple 133. Example of the dssiinit opcode. (Remember to change the Library name)

```
<CsoundSynthesizer>
<CsOptions>
;use appropriate realtime options
</CsOptions>
<CsInstruments>
ksmps = 256
nchnls = 2
```

```
dssilist

gihandle dssiinit "amp.so", 0, 1
;gihandle dssiinit "cmt.so", 30 , 2
;gihandle2 dssiinit "cmt.so", 8 , 1
;gihandle dssiinit "delayorama_1402", 0
gihandle2 dssiinit "cmt.so", 49 , 1
;gihandle dssiinit "freq_tracker_1418.so", 0 , 1, 1
;gihandle dssiinit "g2reverb.so", 0, 1
;gihandle2 dssiinit "declip_1195.so", 0, 1
;gihandle2 dssiinit "revdelay_1605.so", 0, 1
;gihandle2 dssiinit "tap_chorusflanger.so", 0, 1
;gihandle2 dssiinit "plate_1423.so", 0, 1
gihandle3 dssiinit "gate_1410.so", 0, 1
;gihandle3 dssiinit "hexter.so", 0, 1

instr 1
print p4
dssiactivate gihandle, p4
dssiactivate gihandle2, p4
dssiactivate gihandle3, p4
endin


instr 2
ain1 inch 1
ain2 inch 2
;aout1,aout2 dssiaudio gihandle, ain1, ain2
aout1 dssiaudio gihandle, ain1
outs aout1,aout1
endin

instr 3
kval linen 1, p3 /3, p3, p3/ 3
dssictls gihandle, p4, kval, 1
endin

instr 4
ain1 inch 1
aout1 dssiaudio gihandle2, ain1
outs aout1,aout1
endin

</CsInstruments>
<CsScore>

i 1 1 1 1

i 2 2 15 ;plugin 1

i 3 3 12 0  ;Control port 0

i 4 8 2 ;plugin 2
e
</CsScore>
</CsoundSynthesizer>
```

Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

dssilist

dssilist — Lists all available DSSI and LADSPA plugins.

Syntax

```
dssilist
```

Description

dssilist checks the variables DSSI_PATH and LADSPA_PATH and lists all plugins available in all plugin libraries there.

LADSPA and DSSI libraries may contain more than one plugin which must be referenced by the index provided by *dssilist*.

This opcode produces a long printout which may interrupt realtime audio output, so it should be run at the start of a performance.

Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

dumpk

dumpk — Periodically writes an orchestra control-signal value to an external file.

Description

Periodically writes an orchestra control-signal value to a named external file in a specific format.

Syntax

```
dumpk  ksig, ifilname, iformat, iprd
```

Initialization

ifilname -- character string (in double quotes, spaces permitted) denoting the external file name. May either be a full path name with target directory specified or a simple filename to be created within the current directory

iformat -- specifies the output data format:

- 1 = 8-bit signed char(high order 8 bits of a 16-bit integer
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers
- 8 = ASCII floats (2 decimal places)

Note that A-law and U-law output are not available, and that all formats except the last two are binary. The output file contains no header information.

iprd -- the period of *ksig* output i seconds, rounded to the nearest orchestra control period. A value of 0 implies one control period (the enforced minimum), which will create an output file sampled at the orchestra control rate.

Performance

ksig -- a control-rate signal

This opcode allows a generated control signal value to be saved in a named external file. The file contains no self-defining header information. But it contains a regularly sampled time series, suitable for later input or analysis. There may be any number of *dumpk* opcodes in an instrument or orchestra but each must write to a different file.

Examples

Here is an example of the dumpk opcode. It uses the file *dumpk.csd* [examples/dumpk.csd].

Exemple 134. Example of the dumpk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dumpk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 20
nchnls = 1

; By Andres Cabrera 2008

instr 1
; Write fibonacci numbers to file "fibonacci.txt"
; as ascii long integers (mode 7), using the orchestra's
; control rate (iprd = 0)

knumber init 0
koldnumber init 1
ktrans init 1
ktrans = knumber
knumber = knumber + koldnumber
koldnumber = ktrans
dumpk  knumber, "fibonacci.txt", 7, 0
printk2 knumber
endin


</CsInstruments>
<CsScore>

;Write to the file for 1 second. Since control rate is 20, 20 values will be written
 i 1 0 1


</CsScore>
</CsoundSynthesizer>
```

See Also

dumpk2, *dumpk3*, *dumpk4*, *readk*, *readk2*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

dumpk2

dumpk2 — Periodically writes two orchestra control-signal values to an external file.

Description

Periodically writes two orchestra control-signal values to a named external file in a specific format.

Syntax

```
dumpk2 ksig1, ksig2, ifilname, iformat, iprd
```

Initialization

ifilname -- character string (in double quotes, spaces permitted) denoting the external file name. May either be a full path name with target directory specified or a simple filename to be created within the current directory

iformat -- specifies the output data format:

- 1 = 8-bit signed char(high order 8 bits of a 16-bit integer
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers
- 8 = ASCII floats (2 decimal places)

Note that A-law and U-law output are not available, and that all formats except the last two are binary. The output file contains no header information.

iprd -- the period of *ksig* output i seconds, rounded to the nearest orchestra control period. A value of 0 implies one control period (the enforced minimum), which will create an output file sampled at the orchestra control rate.

Performance

ksig1, ksig2 -- control-rate signals.

This opcode allows two generated control signal values to be saved in a named external file. The file contains no self-defining header information. But it contains a regularly sampled time series, suitable for later input or analysis. There may be any number of *dumpk2* opcodes in an instrument or orchestra but each must write to a different file.

Examples

See the example for *dumpk*. The only difference between *dumpk* and *dumpk2* is that *dumpk2* can write

two values at a time from the file.

See Also

dumpk, *dumpk3*, *dumpk4*, *readk*, *readk2*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

dumpk3

dumpk3 — Periodically writes three orchestra control-signal values to an external file.

Description

Periodically writes three orchestra control-signal values to a named external file in a specific format.

Syntax

```
dumpk3 ksig1, ksig2, ksig3, ifilname, iformat, iprd
```

Initialization

ifilname -- character string (in double quotes, spaces permitted) denoting the external file name. May either be a full path name with target directory specified or a simple filename to be created within the current directory

iformat -- specifies the output data format:

- 1 = 8-bit signed char(high order 8 bits of a 16-bit integer
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers
- 8 = ASCII floats (2 decimal places)

Note that A-law and U-law output are not available, and that all formats except the last two are binary. The output file contains no header information.

iprd -- the period of *ksig* output i seconds, rounded to the nearest orchestra control period. A value of 0 implies one control period (the enforced minimum), which will create an output file sampled at the orchestra control rate.

Performance

ksig1, ksig2, ksig3 -- control-rate signals

This opcode allows three generated control signal values to be saved in a named external file. The file contains no self-defining header information. But it contains a regularly sampled time series, suitable for later input or analysis. There may be any number of *dumpk3* opcodes in an instrument or orchestra but each must write to a different file.

Examples

See the example for *dumpk*. The only difference between *dumpk* and *dumpk3* is that *dumpk3* can write

three values at a time from the file.

See Also

dumpk, *dumpk2*, *dumpk4*, *readk*, *readk2*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

dumpk4

dumpk4 — Periodically writes four orchestra control-signal values to an external file.

Description

Periodically writes four orchestra control-signal values to a named external file in a specific format.

Syntax

```
dumpk4 ksig1, ksig2, ksig3, ksig4, ifilname, iformat, iprd
```

Initialization

ifilname -- character string (in double quotes, spaces permitted) denoting the external file name. May either be a full path name with target directory specified or a simple filename to be created within the current directory

iformat -- specifies the output data format:

- 1 = 8-bit signed char(high order 8 bits of a 16-bit integer
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers
- 8 = ASCII floats (2 decimal places)

Note that A-law and U-law output are not available, and that all formats except the last two are binary. The output file contains no header information.

iprd -- the period of *ksig* output i seconds, rounded to the nearest orchestra control period. A value of 0 implies one control period (the enforced minimum), which will create an output file sampled at the orchestra control rate.

Performance

ksig1, ksig2, ksig3, ksig4 -- control-rate signals

This opcode allows four generated control signal values to be saved in a named external file. The file contains no self-defining header information. But it contains a regularly sampled time series, suitable for later input or analysis. There may be any number of *dumpk4* opcodes in an instrument or orchestra but each must write to a different file.

Examples

See the example for *dumpk*. The only difference between *dumpk* and *dumpk4* is that *dumpk4* can write

four values at a time from the file.

See Also

dumpk, *dumpk2*, *dumpk3*, *readk*, *readk2*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

duserrnd

duserrnd — Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

Description

Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

Syntaxe

```
aout duserrnd ktableNum
```

```
iout duserrnd itableNum
```

```
kout duserrnd ktableNum
```

Initialisation

itableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

Exécution

ktableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

duserrnd (Discrete USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire discrète créée par l'utilisateur. L'utilisateur peut créer l'histogramme de la distribution discrète au moyen de GEN41. Afin de créer cette table, on doit définir une quantité arbitraire de couples de nombres, le premier nombre de chaque paire représentant une valeur et le second représentant sa probabilité (voir GEN41 pour plus de détails).

Lorsqu'on l'utilise comme une fonction, le taux de génération dépend du type du taux de la variable d'entrée *XtableNum*. Dans ce cas, on peut l'insérer dans n'importe quelle formule. Le numéro de table peut varier au taux-k, ce qui permet de changer l'histogramme de la distribution durant l'exécution d'une note. *duserrnd* est destiné à être utilisé pour la génération de musique algorithmique.

On peut aussi utiliser *duserrnd* pour générer des valeurs suivant un ensemble d'intervalles de probabilités au moyen de fonctions de distribution générées par GEN42 (voir GEN42 pour plus de détails). Dans ce cas, si l'on veut simuler des intervalles continus, la longueur de la table *XtableNum* doit être raisonnablement grande car *duserrnd* ne fait pas d'interpolation entre les éléments de la table.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Voir Aussi

cuserrnd, *urd*

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

else

else — Executes a block of code when an "if...then" condition is false.

Description

Executes a block of code when an "if...then" condition is false.

Syntax

```
else
```

Performance

else is used inside of a block of code between the *"if...then"* and *endif* opcodes. It defines which statements are executed when a "if...then" condition is false. Only one *else* statement may occur and it must be the last conditional statement before the *endif* opcode.

Examples

See the example for the *if* opcode.

See Also

elseif, *endif*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timout*

Credits

New in version 4.21

elseif

elseif — Defines another "if...then" condition when a "if...then" condition is false.

Description

Defines another "if...then" condition when a "if...then" condition is false.

Syntax

```
elseif xa R xb then
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Performance

elseif is used inside of a block of code between the *"if...then"* and *endif* opcodes. When a "if...then" condition is false, it defines another "if...then" condition to be met. Any number of *elseif* statements are allowed.

Examples

See the example for the *if* opcode.

See Also

else, *endif*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timout*

Credits

New in version 4.21

endif

endif — Closes a block of code that begins with an "if...then" statement.

Description

Closes a block of code that begins with an *"if...then"* statement.

Syntax

```
endif
```

Performance

Any block of code that begins with an *"if...then"* statement must end with an *endif* statement.

Examples

See the example for the *if* opcode.

See Also

elseif, *else*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timout*

Credits

New in version 4.21

endin

endin — Termine un bloc d'instrument.

Description

Termine le bloc d'instrument courant.

Syntax

```
endin
```

Initialisation

Termine le bloc d'instrument courant.

On peut définir les instruments dans n'importe quel ordre (mais ils seronts toujours initialisés et exécutés par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).

Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

Exemples

Voici un exemple de l'opcode *endin*. Il utilise le fichier *endin.csd* [examples/endin.csd].

Exemple 135. Exemple de l'opcode *endin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o endin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0
```

```
  a1 oscils iamp, icps, iphs
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

instr

Crédits

Exemple écrit par Kevin Conder.

endop

endop — Termine un bloc d'opcode défini par l'utilisateur.

Description

Termine un bloc d'opcode défini par l'utilisateur.

Syntaxe

```
endop
```

Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode  nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
[setksmps  iksmps]
... the rest of the instrument's code.
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN]  nom  [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

Exemples

Voir l'exemple pour *opcode*.

Voir Aussi

opcode, *setksmps*, *xin*, *xout*

Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

envlpx

envlpx — Applique une enveloppe constituée de 3 segments.

Description

envlpx -- applique une enveloppe constituée de 3 segments :

1. une attaque dont la forme est donnée par une fonction
2. un pseudo entretien modifié exponentiellement
3. une chute exponentielle

Syntaxe

```
ares envlpx xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
```

```
kres envlpx kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
```

Initialisation

irise -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

idur -- durée globale en seconde. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

idec -- durée de la chute en secondes. Zéro signifie pas de chute. Si *idec* > *idur* la chute sera tronquée.

ifn -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

iatss -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpx* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs*(*iatss*) par seconde sera utilisée. 0 est interdit.

iatdec -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

ixmod (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

Exécution

kamp, xamp -- amplitude du signal d'entrée.

Les modifications de l'attaque sont appliquées pendant les premières *irise* secondes, et celles de la chute à partir de *idur* - *idec*. Si ces périodes sont séparées dans le temps il y aura un entretien au cours duquel *amp* sera modifié selon le schéma exponentiel décrit. Si l'attaque et la chute se chevauchent alors les deux modifications agiront simultanément durant cette période commune. Si la durée globale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, en tendant asymtotiquement vers zéro.

Exemples

Voici un exemple de l'opcode envlpx. Il utilise le fichier *envlpx.csd* [examples/envlpx.csd].

Exemple 136. Exemple de l'opcode envlpx.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o envlpx.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
  ; Set the amplitude.
  kamp init 20000
  ; Get the frequency from the fourth p-field.
  kcps = cpspch(p4)

  a1 vco kamp, kcps, 1
  out a1
endin

; Instrument #2 - instrument with an amplitude envelope.
instr 2
  kamp = 20000
  irise = 0.05
  idur = p3 - .01
  idec = 0.5
  ifn = 2
  iatss = 1
  iatdec = 0.01

  ; Create an amplitude envelope.
  kenv envlpx kamp, irise, idur, idec, ifn, iatss, iatdec

  ; Get the frequency from the fourth p-field.
  kcps = cpspch(p4)

  a1 vco kenv, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1
; Table #2, a rising envelope.
```

```
f 2 0 129 -7 0 128 1

; Set the tempo to 120 beats per minute.
t 0 120

; Make sure the score plays for 33 seconds.
f 0 33

; Play a melody with Instrument #1.
; p4 = frequency in pitch-class notation.
i  1  0   1  8.04
i  1  1   1  8.04
i  1  2   1  8.05
i  1  3   1  8.07
i  1  4   1  8.07
i  1  5   1  8.05
i  1  6   1  8.04
i  1  7   1  8.02
i  1  8   1  8.00
i  1  9   1  8.00
i  1  10  1  8.02
i  1  11  1  8.04
i  1  12  2  8.04
i  1  14  2  8.02

; Repeat the melody with Instrument #2.
; p4 = frequency in pitch-class notation.
i  2  16  1  8.04
i  2  17  1  8.04
i  2  18  1  8.05
i  2  19  1  8.07
i  2  20  1  8.07
i  2  21  1  8.05
i  2  22  1  8.04
i  2  23  1  8.02
i  2  24  1  8.00
i  2  25  1  8.00
i  2  26  1  8.02
i  2  27  1  8.04
i  2  28  2  8.04
i  2  30  2  8.02
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

envlpxr, *linen*, *linenr*

Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.

Exemple écrit par Kevin Conder.

envlpxr

envlpxr — L'opcode *envlpx* avec un segment final de relâchement.

Description

envlpxr est le même que *envlpx* sauf que le segment final n'est exécuté qu'après un évènement MIDI de relâchement de note. La note est ensuite allongée de la durée de la chute.

Syntaxe

```
ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
```

```
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
```

Initialisation

irise -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

idec -- durée de la chute en secondes. Zéro signifie pas de chute.

ifn -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

iatss -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpxr* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs*(*iatss*) par seconde sera utilisée. 0 est interdit.

iatdec -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

ixmod (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

irind (facultatif) -- indicateur d'indépendance. S'il est nul, la durée de relâchement (*idec*) aura une influence sur l'allongement de la note après un note-off. S'il est non nul, la durée *idec* sera relativement indépendante de l'allongement de la note (voir ci-dessous). La valeur par défaut est 0.

Exécution

kamp, xamp -- amplitude du signal d'entrée.

envlpxr fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un évènement ou MIDI note-off est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *idec* secondes à moins qu'il ne soit rendu indépendant par *irind*. Dans ce cas, la chute démarrera de l'endroit, quelqu'il soit, où l'on se trou-

vait à ce moment précis.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *envlpxr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des évènements MIDI note-off provoqués par une vélocité nulle. Si l'indicateur *irind* est positionné (différent de zéro), la durée d'exécution totale n'est pas affectée par les données de note-off ou de vélocité nulle.

Unités « r » multiples. Quand plusieurs unités « r » sont présentes dans le même instrument, il est habituel qu'une seule d'entre elle influence la durée totale de la note. C'est normalement l'unité contrôlant l'amplitude principale de la note. D'autres unités contrôlant par exemple l'évolution d'un filtre, peuvent toujours être sensibles aux commandes note-off tout en n'affectant pas la durée grâce à leur indépendance (*irind* non nul). En fonction de leur propre valeur *idec* (durée de relâchement), les unités « r » indépendantes pourront ou ne pourront pas atteindre leur destination finale avant que la note ne se termine. Si elles y arrivent, elles tiendront simplement leur dernière valeur jusqu'à la fin. Si plusieurs unités « r » sont principales, l'extension de la note sera celle de la plus grande valeur *idec*.

Voir Aussi

envlpx, *linen*, *linenr*

Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.

ephasor

ephasor —

Performance

Ephasor has been added to Csound 5.10, but its behavior will change for 5.11. Stay tuned...

Credits

Author: Victor Lazzarini
2008

New in version 5.10

eqfil

eqfil — Equalizer filter

Description

The opcode eqfil is a 2nd order tunable equalisation filter based on Regalia and Mitra design ("Tunable Digital Frequency Response Equalization Filters", IEEE Trans. on Ac., Sp. and Sig Proc., 35 (1), 1987). It provides a peak/notch filter for building parametric/graphic equalisers.

The amplitude response for this filter will be flat (=1) for kgain=1. With kgain is bigger than 1, there will be a peak at the centre frequency, whose width is given by the kbw parameter, but outside this band, the response will tend towards 1. Conversely, if kgain is smaller than 1, a notch will be created around the CF.

Syntax

```
asig eqfil ain, kcf, kbw, kgain[, istor]
```

Initialization

istor --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- filtered output signal.

ain -- input signal.

kcf -- filter centre frequency

kbw -- peak/notch bandwidth (Hz).

kgain -- peak/notch gain.

Examples

Exemple 137. Example

```
kfe         expseg 10, p3*0.9, 180, p3*0.1, 175
kenv        linen  1000, 0.05, p3, 0.05
asig        buzz   kenv, kfe, sr/(2*kfe), 1
afil        eqfil  asig, 1500, 400,  0.1

            out afil
```

Credits

Author: Victor Lazzarini;
April 2007

New in version 5.06

event

event — Generates a score event from an instrument.

Description

Generates a score event from an instrument.

Syntax

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
```

```
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]
```

Initialization

« scorechar » -- A string (in double-quotes) representing the first p-field in a score statement. This is usually *« e »*, *« f »*, or *« i »*.

« insname » -- A string (in double-quotes) representing a named instrument.

Performance

kinsnum -- The instrument to use for the event. This corresponds to the first p-field, p1, in a score statement.

kdelay -- When (in seconds) the event will occur from the current performance time. This corresponds to the second p-field, p2, in a score statement.

kdur -- How long (in seconds) the event will happen. This corresponds to the third p-field, p3, in a score statement.

kp4, kp5, ... (optional) -- Parameters representing additional p-field in a score statement. It starts with the fourth p-field, p4.

Examples

Here is an example of the event opcode. It uses the file *event.csd* [examples/event.csd].

Exemple 138. Example of the event opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o event.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
  ; Create a trigger and set its initial value to 1.
  ktrigger init 1

  ; If the trigger is equal to 0, continue playing.
  ; If not, schedule another event.
  if (ktrigger == 0) goto contin
    ; kscoreop="i", an i-statement.
    ; kinsnum=2, play Instrument #2.
    ; kwhen=1, start at 1 second.
    ; kdur=0.5, play for a half-second.
    event "i", 2, 1, 0.5

    ; Make sure the event isn't triggered again.
    ktrigger = 0

contin:
  a1 oscils 10000, 440, 1
  out a1
endin

; Instrument #2 - an oscillator with a low note.
instr 2
  a1 oscils 10000, 220, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e


</CsScore>
</CsoundSynthesizer>
```

Here is an example of the event opcode using a named instrument. It uses the file *event_named.csd* [examples/event_named.csd].

Exemple 139. Example of the event opcode using a named instrument.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o event_named.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
```

```
  ; Create a trigger and set its initial value to 1.
  ktrigger init 1

  ; If the trigger is equal to 0, continue playing.
  ; If not, schedule another event.
  if (ktrigger == 0) goto contin
    ; kscoreop="i", an i-statement.
    ; kinsnum="low_note", instrument named "low_note".
    ; kwhen=1, start at 1 second.
    ; kdur=0.5, play for a half-second.
    event "i", "low_note", 1, 0.5

    ; Make sure the event isn't triggered again.
    ktrigger = 0

contin:
  a1 oscils 10000, 440, 1
  out a1
endin

; Instrument "low_note" - an oscillator with a low note.
instr low_note
  a1 oscils 10000, 220, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Examples written by Kevin Conder.

New in version 4.17

Thanks goes to Matt Ingalls for helping to fix the example.

Thanks goes to Matt Ingalls for helping clarify the kwhen/kdelay parameter.

event_i

event_i — Génère un évènement de partition à partir d'un instrument.

Description

Génère un évènement de partition à partir d'un instrument.

Syntaxe

```
event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
```

```
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]
```

Initialisation

« scorechar » -- Une chaîne de caractères (entre guillemets) représentant le premier p-champ d'une instruction de partition. C'est habituellement *« e »*, *« f »*, ou *« i »*.

« insname » -- Une chaîne de caractères (entre guillemets) représentant un instrument par son nom.

iinsnum -- L'instrument à utiliser pour cet évènement. Cela correspond au premier p-champ, p1, dans une instruction de partition.

idelay -- Le moment (en secondes) à partir de la date courante d'exécution où l'évènement se produira. Cela correspond au deuxième p-champ, p2, dans une instruction de partition.

idur -- La durée de l'évènement (en secondes). Cela correspond au troisième p-champ, p3, dans une instruction de partition.

ip4, ip5, ... (facultatif) -- Paramètres représentant des p-champs supplémentaires dans une instruction de partition. Ca commence avec le quatrième p-champ, p4.

Exécution

L'évènement est ajouté à la file d'attente pendant la période d'initialisation.

Crédits

Ecrit par Istvan Varga.

Nouveau dans Csound5

exitnow

exitnow — Exit csound as fast as possible, with no cleaning up.

Description

In Csound4 calls an exit function to leave csoudn as fast as possible. On Csound5 exits back to the driving code.

Syntax

```
exitnow
```

Performance

Stops Csound on the initialisation cycle.

exp

exp — Retourne e élevé à la puissance *x*.

Description

Retourne e élevé à la puissance *x*.

Syntaxe

```
exp(x) (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode exp. Il utilise le fichier *exp.csd* [examples/exp.csd].

Exemple 140. Exemple de l'opcode exp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o exp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = exp(8)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 2980.958
```

Voir Aussi

abs, *frac*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

expcurve

expcurve — Cet opcode implémente une formule qui génère une courbe exponentielle normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

Description

Génère une courbe exponentielle dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lèvera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

```
(exp(x * log(y))-1) / (y-1)
```

où x est égal à *kindex* et y est égal à *ksteepness*.

Syntaxe

```
kout expcurve kindex, ksteepness
```

Exécution

kindex -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

ksteepness -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

kout -- Sortie pondérée.

Exemples

Voici un exemple de l'opcode expcurve. Il utilise le fichier *expcurve.csd* [examples/expcurve.csd].

Exemple 141. Exemple de l'opcode expcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent
-odac          -idac      -d     ;;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

/*--- ---*/

                instr 1 ; logcurve test

kmod phasor 1/200
```

```
kout expcurve kmod, 2

        printk2 kmod
        printk2 kout

                endin

/*--- ---*/
</CsInstruments>
<CsScore>

i1 0 8888

e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

scale, *gainslider*, *logcurve*

Crédits

Auteur : David Akbari
Octobre
2006

expon

expon — Trace une courbe exponentielle entre les points spécifiés.

Description

Trace une courbe exponentielle entre les points spécifiés.

Syntaxe

```
ares expon ia, idur, ib
```

```
kres expon ia, idur, ib
```

Initialisation

ia -- valeur initiale. Zéro est interdit pour les exponentielles.

ib -- valeur après *idur* secondes. Pour les exponentielles, doit être non nulle et du même signe que *ia*.

idur -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par deux points spécifiés. La valeur de *idur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le segment continuera dans la même direction.

Exemples

Voici un exemple de l'opcode expon. Il utilise le fichier *expon.csd* [examples/expon.csd].

Exemple 142. Exemple de l'opcode expon.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expon.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Define kcps as a frequency value that exponentially declines
  ; from 880 to 220. It declines over the period set by p3.
  kcps expon 880, p3, 220

  a1 oscil 20000, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

expseg, *expsegr*, *line*, *linseg*, *linsegr*

Crédits

Exemple écrit par Kevin Conder.

exprand

exprand — Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares exprand klambda
```

```
ires exprand klambda
```

```
kres exprand klambda
```

Exécution

klambda -- paramètre lambda pour la distribution exponentielle.

La fonction de densité de probabilité d'une distribution exponentielle est une courbe exponentielle, dont la moyenne est 0,69515/lambda. Pour des explications plus détaillées de ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode exprand. Il utilise le fichier *exprand.csd* [examples/exprand.csd].

Exemple 143. Exemple de l'opcode exprand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o exprand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random between 0 and 1.
  ; krange = 1

  i1 exprand 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.174
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur: Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

expseg

expseg — Trace une suite de segments d'exponentielle entre les points spécifiés.

Description

Trace une suite de segments d'exponentielle entre les points spécifiés.

Syntaxe

```
ares expseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
kres expseg ia, idur1, ib [, idur2] [, ic] [...]
```

Initialisation

ia -- valeur initiale. Zéro est interdit pour les exponentielles.

ib, ic, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

idur1 -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

idur2, idur3, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

Noter que l'opcode *expseg* n'opère pas correctement au taux audio lorsque les segments sont plus courts qu'une k-période. Dans ce cas, il vaut mieux utiliser l'opcode *expsega*.

Exemples

Voici un exemple de l'opcode expseg. Il utilise le fichier *expseg.csd* [examples/expseg.csd].

Exemple 144. Exemple de l'opcode expseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
```

```
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expseg.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Create an amplitude envelope.
  kenv expseg 0.01, p3*0.25, 1, p3*0.75, 0.01
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

expon, *expsega*, *expsegr*, *line*, *linseg*, *linsegr transeg*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57

expsega

expsega — Un générateur de segments exponentiels opérant au taux-a.

Description

Un générateur de segments exponentiels opérant au taux-a. Cette unité est pratiquement identique à *expseg*, mais elle est plus précise lorsque l'on définit des segments de courte durée (c-à-d., dans une phase d'attaque percussive) au taux audio.

Syntaxe

```
ares expsega ia, idur1, ib [, idur2] [, ic] [...]
```

Initialiation

ia -- valeur initiale. Zéro est interdit.

ib, *ic*, etc. -- valeur après *idur1* secondes, etc. Doivent être non nulles et de même signe que *ia*.

idur1 -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

idur2, idur3, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

Exécution

Cette unité génère des signaux audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

Exemples

Voici une exemple de l'opcode expsega. Il utilise le fichier *expsega.csd* [examples/expsega.csd].

Exemple 145. Exemple de l'opcode expsega.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expsega.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Define a short percussive amplitude envelope that
  ; goes from 0.01 to 20,000 and back.
  aenv expsega 0.01, 0.1, 20000, 0.1, 0.01

  a1 oscil aenv, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
; Play Instrument #1 for one second.
i 1 2 1
; Play Instrument #1 for one second.
i 1 3 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

expseg, *expsegr*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57

expsegr

expsegr — Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement.

Description

Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement (fin de l'entretien de la note).

Syntaxe

```
ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

Initialisation

ia -- valeur initiale. Zéro est interdit pour les exponentielles.

ib, ic, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

idur1 -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

idur2, idur3, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

irel, iz -- durée en secondes et valeur finale du segment de relâchement de la note.

Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

expsegr fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un évènement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vélocités nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *madsr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *expsegr*, car la durée est allongée automatiquement.

Exemples

Voici un exemple de l'opcode expsegr. Il utilise le fichier *expsegr.csd* [examples/expsegr.csd].

Exemple 146. Exemple de l'opcode expsegr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expsegr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Use an amplitude envelope with second-long release.
  kenv expsegr 0.01, p3/2, 1, p3/2, 0.01, 1, 1
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Make sure the score lasts for four seconds.
f 0 4

; p4 = frequency (in pitch-class notation).
; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

linsegr, *expsegr*, *envlpxr*, *mxadsr*, *madsr expon*, *expseg*, *expsega*, *xtratim*

Crédits

Auteur : Barry L. Vercoe

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.47

ficlose

ficlose — Closes a previously opened file.

Description

ficlose can be used to close a file which was opened with *fiopen*.

Syntax

```
ficlose ihandle

ficlose Sfilename
```

Initialization

ihandle -- a number which identifies this file (generated by a previous *fiopen*).

Sfilename -- A string in double quotes or string variable with the filename. The full path must be given if the file directory is not in the system PATH and is not present in the cuurrent directory.

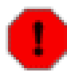
Performance

ficlose closes a file which was previously opened with *fiopen*. *ficlose* is only needed if you need to read a file written to during the same csound performance, since only when csound ends a performance does it close and save data in all open files. The opcode *ficlose* is useful for instance if you want to save presets within files which you want to be accesible without having to terminate csound.

Note

If you don't need this functionality it is safer not to call *ficlose*, and just let csound close the files when it exits.

If a files closed with *ficlose* is being accessed by another opcode (like *fout* or *foutk*, it will be closed later when it is no longer being used.



Avertissement

This opcode should be used with care, as the file handle will become invalid, and will cause an init error when an opcode tries to access the closed file.

See Also

fout, *fout*, *fouti*, *foutir*, *foutk*

Credits

Author: Gabriel Maldonado

Italy
1999

New in Csound version 5.02

filelen

filelen — Returns the length of a sound file.

Description

Returns the length of a sound file.

Syntax

```
ir filelen ifilcod, [iallowraw]
```

Initialization

ifilcod -- sound file to be queried

iallowraw -- Allow raw sound files (default=1)

Performance

filelen returns the length of the sound file *ifilcod* in seconds. *filelen* can return the length of convolve and PVOC files if the "allow raw sound file" flag is not zero (it is non-zero by default).

Examples

Here is an example of the filelen opcode. It uses the file *filelen.csd* [examples/filelen.csd], and *mary.wav* [examples/mary.wav].

Exemple 147. Example of the filelen opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filelen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the length of the audio file
  ; "mary.wav" in seconds.
  ilen  filelen "mary.wav"
  print ilen
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The audio file « mary.wav » is 3.5 seconds long. So *filelen*'s output should include a line like this:

```
instr 1:  ilen = 3.501
```

See Also

filenchnls, *filepeak*, *filesr*

Credits

Author: Matt Ingalls
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

filenchnls

filenchnls — Returns the number of channels in a sound file.

Description

Returns the number of channels in a sound file.

Syntax

```
ir filenchnls ifilcod [, iallowraw]
```

Initialization

ifilcod -- sound file to be queried

iallowraw -- (Optional) Allow raw sound files (default=1)

Performance

filenchnls returns the number of channels in the sound file *ifilcod*. *filechnls* can return the number of channels of convolve and PVOC files if the *iallowraw* flag is not zero (it is non-zero by default).

Examples

Here is an example of the filenchnls opcode. It uses the file *filenchnls.csd* [examples/filenchnls.csd], and *mary.wav* [examples/mary.wav].

Exemple 148. Example of the filenchnls opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filenchnls.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the number of channels in the
  ; audio file "mary.wav".
  ichnls filenchnls "mary.wav"
  print ichnls
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The audio file « mary.wav » is monoaural (1 channel). So *filenchnls*'s output should include a line like this:

```
instr 1:  ichnls = 1.000
```

See Also

filelen, *filepeak*, *filesr*

Credits

Author: Matt Ingalls
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

filepeak

filepeak — Returns the peak absolute value of a sound file.

Description

Returns the peak absolute value of a sound file.

Syntax

```
ir filepeak ifilcod [, ichnl]
```

Initialization

ifilcod -- sound file to be queried

ichnl (optional, default=0) -- channel to be used in calculating the peak value. Default is 0.

- *ichnl* = 0 returns peak value of all channels
- *ichnl* > 0 returns peak value of *ichnl*

Performance

filepeak returns the peak absolute value of the sound file *ifilcod*.

Examples

Here is an example of the filepeak opcode. It uses the file *filepeak.csd* [examples/filepeak.csd], and *mary.wav* [examples/mary.wav].

Exemple 149. Example of the filepeak opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filepeak.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Print out the peak absolute value of the
  ; audio file "mary.wav".
  ipeak filepeak "mary.wav"
  print ipeak
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The peak absolute value of the audio file « mary.wav » is 0.306902. So *filepeak*'s output should include a line like this:

```
instr 1:  ipeak = 0.307
```

See Also

filelen, *filenchnls*, *filesr*

Credits

Author: Matt Ingalls
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

filesr

filesr — Returns the sample rate of a sound file.

Description

Returns the sample rate of a sound file.

Syntax

```
ir filesr ifilcod [, iallowraw]
```

Initialization

ifilcod -- sound file to be queried

iallowraw -- (Optional) Allow raw sound files (default=1)

Performance

filesr returns the sample rate of the sound file *ifilcod*. *filesr* can return the sample rate of convolve and PVOC files if the *iallowraw* flag is not zero (it is non-zero by default).

Examples

Here is an example of the filesr opcode. It uses the file *filesr.csd* [examples/filesr.csd], and *mary.wav* [examples/mary.wav].

Exemple 150. Example of the filesr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filesr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the sampling rate of the
  ; audio file "mary.wav".
  isr filesr "mary.wav"
  print isr
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The audio file « mary.wav » was sampled at 44.1 KHz. So *filesr*'s output should include a line like this:

```
instr 1:  isr = 44100.000
```

See Also

filelen, *filenchnls*, *filepeak*

Credits

Author: Matt Ingalls
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

filter2

filter2 — Performs filtering using a transposed form-II digital filter lattice with no time-varying control.

Description

General purpose custom filter with time-varying pole control. The filter coefficients implement the following difference equation:

(1)*y(n) = b0*x[n] + b1*x[n-1] +...+ bM*x[n-M] - a1*y[n-1] -...- aN*y[n-N]

the system function for which is represented by:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + ... + bM*Z^{-M}}{1 + a1*Z^{-1} + ... + aN*Z^{-N}}$$

Syntax

```
ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

```
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

Initialization

At initialization the number of zeros and poles of the filter are specified along with the corresponding zero and pole coefficients. The coefficients must be obtained by an external filter-design application such as Matlab and specified directly or loaded into a table via *GEN01*.

Performance

The *filter2* opcodes perform filtering using a transposed form-II digital filter lattice with no time-varying control.

Since *filter2* implements generalized recursive filters, it can be used to specify a large range of general DSP algorithms. For example, a digital waveguide can be implemented for musical instrument modeling using a pair of *delayr* and *delayw* opcodes in conjunction with the *filter2* opcode.

Examples

A first-order linear-phase lowpass linear-phase FIR filter operating on a k-rate signal:

```
k1 filter2 ksig, 2, 0, 0.5, 0.5   ;; k-rate FIR filter
```

See Also

zfilter2

Credits

Author: Michael A. Casey
M.I.T.
Cambridge, Mass.
1997

New in version 3.47

fin

fin — Read signals from a file at a-rate.

Description

Read signals from a file at a-rate.

Syntax

```
fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [,...]
```

Initialization

ifilename -- input file name (can be a string or a handle number generated by fiopen)

iskipframes -- number of frames to skip at the start (every frame contains a sample of each channel)

iformat -- a number specifying the input file format for headerless files. If a header is found, this argument is ignored.

- 0 - 32 bit floating points without header
- 1 - 16 bit integers without header

Performance

fin (file input) is the complement of *fout*: it reads a multichannel file to generate audio rate signals. The user must be sure that the number of channels of the input file is the same as the number of *ainX* arguments.

Note

Please note that since this opcode generates its output using input parameters (on the right side of the opcode), these variables must be initialized before use, otherwise a 'used before defined' error will occur. You can use the *init* opcode for this.

See Also

fini, *fink*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

fini

fini — Read signals from a file at i-rate.

Description

Read signals from a file at i-rate.

Syntax

```
fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]
```

Initialization

ifilename -- input file name (can be a string or a handle number generated by *fiopen*)

iskipframes -- number of frames to skip at the start (every frame contains a sample of each channel)

iformat -- a number specifying the input file format. If a header is found, this argument is ignored.

- 0 - floating points in text format (loop; see below)
- 1 - floating points in text format (no loop; see below)
- 2 - 32 bit floating points in binary format (no loop)

Performance

fini is the complement of *fouti* and *foutir*. It reads the values each time the corresponding instrument note is activated. When *iformat* is set to 0 and the end of file is reached, the file pointer is zeroed. This restarts the scan from the beginning. When *iformat* is set to 1 or 2, no looping is enabled and at the end of file the corresponding variables will be filled with zeroes.

Note

Please note that since this opcode generates its output using input parameters (on the right side of the opcode), these variables must be initialized before use, otherwise a 'used before defined' error will occur. You can use the *init* opcode for this.

See Also

fin, *fink*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

fink

fink — Read signals from a file at k-rate.

Description

Read signals from a file at k-rate.

Syntax

```
fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [,...]
```

Initialization

ifilename -- input file name (can be a string or a handle number generated by fiopen)

iskipframes -- number of frames to skip at the start (every frame contains a sample of each channel)

iformat -- a number specifying the input file format. If a header is found, this argument is ignored.

- 0 - 32 bit floating points without header
- 1 - 16 bit integers without header

Performance

fink is the same as *fin* but operates at k-rate.

Note

Please note that since this opcode generates its output using input parameters (on the right side of the opcode), these variables must be initialized before use, otherwise a 'used before defined' error will occur. You can use the *init* opcode for this.

See Also

fin, *fini*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

fiopen

fiopen — Opens a file in a specific mode.

Description

fiopen can be used to open a file in one of the specified modes.

Syntax

```
ihandle fiopen ifilename, imode
```

Initialization

ihandle -- a number which specifies this file.

ifilename -- the output file's name (in double-quotes).

imode -- choose the mode of opening the file. *imode* can be a value chosen among the following:

- 0 - open a text file for writing
- 1 - open a text file for reading
- 2 - open a binary file for writing
- 3 - open a binary file for reading

Performance

fiopen opens a file to be used by the *fout* family of opcodes. It is safer to use it in the header section, external to any instruments. It returns a number, *ihandle*, which unequivocally refers to the opened file.

If *fiopen* is called on an already open file, it just returns the same handle again, and does not close the file.

Notice that *fout* and *foutk* can use either a string containing a file pathname, or a handle-number generated by *fiopen*. Whereas, with *fouti* and *foutir*, the target file can be only specified by means of a handle-number.

See Also

ficlose *fout*, *fouti*, *foutir*, *foutk*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

flanger

flanger — A user controlled flanger.

Description

A user controlled flanger.

Syntax

```
ares flanger asig, adel, kfeedback [, imaxd]
```

Initialization

imaxd(optional) -- maximum delay in seconds (needed for inital memory allocation)

Performance

asig -- input signal

adel -- delay in seconds

kfeedback -- feedback amount (in normal tasks this should not exceed 1, even if bigger values are allowed)

This unit is useful for generating choruses and flangers. The delay must be varied at a-rate connecting *adel* to an oscillator output. Also the feedback can vary at k-rate. This opcode is implemented to allow *kr* different than *sr* (else delay could not be lower than *ksmps*) enhancing realtime performance. This unit is very similar to *wguide1*, the only difference is *flanger* does not have the lowpass filter.

Examples

Here is an example of the flanger opcode. It uses the file *flanger.csd* [examples/flanger.csd], and *beats.wav* [examples/beats.wav].

Exemple 151. Example of the flanger opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o flanger.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Use the "beat.wav" audio file.
  asig soundin "beats.wav"

  ; Vary the delay amount from 0 to 0.01 seconds.
  adel line 0, p3, 0.01
  kfeedback = 0.7

  ; Apply flange to the input signal.
  aflang flanger asig, adel, kfeedback

  ; It can get loud, so clip its amplitude to 30,000.
  a1 clip aflang, 1, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy

Example written by Kevin Conder.

New in Csound version 3.49

flashtxt

flashtxt — Allows text to be displayed from instruments like sliders

Description

Allows text to be displayed from instruments like sliders etc. (only on Unix and Windows at present)

Syntax

```
flashtxt  iwhich, String
```

Initialization

iwhich -- the number of the window.

String -- the string to be displayed.

Performance

A window is created, identified by the iwhich argument, with the text string displayed. If the text is replaced by a number then the window id deleted. Note that the text windows are globally numbered so different instruments can change the text, and the window survives the instance of the instrument.

Examples

Here is an example of the flashtxt opcode. It uses the file *flashtxt.csd* [examples/flashtxt.csd].

Exemple 152. Example of the flashtxt opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o flashtxt.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
  flashtxt 1, "Instr 1 live"
  ao oscil 4000, 440, 1
  out ao
endin
```

```
</CsInstruments>
<CsScore>

; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

FLbox

FLbox — A FLTK widget that displays text inside of a box.

Description

A FLTK widget that displays text inside of a box.

Syntax

```
ihandle FLbox "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]
```

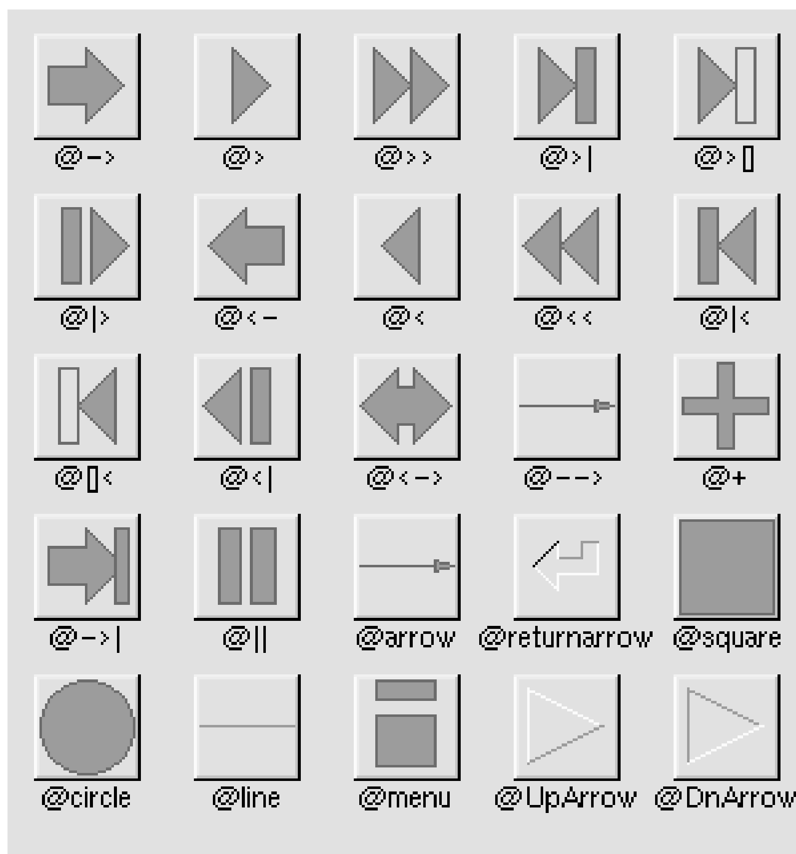
Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLbox* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near corresponding widget.

Notice that with *FLbox*, it is not necessary to call the *FLsetTextType* opcode at all in order to use a symbol. In this case, it is sufficient to set a label starting with « @ » followed by the proper formatting string.

The following symbols are supported:



FLTK label supported symbols.

The @ sign may be followed by the following optional « formatting » characters, in this order:

1. « # » forces square scaling rather than distortion to the widget's shape.
2. +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
3. [1-9] rotates by a multiple of 45 degrees. « 6 » does nothing, the others point in the direction of that key on a numeric keypad.

itype -- an integer number denoting the appearance of the widget.

The following values are legal for *itype*:

- 1 - flat box
- 2 - up box
- 3 - down box
- 4 - thin up box
- 5 - thin down box
- 6 - engraved box

- 7 - embossed box
- 8 - border box
- 9 - shadow box
- 10 - rounded box
- 11 - rounded box with shadow
- 12 - rounded flat box
- 13 - rounded up box
- 14 - rounded down box
- 15 - diamond up box
- 16 - diamond down box
- 17 - oval box
- 18 - oval shadow box
- 19 - oval flat box

ifont -- an integer number denoting the font of *FLbox*.

ifont argument to set the font type. The following values are legal for *ifont*:

- 1 - helvetica (same as "Arial" under Windows)
- 2 - helvetica bold
- 3 - helvetica italic
- 4 - helvetica bold italic
- 5 - courier
- 6 - courier bold
- 7 - courier italic
- 8 - courier bold italic
- 9 - times
- 10 - times bold
- 11 - times italic
- 12 - times bold italic
- 13 - symbol
- 14 - screen

- 15 - screen bold
- 16 - dingbats

isize -- size of the font.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of the upper left corner of the valuator, relative to the upper left corner of corresponding window. (Expressed in pixels.)

iy -- vertical position of the upper left corner of the valuator, relative to the upper left corner of corresponding window. (Expressed in pixels.)

image -- a handle referring to an eventual image opened with *bmopen* opcode. If it is set, it allows a skin for that widget.

Note about the bmopen opcode

Although the documentation mentions the *bmopen* opcode, it has not been implemented in Csound 4.22.

Performance

FLbox is useful to show some text in a window. The text is bounded by a box, whose aspect depends on *itype* argument.

Note that *FLbox* is not a valuator and its value is fixed. Its value cannot be modified.

Examples

Here is an example of the FLbox opcode. It uses the file *FLbox.csd* [examples/FLbox.csd].

Exemple 153. Example of the FLbox opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbox.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 700, 400, 50, 50
    ; Box border type (7=embossed box)
    itype = 7
```

```
    ; Font type (10='Times Bold')
    ifont = 10
    ; Font size
    isize = 20
    ; Width of the flbox
    iwidth = 400
    ; Height of the flbox
    iheight = 30
    ; Distance of the left edge of the flbox
    ; from the left edge of the panel
    ix = 150
    ; Distance of the upper edge of the flbox
    ; from the upper edge of the panel
    iy = 100

    ih3 FLbox "Use Text Boxes For Labelling", itype, ifont, isize, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin


</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLbutBank, *FLbutton*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLbutBank

FLbutBank — A FLTK widget opcode that creates a bank of buttons.

Description

A FLTK widget opcode that creates a bank of buttons.

Syntax

```
kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, \
      iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLbutBank* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

itype -- an integer number denoting the appearance of the widget. The valid numbers are:

- 1 - normal button
- 2 - light button
- 3 - check button
- 4 - round button

You can add 20 to the value to create a "plastic" type button. (Note that there is no Platic Round button. i.e. if you set type to 24 it will look exactly like type 23).

inumx -- number of buttons in each row of the bank.

inumy -- number of buttons in each column of the bank

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window, expressed in pixels

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window, expressed in pixels

iopcode -- score opcode type. You have to provide the ascii code of the letter corresponding to the score opcode. At present time only « i » (ascii code 105) score statements are supported. A zero value refers to a default value of « i ». So both 0 and 105 activates the *i* opcode. A value of -1 disables this opcode feature.

Performance

kout -- output value

kp1, *kp2*, ..., *kpN* -- arguments of the activated instruments.

The *FLbutBank* opcode creates a bank of buttons. For example, the following line:

```
gkButton,ihb1  FLbutBank  22, 8, 8, 380, 180, 50, 350, 0, 7, 0, 0, 5000, 6000
```

will create the this bank:

FLbutBank.

A click to a button checks that button. It may also uncheck a previous checked button belonging to the same bank. So the behaviour is always that of radio-buttons. Notice that each button is labeled with a progressive number. The *kout* argument is filled with that number when corresponding button is checked.

FLbutBank not only outputs a value but can also activate (or schedule) an instrument provided by the user each time a button is pressed. If the *iopcode* argument is set to a negative number, no instrument is activated so this feature is optional. In order to activate an instrument, *iopcode* must be set to 0 or to 105 (the ascii code of character « i », referring to the *i* score opcode). P-fields of the activated instrument are *kp1* (instrument number), *kp2* (action time), *kp3* (duration) and so on with user p-fields.

The *itype* argument sets the type of buttons identically to the *FLbutton* opcode. By adding 10 to the *itype* argument (i.e. by setting 11 for type 1, 12 for type 2, 13 for type 3 and 14 for type 4), it is possible to skip the current *FLbutBank* value when getting/setting snapshots (see *General FLTK Widget-related Opcodes*). You can also add 10 to "plastic" button types (31 for type 1, 32 for type 2, etc.)

FLbutBank is very useful to retrieve snapshots.

Examples

Here is an example of the FLbutBank opcode. It uses the file *FLbutBank.csd* [examples/FLbutBank.csd].

Exemple 154. Example of the FLbutBank opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr = 44100
nchnls = 1

FLpanel "Button Bank", 520, 140, 100, 100
    ;itype = 2    ;Light Buttons
    itype = 22    ;Plastic Light Buttons
    inumx = 10
    inumy = 4
    iwidth = 500
    iheight = 120
    ix = 10
    iy = 10
    iopcode = 0
    istarttim = 0
    idur = 1

    gkbutton, ihbb FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur

FLpanelEnd
FLrun

instr 1
  ibutton = i(gkbutton)
  prints "Button %i pushed!\\n", ibutton
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLbox, *FLbutton*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLbutton

FLbutton — A FLTK widget opcode that creates a button.

Description

A FLTK widget opcode that creates a button.

Syntax

```
kout, ihandle FLbutton "label", ion, ioff, itype, iwidth, iheight, ix, \
      iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

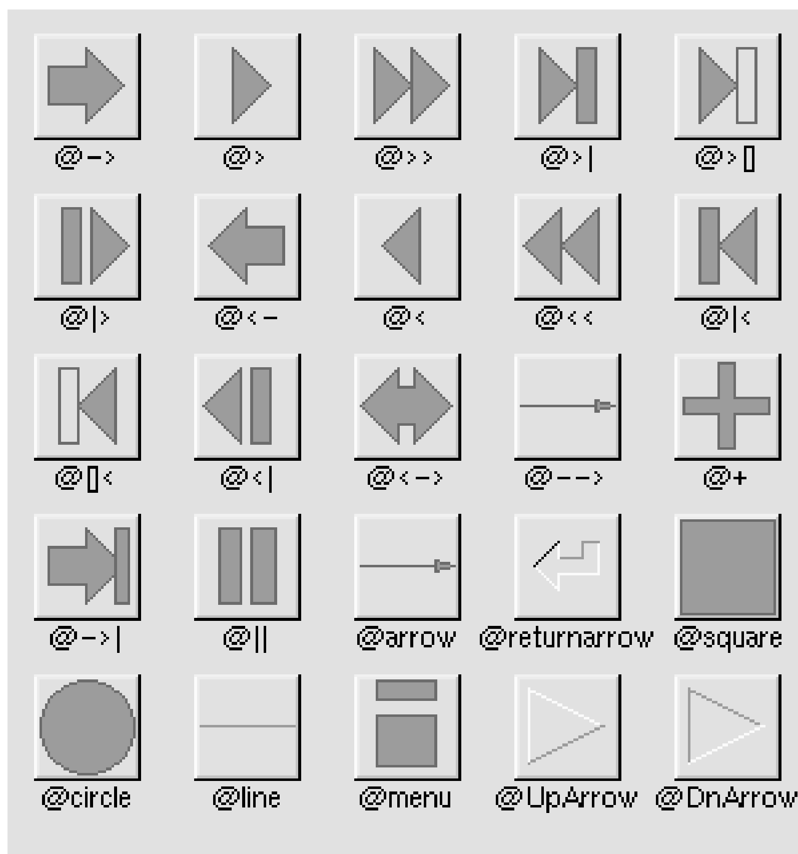
Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLbutton* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

Notice that with *FLbutton*, it is not necessary to call the *FLsetTextType* opcode at all in order to use a symbol. In this case, it is sufficient to set a label starting with « @ » followed by the proper formatting string.

The following symbols are supported:



FLTK label supported symbols.

The @ sign may be followed by the following optional « formatting » characters, in this order:

1. « # » forces square scaling rather than distortion to the widget's shape.
2. +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
3. [1-9] rotates by a multiple of 45 degrees. « 6 » does nothing, the others point in the direction of that key on a numeric keypad.

ion -- value output when the button is checked.

ioff -- value output when the button is unchecked.

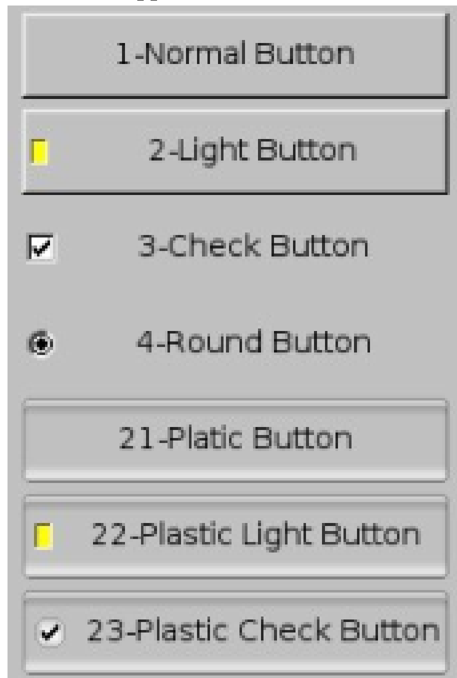
itype -- an integer number denoting the appearance of the widget.

Several kind of buttons are possible, according to the value of *itype* argument:

- 1 - normal button
- 2 - light button
- 3 - check button
- 4 - round button

You can add 20 to the value to create a "plastic" type button. (Note that there is no Platic Round button. i.e. if you set type to 24 it will look exactly like type 23).

This is the appearance of the buttons:



FLbutton.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iopcode -- score opcode type. You have to provide the ascii code of the letter corresponding to the score opcode. At present time only « i » (ascii code 105) score statements are supported. A zero value refers to a default value of « i ». So both 0 and 105 activates the *i* opcode. A value of -1 disables this opcode feature.

Performance

kout -- output value

kp1, *kp2*, ..., *kpN* -- arguments of the activated instruments.

Buttons of type 2, 3, and 4 also output (*kout* argument) the value contained in the *ion* argument when checked, and that contained in *ioff* argument when unchecked.

By adding 10 to *itype* argument (i.e. by setting 11 for type 1, 12 for type 2, 13 for type 3 and 14 for type 4) it is possible to skip the button value when getting/setting snapshots (see later section). *FLbutton* not only outputs a value, but can also activate (or schedule) an instrument provided by the user each time a button is pressed. You can also add 10 to "plastic" button types (31 for type 1, 32 for type 2, etc.)

If the *iopcode* argument is set to a negative number, no instrument is activated. So this feature is optional. In order to activate an instrument, *iopcode* must be set to 0 or to 105 (the ascii code of character « i », referring to the *i* score opcode).

P-fields of the activated instrument are *kp1* (instrument number), *kp2* (action time), *kp3* (duration) and so on with user p-fields. Notice that in dual state buttons (light button, check button and round button), the instrument is activated only when button state changes from unchecked to checked (not when passing from checked to unchecked).

Examples

Here is an example of the FLbutton opcode. It uses the file *FLbutton.csd* [examples/FLbutton.csd], and *beats.wav* [examples/beats.wav].

Exemple 155. Example of the FLbutton opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Using FLbuttons to create on screen controls for play,
; stop, fast forward and fast rewind of a sound file
; This example also makes use of a preset graphic for buttons.

sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

FLpanel "Buttons", 240, 400, 100, 100
    ion = 0
    ioff = 0
    itype = 1
    iwidth = 50
    iheight = 50
    ix = 10
    iy = 10
    iopcode = 0
    istarttim = 0
    idur = -1  ;Turn instruments on idefinitely

    ; Normal speed forwards
    gkplay, ihb1 FLbutton "@>", ion, ioff, itype, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur,
    ; Stationary
    gkstop, ihb2 FLbutton "@square", ion,ioff, itype, iwidth, iheight, ix+55, iy, iopcode, 2, istarttim
    ; Double speed backwards
    gkrew, ihb3 FLbutton "@<<", ion, ioff, itype, iwidth, iheight, ix + 110, iy, iopcode, 1, istarttim,
    ; Double speed forward
    gkff, ihb4 FLbutton "@>>", ion, ioff, itype, iwidth, iheight, ix+165, iy, iopcode, 1, istarttim, id
    ; Type 1
    gkt1, iht1 FLbutton "1-Normal Button", ion, ioff, 1, 200, 40, ix, iy + 65, -1
    ; Type 2
    gkt2, iht2 FLbutton "2-Light Button", ion, ioff, 2, 200, 40, ix, iy + 110, -1
    ; Type 3
    gkt3, iht3 FLbutton "3-Check Button", ion, ioff, 3, 200, 40, ix, iy + 155, -1
    ; Type 4
    gkt4, iht4 FLbutton "4-Round Button", ion, ioff, 4, 200, 40, ix, iy + 200, -1
    ; Type 21
    gkt5, iht5 FLbutton "21-Plastic Button", ion, ioff, 21, 200, 40, ix, iy + 245, -1
    ; Type 22
```

```
    gkt6, iht6 FLbutton "22-Plastic Light Button", ion, ioff, 22, 200, 40, ix, iy + 290, -1
    ; Type 23
    gkt7, iht7 FLbutton "23-Plastic Check Button", ion, ioff, 23, 200, 40, ix, iy + 335, -1
FLpanelEnd
FLrun

; Ensure that only 1 instance of instr 1
; plays even if the play button is clicked repeatedly
insnum = 1
icount = 1
maxalloc insnum, icount

instr 1
    asig diskin "beats.wav", p4, 0, 1
    outs asig, asig
endin

instr 2
    turnoff2 1, 0, 0   ;Turn off instr 1
    turnoff   ;Turn off this instrument
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLbox, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLcloseButton

FLcloseButton — A FLTK widget opcode that creates a button that will close the panel window it is a part of.

Description

A FLTK widget opcode that creates a button that will close the panel window it is a part of.

Syntax

```
ihandle FLcloseButton "label", iwidth, iheight, ix, iy
```

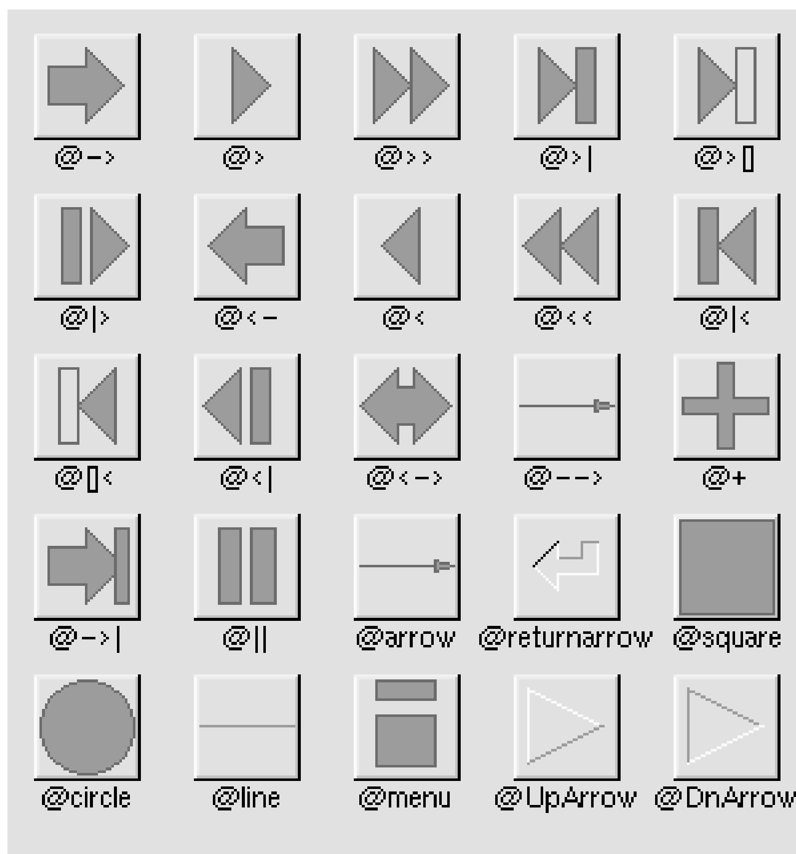
Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLcloseButton* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

Notice that with *FLcloseButton*, it is not necessary to call the *FLsetTextType* opcode at all in order to use a symbol. In this case, it is sufficient to set a label starting with « @ » followed by the proper formatting string.

The following symbols are supported:



FLTK label supported symbols.

The @ sign may be followed by the following optional « formatting » characters, in this order:

1. « # » forces square scaling rather than distortion to the widget's shape.
2. +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
3. [1-9] rotates by a multiple of 45 degrees. « 6 » does nothing, the others point in the direction of that key on a numeric keypad.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

See Also

FLbutton, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Steven Yi

New in version 5.05

FLcolor

FLcolor — A FLTK opcode that sets the primary colors.

Description

Sets the primary colors to RGB values given by the user.

Syntax

```
FLcolor ired, igreen, iblue [, ired2, igreen2, iblue2]
```

Initialization

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

ired2 -- The red component for the secondary color of the target widget. The range for each RGB component is 0-255

igreen2 -- The green component for the secondary color of the target widget. The range for each RGB component is 0-255

iblue2 -- The blue component for the secondary color of the target widget. The range for each RGB component is 0-255

Performance

These opcodes modify the appearance of other widgets. There are two types of such opcodes, those that don't contain the *ihandle* argument which affect all subsequently declared widgets, and those without *ihandle* which affect only a target widget previously defined.

FLcolor sets the primary colors to RGB values given by the user. This opcode affects the primary color of (almost) all widgets defined next its location. User can put several instances of *FLcolor* in front of each widget he intend to modify. However, to modify a single widget, it would be better to use the opcode belonging to the second type (i.e. those containing *ihandle* argument).

FLcolor is designed to modify the colors of a group of related widgets that assume the same color. The influence of *FLcolor* on subsequent widgets can be turned off by using -1 as the only argument of the opcode. Also, using -2 (or -3) as the only value of *FLcolor* makes all next widget colors randomly selected. The difference is that -2 selects a light random color, while -3 selects a dark random color.

Using *ired2*, *igreen2*, *iblue2* is equivalent to using a separate *FLcolor2*.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLcolor2

FLcolor2 — A FLTK opcode that sets the secondary (selection) color.

Description

FLcolor2 is the same of *FLcolor* except it affects the secondary (selection) color.

Syntax

```
FLcolor2 ired, igreen, iblue
```

Initialization

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

Performance

These opcodes modify the appearance of other widgets. There are two types of such opcodes: those that don't contain the *ihandle* argument which affect all subsequently declared widgets, and those without *ihandle* which affect only a target widget previously defined.

FLcolor2 is the same of *FLcolor* except it affects the secondary (selection) color. Setting it to -1 turns off the influence of *FLcolor2* on subsequent widgets. A value of -2 (or -3) makes all next widget secondary colors randomly selected. The difference is that -2 selects a light random color, while -3 selects a dark random color.

See Also

FLcolor, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLcount

FLcount — A FLTK widget opcode that creates a counter.

Description

Allows the user to increase/decrease a value with mouse clicks on a corresponding arrow button.

Syntax

```
kout, ihandle FLcount "label", imin, imax, istep1, istep2, itype, \
      iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. Used by further opcodes that changes some valuator's properties. It is automatically set by the corresponding valuator.

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

imin -- minimum value of output range

imax -- maximum value of output range

istep1 -- a floating-point number indicating the increment of valuator value corresponding to of each mouse click. *istep1* is for fine adjustments.

istep2 -- a floating-point number indicating the increment of valuator value corresponding to of each mouse click. *istep2* is for coarse adjustments.

itype -- an integer number denoting the appearance of the valuator.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iopcode -- score opcode type. You have to provide the ascii code of the letter corresponding to the score opcode. At present time only « i » (ascii code 105) score statements are supported. A zero value refers to a default value of « i ». So both 0 and 105 activates the *i* opcode. A value of -1 disables this opcode feature.

Performance

kout -- output value

kp1, *kp2*, ..., *kpN* -- arguments of the activated instruments.

FLcount allows the user to increase/decrease a value with mouse clicks on corresponding arrow buttons:

FLcount.

There are two kind of arrow buttons, for larger and smaller steps. Notice that *FLcount* not only outputs a value and a handle, but can also activate (schedule) an instrument provided by the user each time a button is pressed. P-fields of the activated instrument are *kp1* (instrument number), *kp2* (action time), *kp3* (duration) and so on with user p-fields. If the *iopcode* argument is set to a negative number, no instrument is activated. So this feature is optional.

Examples

Here is an example of the FLcount opcode. It uses the file *FLcount.csd* [examples/FLcount.csd].

Exemple 156. Example of the FLcount opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLcount.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flcount opcode
; clicking on the single arrow buttons
; increments the oscillator in semitone steps
; clicking on the double arrow buttons
; increments the oscillator in octave steps
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Counter", 900, 400, 50, 50
    ; Minimum value output by counter
    imin = 6
    ; Maximum value output by counter
    imax = 12
    ; Single arrow step size (semitones)
    istep1 = 1/12
    ; Double arrow step size (octave)
    istep2 = 1
    ; Counter type (1=double arrow counter)
    itype = 1
    ; Width of the counter in pixels
    iwidth = 200
    ; Height of the counter in pixels
    iheight = 30
    ; Distance of the left edge of the counter
    ; from the left edge of the panel
    ix = 50
    ; Distance of the top edge of the counter
    ; from the top edge of the panel
    iy = 50
    ; Score event type (-1=ignored)
    iopcode = -1
```

```
    gkoct, ihandle FLcount "pitch in oct format", imin, imax, istep1, istep2, itype, iwidth, iheight, i
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, cpsoct(gkoct), ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLjoy, *FLknob*, *FLroller*, *FLslider*, *FLtext*

Credits

Author: Gabriel Maldonado

New in version 4.22

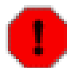
Example written by Iain McCurdy, edited by Kevin Conder.

FLexecButton

FLexecButton — A FLTK widget opcode that creates a button that executes a command.

Description

A FLTK widget opcode that creates a button that executes a command. Useful for opening up HTML documentation as About text or to start a separate program from an FLTK widget interface.



Warning

Because any command can be executed, the user is advised to be very careful when using this opcode and when running orchestras by others using this opcode.

Syntax

```
ihandle FLexecButton "command", iwidth, iheight, ix, iy
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLexecButton*.

« command » -- a double-quoted string containing a command to execute.

Notice that with *FLexecButton*, the default text for the button is "About" and it is necessary to call the *FLsetText* opcode to change the text of the button.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Examples

Here is an example of the FLexecButton opcode. It uses the file *FLexecButton.csd* [examples/FLexecButton.csd].

Exemple 157. Example of the FLexecButton opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
```

```
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No display
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

  sr      =  44100
  ksmps     =  10
  nchnls    =  1

; Example by Jonathan Murphy 2007

;;; reset amplitude range
  0dbfs     =  1

;;; set the base colour for the panel
            FLcolor   100, 0, 200
;;; define the panel
            FLpanel   "FLexecButton", 250, 100, 0, 0
;;; sliders to control time stretch and pitch
  gkstr, gistretch    FLslider                   "Time", 0.5, 1.5, 0, 6, -1, 10, 60, 150, 20
  gkpch, gipitch      FLslider                   "Pitch", 0.5, 1.5, 0, 6, -1, 10, 60, 200, 20
;;; set FLexecButton colour
            FLcolor   255, 255, 0
;;; when this button is pressed, fourier analysis is performed on the file
;;; "beats.wav", producing the analysis file "beats.pvx"
  gipvoc    FLexecButton  "csound -U pvanal beats.wav beats.pvx", 60, 20, 20, 20
;;; set FLexecButton text
            FLsetText "PVOC", gipvoc
;;; when this button is pressed, instr 10000 is called, exiting
;;; Csound immediately

;;; cancel previous  colour
            FLcolor   -1
;;; set colour for kill button
            FLcolor   255, 0, 0
  gkkill, gikill      FLbutton                   "X", 1, 1, 1, 20, 20, 100, 20, 0, 10000, 0, 0.1
;;; cancel previous colour
            FLcolor   -1
;;; set colour for play/stop and pause buttons
            FLcolor   0, 200, 0
;;; pause and play/stop buttons
  gkpause, gipause    FLbutton                   "@||", 1, 0, 2, 40, 20, 20, 60, -1
  gkplay, giplay      FLbutton                   "@|>", 1, 0, 2, 40, 20, 80, 60, -1
;;; end the panel
            FLpanelEnd
;;; set initial values for time stretch and pitch
            FLsetVal_i 1, gistretch
            FLsetVal_i 1, gipitch
;;; run the panel
            FLrun


    instr 1                                ; trigger play/stop
;;; is the play/stop button on or off?
;;; either way we need to trigger something,
;;; so we can't just use the value of gkplay
  kon      trigger   gkplay, 0, 0
  koff      trigger   gkplay, 1, 1
;;; if on, start instr 2
            schedkwhen kon, -1, -1, 2, 0, -1
;;; if off, stop instr 2
            schedkwhen koff, -1, -1, -2, 0, -1

    endin

    instr 2

;;; paused or playing?
if (gkpause == 1) kgoto pause
            kgoto     start
pause:
;;; if the pause button is on, skip sound production
            kgoto     end
start:
;;; get the length of the analysis file in seconds
  ilen     filelen   "beats.pvx"
;;; determine base frequency of playback
  icps      =  1/ilen
;;; create a table over the length of the file
```

```
  itpt      ftgen     0, 0, 513, -7, 0, 512, ilen
;;; phasor for time control
  kphs      phasor    icps * gkstr
;;; use phasor as index into table
  kndx      =  kphs * 512
;;; read table
  ktpt      tablei    kndx, itpt
;;; use value from table as time pointer into file
  fsig1      pvsfread  ktpt, "beats.pvx"
;;; change playback pitch
  fsig2      pvscale   fsig1, gkpch
;;; resynthesize
  aout      pvsynth   fsig2
;;; envelope to avoid clicks and clipping
  aenv      linsegr   0, 0.3, 0.75, 0.1, 0
  aout      =  aout * aenv
             out       aout
end:

    endin


    instr 10000                                  ; kill

            exitnow

    endin

</CsInstruments>
<CsScore>
i1 0 10000
e
</CsScore>
</CsoundSynthesizer>
```

See Also

FLbutton, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Steven Yi

Example by: Jonathan Murphy

New in version 5.05

FLgetsnap

FLgetsnap — Retrieves a previously stored FLTK snapshot.

Description

Retrieves a previously stored snapshot (in memory), i.e. sets all valuator to the corresponding values stored in that snaphot.

Syntax

```
inumsnap FLgetsnap index [, igroup]
```

Initialization

inumsnap -- current number of snapshots.

index -- a number referring unequivocally to a snapshot. Several snapshots can be stored in the same bank.

igroup -- (optional) an integer number referring to a snapshot-related group of widget. It allows to get/set, or to load/save the state of a subset of valuators. Default value is zero that refers to the first group. The group number is determined by the opcode *FLsetSnapGroup*.

Note

The *igroup* parameter has not been yet fully implemented in the current version of csound. Please do not rely on it yet.

Performance

FLgetsnap retrieves a previously stored snapshot (in memory), i.e. sets all valuator to the corresponding values stored in that snapshot. The *index* argument unequivocally must refer to an already existing snapshot. If the *index* argument refers to an empty snapshot or to a snapshot that doesn't exist, no action is done. *FLsetsnap* outputs the current number of snapshots (*inumsnap* argument).

For purposes of snapshot saving, widgets can be grouped, so that snapshots affect only a defined group of widgets. The opcode *FLsetSnapGroup* is used to specify the group for all widgets declared after it, until the next *FLsetSnapGroup* statement.

See Also

FLloadsnap, *FLrun*, *FLsavesnap*, *FLsetsnap*, *FLsetSnapGroup*, *FLupdate*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLgroup

FLgroup — A FLTK container opcode that groups child widgets.

Description

A FLTK container opcode that groups child widgets.

Syntax

```
FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]
```

Initialization

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iborder (optional, default=0) -- border type of the container. It is expressed by means of an integer number chosen from the following:

- 0 - no border
- 1 - down box border
- 2 - up box border
- 3 - engraved border
- 4 - embossed border
- 5 - black line border
- 6 - thin down border
- 7 - thin up border

If the integer number doesn't match any of the previous values, no border is provided as the default.

image (optional) -- a handle referring to an eventual image opened with the *bmopen* opcode. If it is set, it allows a skin for that widget.

Note about the bmopen opcode

Although the documentation mentions the *bmopen* opcode, it has not been implemented in Csound 4.22.

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroupEnd, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLgroupEnd

FLgroupEnd — Marks the end of a group of FLTK child widgets.

Description

Marks the end of a group of FLTK child widgets.

Syntax

```
FLgroupEnd
```

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroup, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLgroupEnd

FLgroup_end — Marks the end of a group of FLTK child widgets.

Description

Marks the end of a group of FLTK child widgets. This is another name for **FLgroupEnd** provides for compatibility. See *FLgroupEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLhide

FLhide — Hides the target FLTK widget.

Description

Hides the target FLTK widget, making it invisible.

Syntax

```
FLhide ihandle
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLbutBank* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

Performance

FLhide hides target widget, making it invisible.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLhvsBox

FLhvsBox — Displays a box with a grid useful for visualizing two-dimensional Hyper Vectorial Synthesis.

Description

FLhvsBox displays a box with a grid useful for visualizing two-dimensional Hyper Vectorial Synthesis.

Syntax

```
ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]
```

Initialization

ihandle – an integer number used a univocally-defined handle for identifying a specific HVS box (see below).

inumlinesX, inumlinesY - number of vertical and horizontal lines delimiting the HVS squared areas

iwidth, iheight - width and height of the HVS box

ix, iy - the position of the HVS box

image – (optional, default 0) an integer number denoting an RGB image opened with the *bmopen* opcode. A zero indicates no image.

Performance

FLhvsBox is a widget able to visualize current position of the HVS cursor in an HVS box (i.e. a squared area containing a grid). The number of horizontal and vertical lines of the grid can be defined with the *inumlinesX, inumlinesY* arguments. This opcode has to be declared inside an *FLpanel* - *FLpanelEnd* block. See the entry for *hvs2* for an example of usage of *FLhvsBox*.

FLhvsBoxSetValue is used to set the cursor position of an *FLhvsBox* widget.

Note

The opcode *bmscan* has not been implemented, so currently the parameter *image* has no effect.

See Also

hvs2, *FLhvsBoxSetValue*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLhvsBoxSetValue

FLhvsBoxSetValue — Sets the cursor position of a previously-declared FLhvsBox widget.

Description

FLhvsBoxSetValue sets the cursor position of a previously-declared *FLhvsBox* widget.

Syntax

```
FLhvsBox kx, ky, ihandle
```

Initialization

ihandle – an integer number used a univocally-defined handle for identifying a specific HVS box (see below).

Performance

kx, ky– the coordinates of the HVS cursor position to be set.

FLhvsBoxSetValue sets the cursor position of a previously-declared *FLhvsBox* widget. The *kx* and *ky* arguments, denoting the cursor position, have to be expressed in normalized values (0 to 1 range).

See the entry for *hvs2* for an example of usage of *FLhvsBoxSetValue*.

See Also

hvs2, *FLhvsBox*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLjoy

FLjoy — A FLTK opcode that acts like a joystick.

Description

FLjoy is a squared area that allows the user to modify two output values at the same time. It acts like a joystick.

Syntax

```
koutx, kouty, ihandlex, ihandley FLjoy "label", iminx, imaxx, iminy, \
      imaxy, iexpx, iexpy, idispx, idispy, iwidth, iheight, ix, iy
```

Initialization

ihandlex -- a handle value (an integer number) that unequivocally references a corresponding widget. Used by further opcodes that changes some valuator's properties. It is automatically set by the corresponding valuator.

ihandley -- a handle value (an integer number) that unequivocally references a corresponding widget. Used by further opcodes that changes some valuator's properties. It is automatically set by the corresponding valuator.

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

iminx -- minimum x value of output range

imaxx -- maximum x value of output range

iminy -- minimum y value of output range

imaxy -- maximum y value of output range

iwidth -- width of widget.

idispx -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

idispy -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

iexpx -- an integer number denoting the behaviour of valuator:

- 0 = valuator output is linear
- -1 = valuator output is exponential

All other positive numbers for *iexpx* indicate the number of an existing table that is used for indexing.

Linear interpolation is provided in table indexing. A negative table number suppresses interpolation.

iexpy -- an integer number denoting the behaviour of valuator:

- 0 = valuator output is linear
- -1 = valuator output is exponential

All other positive numbers for *iexpy* indicate the number of an existing table that is used for indexing. Linear interpolation is provided in table indexing. A negative table number suppresses interpolation.

IMPORTANT!

Notice that the tables used by valuators must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. In fact, tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

koutx -- x output value

kouty -- y output value

Examples

Here is an example of the FLjoy opcode. It uses the file *FLjoy.csd* [examples/FLjoy.csd].

Exemple 158. Example of the FLjoy opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLjoy.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flpanel opcode
; Horizontal click-dragging controls the frequency of the oscillator
; Vertical click-dragging controls the amplitude of the oscillator
sr = 44100
kr = 441
ksmps = 100
nchnls = 1
```

```
FLpanel "X Y Panel", 900, 400, 50, 50
    ; Minimum value output by x movement (frequency)
    iminx = 200
    ; Maximum value output by x movement (frequency)
    imaxx = 5000
    ; Minimum value output by y movement (amplitude)
    iminy = 0
    ; Maximum value output by y movement (amplitude)
    imaxy = 15000
    ; Logarithmic change in x direction
    iexpx = -1
    ; Linear change in y direction
    iexpy = 0
    ; Display handle x direction (-1=not used)
    idispx = -1
    ; Display handle y direction (-1=not used)
    idispy = -1
    ; Width of the x y panel in pixels
    iwidth = 800
    ; Height of the x y panel in pixels
    iheight = 300
    ; Distance of the left edge of the x y panel from
    ; the left edge of the panel
    ix = 50
    ; Distance of the top edge of the x y
    ; panel from the top edge of the panel
    iy = 50

    gkfreqx, gkampy, ihandlex, ihandley FLjoy "X - Frequency Y - Amplitude", iminx, imaxx, iminy, imaxy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
    asig oscili gkampy, gkfreqx, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLcount, *FLknob*, *FLroller*, *FLslider*, *FLtext*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLkeyIn

FLkeyIn — Reports keys pressed (on alphanumeric keyboard) when an FLTK panel has focus.

Description

FLkeyIn informs about the status of a key pressed by the user on the alphanumeric keyboard when an FLTK panel has got the focus.

Syntax

```
kascii FLkeyIn [ifn]
```

Initialization

ifn – (optional, default value is zero) set the behavior of FLkeyIn (see below).

Performance

kascii - the ascii value of last pressed key. If the key is pressed, the value is positive, when the key is released the value is negative.

FLkeyIn is useful to know whether a key has been pressed on the computer keyboard. The behavior of this opcode depends on the optional *ifn* argument.

If *ifn* = 0 (default), *FLkeyIn* outputs the ascii code of the last pressed key. If it is a special key (ctrl, shift, alt, f1-f12 etc.), a value of 256 is added to the output value in order to distinguish it from normal keys. The output will continue to output the last key value, until a new key is pressed or released. Notice that the output will be negative when a key is depressed.

If *ifn* is set to the number of an already-allocated table having at least 512 elements, then the table element having index equal to the ascii code of the key pressed is set to 1, all other table elements are set to 0. This allows to check the state of a certain key or set of keys.

Be aware that you must set the *ikbdcapture* parameter to something other than 0 on a designated *FLpanel* for *FLkeyIn* to capture keyboard events from that panel.

Note

FLkeyIn works internally at k-rate, so it can't be used in the header as other FLTK opcodes. It must be used inside an instrument.

Examples

Here is an example of the FLkeyIn opcode. It uses the file *FLkeyIn.csd* [examples/FLkeyIn.csd].

Exemple 159. Example of the FLkeyIn opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command

line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007

FLpanel "FLkeyIn", 400, 300, -1, -1, 5, 1, 1
FLpanelEnd

FLrun

0dbfs = 1

instr 1
kascii   FLkeyIn
ktrig changed kascii
if (kascii > 0) then
  printf "Key Down: %i\n", ktrig, kascii
else
  printf "Key Up: %i\n", ktrig, -kascii
endif
endin

</CsInstruments>
<CsScore>
i 1 0 120
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado

New in version 5.06

FLknob

FLknob — A FLTK widget opcode that creates a knob.

Description

A FLTK widget opcode that creates a knob.

Syntax

```
kout, ihandle FLknob "label", imin, imax, iexp, itype, idisp, iwidth, \
      ix, iy [, icursorsize]
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically utput by *FLknob* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

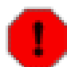
imin -- minimum value of output range.

imax -- maximum value of output range.

iexp -- an integer number denoting the behaviour of valuator:

- 0 = valuator output is linear
- -1 = valuator output is exponential

All other positive numbers for *iexp* indicate the number of an existing table that is used for indexing. Linear interpolation is provided in table indexing. A negative table number suppresses interpolation.



IMPORTANT!

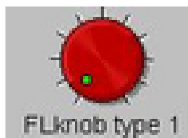
Notice that the tables used by valuators must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. In fact, tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

itype -- an integer number denoting the appearance of the valuator.

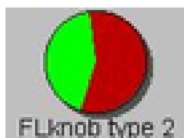
The *itype* argument can be set to the following values:

- 1 - a 3-D knob
- 2 - a pie-like knob

- 3 - a clock-like knob
- 4 - a flat knob



A 3-D knob.



A pie knob.

A clock knob.

A flat knob.

idisp -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

icursorsize (optional) -- If *FLknob*'s *itype* is set to 1 (3D knob), this parameter controls the size of knob cursor.

Performance

kout -- output value

FLknob puts a knob in the corresponding container.

Examples

Here is an example of the FLknob opcode. It uses the file *FLknob.csd* [examples/FLknob.csd].

Exemple 160. Example of the FLknob opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLknob.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flknob controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Knob", 900, 400, 50, 50
    ; Minimum value output by the knob
    imin = 200
    ; Maximum value output by the knob
    imax = 5000
    ; Logarithmic type knob selected
    iexp = -1
    ; Knob graphic type (1=3D knob)
    itype = 1
    ; Display handle (-1=not used)
    idisp = -1
    ; Width of the knob in pixels
    iwidth = 70
    ; Distance of the left edge of the knob
    ; from the left edge of the panel
    ix = 70
    ; Distance of the top edge of the knob
    ; from the top of the panel
    iy = 125

    gkfreq, ihandle FLknob "Frequency", imin, imax, iexp, itype, idisp, iwidth, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

Here is another example of the FLknob opcode, showing the different styles of knobs and the usage of FLvalue to display a knob's value. It uses the file *FLknob-2.csd* [examples/FLknob-2.csd].

Exemple 161. More complex example of the FLknob opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac          -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLknob.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007
FLpanel "Knob Types", 330, 230, 50, 50
    ; Distance of the left edge of the knob
    ; from the left edge of the panel
    ix = 20
    ; Distance of the top edge of the knob
    ; from the top of the panel
    iy = 20

    ;Create boxes that display a widget's value
    ihandleA FLvalue "A", 60, 20, ix + 130, iy + 110
    ihandleB FLvalue "B", 60, 20, ix + 220, iy + 110
    ihandleC FLvalue "C", 60, 20, ix + 130, iy + 160
    ihandleD FLvalue "D", 60, 20, ix + 220, iy + 160

    ; The foru types of FLknobs
    gkdummy1, ihandle1 FLknob "Type 1", 200, 5000, -1, 1, ihandleA, 70, ix, iy, 90
    gkdummy2, ihandle2 FLknob "Type 2", 200, 5000, -1, 2, ihandleB, 70, ix + 100, iy
    gkdummy3, ihandle3 FLknob "Type 3", 200, 5000, -1, 3, ihandleC, 70, ix + 200, iy
    gkdummy4, ihandle4 FLknob "Type 4", 200, 5000, -1, 4, ihandleD, 70, ix , iy + 100
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the color of widgets
FLsetColor 20, 23, 100, ihandle1
FLsetColor 0, 123, 100, ihandle2
FLsetColor 180, 23, 12, ihandle3
FLsetColor 10, 230, 0, ihandle4

FLsetColor2 200, 230, 0, ihandle1
FLsetColor2 200,0 ,123 , ihandle2
FLsetColor2 180, 180, 100, ihandle3
FLsetColor2 180, 23, 12, ihandle4


; Set the initial value of the widget
FLsetVal_i 300, ihandle1
FLsetVal_i 1000, ihandle2


instr 1
; Nothing here for now
endin


</CsInstruments>
<CsScore>

f 0 3600   ;Dumy table to make csound wait for realtime events

e
```

```
</CsScore>
</CsoundSynthesizer>
```

See Also

FLcount, *FLjoy*, *FLroller*, *FLslider*, *FLtext*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLlabel

FLlabel — A FLTK opcode that modifies the appearance of a text label.

Description

Modifies a set of parameters related to the text label appearence of a widget (i.e. size, font, alignment and color of corresponding text).

Syntax

```
FLlabel isize, ifont, ialign, ired, igreen, iblue
```

Initialization

isize -- size of the font of the target widget. Normal values are in the order of 15. Greater numbers enlarge font size, while smaller numbers reduce it.

ifont -- sets the the font type of the label of a widget.

Legal values for ifont argument are:

- 1 - Helvetica (same as Arial under Windows)
- 2 - Helvetica Bold
- 3 - Helvetica Italic
- 4 - Helvetica Bold Italic
- 5 - Courier
- 6 - Courier Bold
- 7 - Courier Italic
- 8 - Courier Bold Italic
- 9 - Times
- 10 - Times Bold
- 11 - Times Italic
- 12 - Times Bold Italic
- 13 - Symbol
- 14 - Screen
- 15 - Screen Bold
- 16 - Dingbats

ialign -- sets the alignment of the label text of the widget.

Legal values for ialign argument are:

- 1 - align center
- 2 - align top
- 3 - align bottom
- 4 - align left
- 5 - align right
- 6 - align top-left
- 7 - align top-right
- 8 - align bottom-left
- 9 - align bottom-right

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

Performance

FLlabel modifies a set of parameters related to the text label appearance of a widget, i.e. size, font, alignment and color of corresponding text. This opcode affects (almost) all widgets defined next its location. A user can put several instances of *FLlabel* in front of each widget he intends to modify. However, to modify a particular widget, it is better to use the opcode belonging to the second type (i.e. those containing the *ihandle* argument).

The influence of *FLlabel* on the next widget can be turned off by using -1 as its only argument. *FLlabel* is designed to modify text attributes of a group of related widgets.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLloadsnap

FLloadsnap — Loads all snapshots into the memory bank of the current orchestra.

Description

FLloadsnap loads all the snapshots contained in a file into the memory bank of the current orchestra.

Syntax

```
FLloadsnap "filename" [, igroup]
```

Initialization

"filename" -- a double-quoted string corresponding to a file to load a bank of snapshots.

igroup -- (optional) an integer number referring to a snapshot-related group of widget. It allows to get/set, or to load/save the state of a subset of valuators. Default value is zero that refers to the first group. The group number is determined by the opcode *FLsetSnapGroup*.

Note

The *igroup* parameter has not been yet fully implemented in the current version of csound. Please do not rely on it yet.

Performance

FLloadsnap loads all snapshots contained in filename into the memory bank of current orchestra.

For purposes of snapshot saving, widgets can be grouped, so that snapshots affect only a defined group of widgets. The opcode *FLsetSnapGroup* is used to specify the group for all widgets declared after it, until the next *FLsetSnapGroup* statement.

See Also

FLgetsnap, *FLrun*, *FLsetSnapGroup*, *FLsavesnap*, *FLsetsnap*, *FLupdate*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLmouse

FLmouse — Returns the mouse position and the state of the three mouse buttons.

Description

FLmouse returns the coordinates of the mouse position within an FLTK panel and the state of the three mouse buttons.

Syntax

```
kx, ky, kb1, kb2, kb3 FLmouse [, imode]
```

Initialization

imode – (optional, default = 0) Determines the mode for mouse location reporting.

- 0 - Absolute position normalized to range 0-1
- 1 - Absolute raw pixel position
- 2 - Raw pixel position, relative to FLTK panel

Performance

kx, ky – the mouse coordinates, whose range depends on the iflag argument (see above).

kb1, kb2, kb3 – the states of the mouse buttons, 1 when corresponding button is pressed, 0 when the button is not pressed.

FLmouse returns the coordinates of the mouse position and the state of the three mouse buttons. The coordinates can be retrieved in three modes modes depending on the *imode* argument value (see above). Modes 0 and 1 report mouse position in realtion to the complete screen (Absolute mode), while mode 2, reports the pixel position within an FLTK panel. Notice that FLmouse is only active when the mouse cursor passes on an *FLpanel* area.

Examples

Here is an example of the FLmouse opcode. It uses the file *FLmouse.csd* [examples/FLmouse.csd].

Exemple 162. Example of the FLmouse opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
```

```
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007
giwidth = 400
giheight = 300
FLpanel "FLmouse", giwidth, giheight, 10, 10
FLpanelEnd

FLrun

0dbfs = 1

instr 1
  kx, ky, kb1, kb2, kb3    FLmouse 2
  ktrig changed kx, ky  ;Print only if coordinates have changed
  printf "kx = %f   ky = %f \n", ktrig, kx, ky
kfreq = ((giwidth - ky)*1000/giwidth) + 300

; y coordinate determines frequency, x coordinate determines amplitude
; Left mouse button (kb1) doubles the frequency
; Right mouse button (kb3) activates sound on channel 2
  aout oscil kx /giwidth , kfreq * (kb1 + 1), 1
  outs aout, aout * kb3
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1

i 1 0 120
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado

New in version 5.06

flooper

flooper — Function-table-based crossfading looper.

Description

This opcode reads audio from a function table and plays it back in a loop with user-defined start time, duration and crossfade time. It also allows the pitch of the loop to be controlled, including reversed playback. It accepts non-power-of-two tables, such as deferred-allocation GEN01 tables.

Syntax

```
asig flooper kamp, kpitch, istart, idur, ifad, ifn
```

Initialisation

istart -- loop start pos in seconds

idur -- loop duration in seconds

ifad -- crossfade duration in seconds

ifn -- function table number, generally created using GEN01

Performance

asig -- output sig

kon -- amplitude control

kpitch -- pitch control (transposition ratio); negative values play the loop back in reverse

Examples

Exemple 163. Example

```
aout flooper 16000, 1, 1, 4, 0.05, 1  ;  loop starts at 1 sec, for 4 secs 0.05 crossfade
       out aout
```

The example above shows the basic operation of flooper. Pitch can be controlled at the k-rate, as well as amplitude. The example assumes table 1 to contain at least 5.05 seconds of audio (4 secs loop duration, starting 1 sec into the table, using 0.05 secs after the loop end for the crossfade).

Credits

Author: Victor Lazzarini;
April 2005

New plugin in version 5

April 2005.

flooper2

flooper2 — Function-table-based crossfading looper.

Description

This opcode implements a crossfading looper with variable loop parameters and three looping modes, optionally using a table for its crossfade shape. It accepts non-power-of-two tables for its source sounds, such as deferred-allocation GEN01 tables.

Syntax

```
asig flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \
      [, istart, imode, ifenv, iskip]
```

Initialisation

ifn -- sound source function table number, generally created using GEN01

istart -- playback start pos in seconds

imode -- loop modes: 0 forward, 1 backward, 2 back-and-forth [def: 0]

ifenv -- if non-zero, crossfade envelope shape table number. The default, 0, sets the crossfade to linear.

iskip -- if 1, the opcode initialisation is skipped, for tied notes, performance continues from the position in the loop where the previous note stopped. The default, 0, does not skip initialisation

Performance

asig -- output sig

kamp -- amplitude control

kpitch -- pitch control (transposition ratio); negative values are not allowed.

kloopstart -- loop start point (secs). Note that although k-rate, loop parameters such as this are only updated once per loop cycle.

kloopend -- loop end point (secs), updated once per loop cycle.

kcrossfade -- crossfade length (secs), updated once per loop cycle and limited to loop length.

Examples

Exemple 164. Example

```
aout flooper2 16000, 1, 1, 5, 0.05, 1   ;  loop starts at 1 sec, for 4 secs 0.05 crossfade
       out aout
```

The example above shows the basic operation of flooper. Pitch can be controlled at the k-rate, as well as amplitude and loop parameters. The example assumes table 1 to contain at least 5.05 seconds of audio (4 secs loop duration, starting 1 sec into the table, using 0.05 secs after the loop end for the crossfade). Looping is in mode 0 (normal forward loop).

Credits

Author: Victor Lazzarini;
July 2006

New plugin in version 5

July 2006.

floor

floor — Retourne le plus grand entier inférieur ou égal à *x*.

Description

Retourne le plus grand entier inférieur ou égal à *x*.

Syntaxe

```
floor(x) (argument au taux d'initialisation, de contrôle ou audio)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Voir Aussi

abs, *exp*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Auteur : Istvan Varga
Nouveau dans Csound 5
2005

FLpack

FLpack — Provides the functionality of compressing and aligning FLTK widgets.

Description

FLpack provides the functionality of compressing and aligning widgets.

Syntax

```
FLpack iwidth, iheight, ix, iy, itype, ispace, iborder
```

Initialization

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

itype -- an integer number that modifies the appearance of the target widget.

The *itype* argument expresses the type of packing:

- 0 - vertical
- 1 - horizontal

ispace -- sets the space between the widgets.

iborder -- border type of the container. It is expressed by means of an integer number chosen from the following:

- 0 - no border
- 1 - down box border
- 2 - up box border
- 3 - engraved border
- 4 - embossed border
- 5 - black line border
- 6 - thin down border
- 7 - thin up border

Performance

FLpack provides the functionality of compressing and aligning widgets.

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

Examples

The following example:

```
        FLpanel "Panel1",450,300,100,100
        FLpack  400,300, 10,40,0,15,3
gk1,ihs1        FLslider        "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2,ihs2        FLslider        "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3,ihs3        FLslider        "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4,ihs4        FLslider        "FLslider 4", 250, 5000, 1 ,11, -1, 300,30, 20,200
gk5,ihs5        FLslider        "FLslider 5", 220, 8000, 2 ,1, -1, 300,15, 20,250
gk6,ihs6        FLslider        "FLslider 6", 1, 5000, 1 ,13, -1, 300,15, 20,300
gk7,ihs7        FLslider        "FLslider 7", 870, 5000, 1 ,15, -1, 300,30, 20,350
        FLpackEnd
        FLpanelEnd
```

...will produce this result, when resizing the window:

FLpack.

See Also

FLgroup, *FLgroupEnd*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLpackEnd

FLpackEnd — Marks the end of a group of compressed or aligned FLTK widgets.

Description

Marks the end of a group of compressed or aligned FLTK widgets.

Syntax

FLpackEnd

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLpack_end

FLpack_End — Marks the end of a group of compressed or aligned FLTK widgets.

Description

Marks the end of a group of compressed or aligned FLTK widgets. This is another name for **FLpanelEnd** provided for compatibility. See *FLpanel_end*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLpanel

FLpanel — Creates a window that contains FLTK widgets.

Description

Creates a window that contains FLTK widgets.

Syntax

```
FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]
```

Initialization

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

iwidth -- width of widget.

iheight -- height of widget.

ix (optional) -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy (optional) -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iborder (optional) -- border type of the container. It is expressed by means of an integer number chosen from the following:

- 0 - no border
- 1 - down box border
- 2 - up box border
- 3 - engraved border
- 4 - embossed border
- 5 - black line border
- 6 - thin down border
- 7 - thin up border

ikbdcapture (default = 0) -- If this flag is set to 1, keyboard events are captured by the window (for use with *sensekey* and *FLkeyIn*)

iclose (default = 0) -- If this flag is set to anything other than 0, the close button of the window is disabled, and the window cannot be closed by the user directly. It will close when csound exits.

Performance

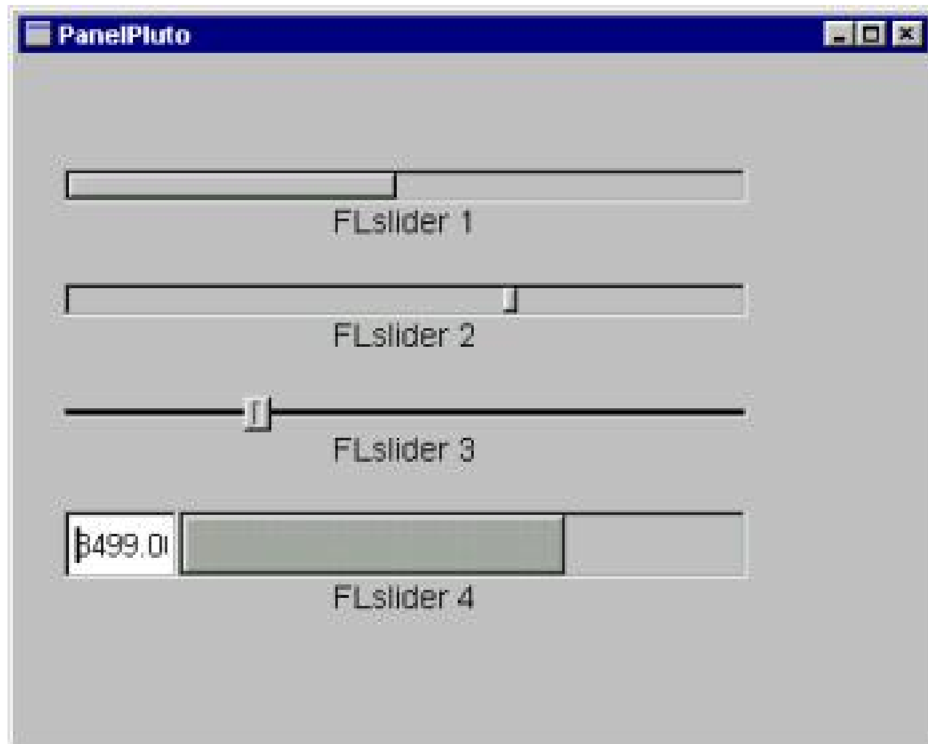
Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

FLpanel creates a window. It must be followed by the opcode *FLpanelEnd* when all widgets internal to it are declared. For example:

```
        FLpanel   "PanelPluto",450,550,100,100 ;***** start of container
gk1,ih1 FLslider  "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2,ih2 FLslider  "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3,ih3 FLslider  "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4,ih4 FLslider  "FLslider 4", 250, 5000, 1 ,11,-1, 300,30, 20,200
        FLpanelEnd ;***** end of container
```

will output the following result:



FLpanel.

If the *ikbdcapture* flag is set, the window captures keyboard events, and sends them to all *sensekey*. This flag modifies the behavior of *sensekey*, and makes it receive events from the FLTK window instead of stdin.

Examples

Here is an example of the FLpanel opcode. It uses the file *FLpanel.csd* [examples/FLpanel.csd].

Exemple 165. Example of the FLpanel opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Creates an empty window panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

; Panel height in pixels
ipanelheight = 900
; Panel width in pixels
ipanelwidth = 400
; Horizontal position of the panel on screen in pixels
ix = 50
; Vertical position of the panel on screen in pixels
iy = 50

FLpanel "A Window Panel", ipanelheight, ipanelwidth, ix, iy
; End of panel contents
FLpanelEnd

;Run the widget thread!
FLrun

instr 1
endin


</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*, *sensekey*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLpanelEnd

FLpanelEnd — Marks the end of a group of FLTK widgets contained inside of a window (panel).

Description

Marks the end of a group of FLTK widgets contained inside of a window (panel).

Syntax

```
FLpanelEnd
```

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLpanel_end

FLpanel_end — Marks the end of a group of FLTK widgets contained inside of a window (panel).

Description

Marks the end of a group of FLTK widgets contained inside of a window (panel). This is another name for **FLpanelEnd** provided for compatibility. See *FLpanelEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLprintk

FLprintk — A FLTK opcode that prints a k-rate value at specified intervals.

Description

FLprintk is similar to *printk* but shows values of a k-rate signal in a text field instead of on the console.

Syntax

```
FLprintk itime, kval, idisp
```

Initialization

itime -- how much time in seconds is to elapse between updated displays.

idisp -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

Performance

kval -- k-rate signal to be displayed.

FLprintk is similar to *printk*, but shows values of a k-rate signal in a text field instead of showing it in the console. The *idisp* argument must be filled with the *ihandle* return value of a previous *FLvalue* opcode. While *FLvalue* should be placed in the header section of an orchestra inside an *FLpanel*/*FLpanelEnd* block, *FLprintk* must be placed inside an instrument to operate correctly. For this reason, it slows down performance and should be used for debugging purposes only.

See Also

FLbox, *FLbutBank*, *FLbutton*, *FLprintk2*, *FLvalue*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLprintk2

FLprintk2 — A FLTK opcode that prints a new value every time a control-rate variable changes.

Description

FLprintk2 is similar to *FLprintk* but shows a k-rate variable's value only when it changes.

Syntax

```
FLprintk2 kval, idisp
```

Initialization

idisp -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

Performance

kval -- k-rate signal to be displayed.

FLprintk2 is similar to *FLprintk*, but shows the k-rate variable's value only each time it changes. Useful for monitoring MIDI control changes when using sliders. It should be used for debugging purposes only, since it slows-down performance.

See Also

FLbox, *FLbutBank*, *FLbutton*, *FLprintk*, *FLvalue*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLroller

FLroller — A FLTK widget that creates a transversal knob.

Description

FLroller is a sort of knob, but put transversally.

Syntax

```
kout, ihandle FLroller "label", imin, imax, istep, iexp, itype, idisp, \
      iwidth, iheight, ix, iy
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLroller* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

imin -- minimum value of output range.

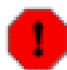
imax -- maximum value of output range.

istep -- a floating-point number indicating the increment of valuator value corresponding to of each mouse click. The *istep* argument allows the user to arbitrarily slow roller's motion, enabling arbitrary precision.

iexp -- an integer number denoting the behaviour of valuator:

- 0 = valuator output is linear
- -1 = valuator output is exponential

All other positive numbers for *iexp* indicate the number of an existing table that is used for indexing. Linear interpolation is provided in table indexing. A negative table number suppresses interpolation.



IMPORTANT!

Notice that the tables used by valuators must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. In fact, tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

itype -- an integer number denoting the appearance of the valuator.

The *itype* argument can be set to the following values:

- 1 - horizontal roller
- 2 - vertical roller

idisp -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

kout -- output value

FLroller is a sort of knob, but put transversally:

FLroller.

Examples

Here is an example of the FLroller opcode. It uses the file *FLroller.csd* [examples/FLroller.csd].

Exemple 166. Example of the FLroller opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLroller.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flroller controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Roller", 900, 400, 50, 50
    ; Minimum value output by the roller
    imin = 200
    ; Maximum value output by the roller
    imax = 5000
    ; Increment with each pixel
```

```
    istep = 1
    ; Logarithmic type roller selected
    iexp = -1
    ; Roller graphic type (1=horizontal)
    itype = 1
    ; Display handle (-1=not used)
    idisp = -1
    ; Width of the roller in pixels
    iwidth = 300
    ; Height of the roller in pixels
    iheight = 50
    ; Distance of the left edge of the knob
    ; from the left edge of the panel
    ix = 300
    ; Distance of the top edge of the knob
    ; from the top edge of the panel
    iy = 50

    gkfreq, ihandle FLroller "Frequency", imin, imax, istep, iexp, itype, idisp, iwidth, iheight, ix, i
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLcount, *FLjoy*, *FLknob*, *FLslider*, *FLtext*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLrun

FLrun — Starts the FLTK widget thread.

Description

Starts the FLTK widget thread.

Syntax

```
FLrun
```

Performance

This opcode must be located at the end of all widget declarations. It has no arguments, and its purpose is to start the thread related to widgets. Widgets would not operate if *FLrun* is missing.

See Also

FLgetsnap, *FLloadsnap*, *FLsavesnap*, *FLsetsnap*, *FLupdate*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsavesnap

FLsavesnap — Saves all snapshots currently created into a file.

Description

FLsavesnap saves all snapshots currently created (i.e. the entire memory bank) into a file.

Syntax

```
FLsavesnap "filename" [, igroup]
```

Initialization

« filename » -- a double-quoted string corresponding to a file to store a bank of snapshots.

igroup -- (optional) an integer number referring to a snapshot-related group of widget. It allows to get/set, or to load/save the state of a subset of valuators. Default value is zero that refers to the first group. The group number is determined by the opcode *FLsetSnapGroup*.

Note

The *igroup* parameter has not been yet fully implemented in the current version of csound. Please do not rely on it yet.

Performance

FLsavesnap saves all snapshots currently created (i.e. the entire memory bank) into a file whose name is *filename*. Since the file is a text file, snapshot values can also be edited manually by means of a text editor. The format of the data stored in the file is the following (at present time, this could be changed in next Csound version):

```
----------- 0 -----------
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 331.946 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 0 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----------- 1 -----------
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 819.72 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 1 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----------- 2 -----------
```

```
..... etc...
----------- 3 -----------
..... etc...
---------------------------
```

As you can see, each snapshot contain several lines. Each snapshot is separated from previous and next snapshot by a line of this kind:

"----------- snapshot Num -----------"

Then there are several lines containing data. Each of these lines corresponds to a widget.

The first field of each line is an unquoted string containing opcode name corresponding to that widget. Second field is a number that expresses current value of a snapshot. In current version, this is the only field that can be modified manually. The third and fourth fields shows minimum and maximum values allowed for that valuator. The fifth field is a special number that indicates if the valuator is linear (value 0), exponential (value -1), or is indexed by a table interpolating values (negative table numbers) or non-interpolating (positive table numbers). The last field is a quoted string with the label of the widget. Last line of the file is always

"---------------------------"

.

Note that *FLvalue* and*FLbox* are not valuators and their values are fixed, so they cannot be modified.

For purposes of snapshot saving, widgets can be grouped, so that snapshots affect only a defined group of widgets. The opcode *FLsetSnapGroup* is used to specify the group for all widgets declared after it, until the next *FLsetSnapGroup* statement.

Examples

Here is a simple example of the FLTK snapshot saving. It uses the file *FLsavesnap_simple.csd* [examples/FLsavesnap_simple.csd].

Exemple 167. Example of FLTK snapshot saving.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007
```

```
; giSWMtab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMtab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 190, 40, 410, 3
    FLcolor  100, 118 ,140
    ivalSM1                    FLvalue  "", 70, 20, 270, 20
    gksliderA, gislidSM1           FLslider "Slider", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
    itext1           FLbox   "store", 1, 1, 14, 50, 25, 355, 15
    itext2           FLbox   "load", 1, 1, 14, 50, 25, 415, 15
    gksnap, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
    gksnap, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
    gksnap, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
    gksnap, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4

    gkload, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
    gkload, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
    gkload, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
    gkload, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

    ivalSM2                    FLvalue  "", 70, 20, 270, 80
    gkknobA, gislidSM2           FLknob "Knob", -4, 4, 0, 3, ivalSM2, 60, 120, 60
FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLrun

        instr 1

        endin

instr 3 ; Save snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno


inumsnap, inumval  FLsetsnap index ;, -1, igroup
FLsavesnap Sfile

        endin


instr 4 ;Load snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno

FLloadsnap Sfile
inumload FLgetsnap index ;, igroup

        endin


</CsInstruments>
<CsScore>
f 0 3600

e

</CsScore>
</CsoundSynthesizer>
```

Here is another example of FLTK snapshot saving using snapshot groups. It uses the file *FLsavesnap.csd* [examples/FLsavesnap.csd].

Exemple 168. Example of FLTK snapshot saving using snapshot groups.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007

; giSWMtab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMtab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 350, 40, 410, 3
    FLcolor  100, 118 ,140
   FLsetSnapGroup  0
    ivalSM1                  FLvalue  "", 70, 20, 270, 20
    ivalSM2                 FLvalue  "", 70, 20, 270, 60
    ivalSM3                 FLvalue  "", 70, 20, 270, 100
    ivalSM4                 FLvalue  "", 70, 20, 270, 140
    gksliderA, gislidSM1         FLslider "Slider A", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
    gksliderB, gislidSM2         FLslider "Slider B", 1, 10, 0, 3, ivalSM2, 250, 20, 20, 60
    gksliderC, gislidSM3         FLslider "Slider C", 0, 1, 0, 3, ivalSM3, 250, 20, 20, 100
    gksliderD, gislidSM4         FLslider "Slider D", 0, 1, 0, 3, ivalSM4, 250, 20, 20, 140
    itext1          FLbox   "store", 1, 1, 14, 50, 25, 355, 15
    itext2          FLbox   "load", 1, 1, 14, 50, 25, 415, 15
    itext3          FLbox   "G\nr\no\nu\np\n \n1", 1, 1, 14, 30, 145, 485, 15
    gksnap, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
    gksnap, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
    gksnap, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
    gksnap, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4
    gkload, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
    gkload, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
    gkload, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
    gkload, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

    FLcolor  100, 140 ,118
   FLsetSnapGroup  1
    ivalSM5                  FLvalue  "", 70, 20, 270, 190
    ivalSM6                 FLvalue  "", 70, 20, 270, 230
    ivalSM7                 FLvalue  "", 70, 20, 270, 270
    ivalSM8                 FLvalue  "", 70, 20, 270, 310
    gkknobA, gislidSM5         FLknob "Knob A", -4, 4, 0, 3, ivalSM5, 45, 10, 230
    gkknobB, gislidSM6         FLknob "Knob B", 1, 10, 0, 3, ivalSM6, 45, 75, 230
    gkknobC, gislidSM7         FLknob "Knob C", 0, 1, 0, 3, ivalSM7, 45, 140, 230
    gkknobD, gislidSM8         FLknob "Knob D", 0, 1, 0, 3, ivalSM8, 45, 205, 230
    itext4          FLbox   "store", 1, 1, 14, 50, 25, 355, 185
    itext5          FLbox   "load", 1, 1, 14, 50, 25, 415, 185
    itext6          FLbox   "G\nr\no\nu\np\n \n2", 1, 1, 14, 30, 145, 485, 185
    gksnap, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 364, 215, 0, 3, 0, 3, 5
    gksnap, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 364, 245, 0, 3, 0, 3, 6
    gksnap, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 364, 275, 0, 3, 0, 3, 7
    gksnap, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 364, 305, 0, 3, 0, 3, 8
    gkload, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 424, 215, 0, 4, 0, 3, 5
    gkload, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 424, 245, 0, 4, 0, 3, 6
    gkload, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 424, 275, 0, 4, 0, 3, 7
    gkload, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 424, 305, 0, 4, 0, 3, 8
FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLsetVal_i 0, gislidSM3
FLsetVal_i 0, gislidSM4
FLsetVal_i 1, gislidSM5
FLsetVal_i 1, gislidSM6
FLsetVal_i 0, gislidSM7
FLsetVal_i 0, gislidSM8
FLrun

        instr 1

        endin

instr 3 ; Save snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
  igroup = 1
```

```
endif


  inumsnap, inumval  FLsetsnap index , -1, igroup
FLsavesnap Sfile

        endin


instr 4 ;Load snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
  igroup = 1
endif

FLloadsnap Sfile
  inumload FLgetsnap index , igroup

        endin


</CsInstruments>
<CsScore>
  ;Dummy table for FLgetsnap
; f 1 0 1024 10 1
f 0 3600

e

</CsScore>
</CsoundSynthesizer>
```

See Also

FLgetsnap, *FLloadsnap*, *FLsetSnapGroup*, *FLrun*, *FLsetsnap*, *FLupdate*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLscroll

FLscroll — A FLTK opcode that adds scroll bars to an area.

Description

FLscroll adds scroll bars to an area.

Syntax

```
FLscroll iwidth, iheight [, ix] [, iy]
```

Initialization

iwidth -- width of widget.

iheight -- height of widget.

ix (optional) -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy (optional) -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

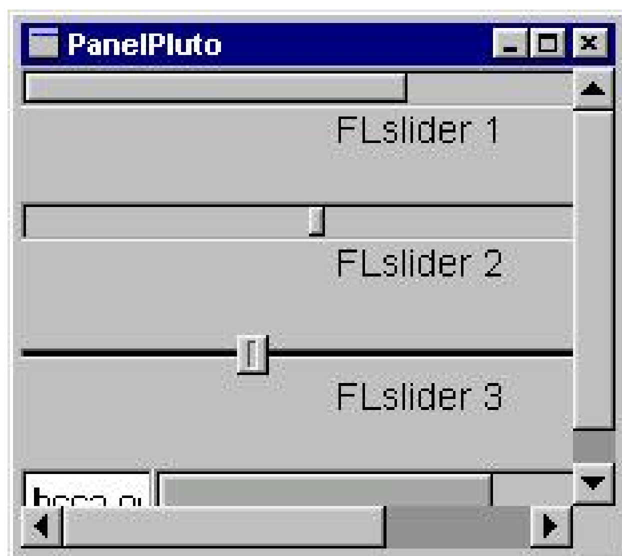
Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

FLscroll adds scroll bars to an area. Normally you must set arguments *iwidth* and *iheight* equal to that of the parent window or other parent container. *ix* and *iy* are optional since they normally are set to zero. For example the following code:

```
        FLpanel   "PanelPluto",400,300,100,100
        FLscroll  400,300
gk1,ih1 FLslider  "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2,ih2 FLslider  "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3,ih3 FLslider  "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4,ih4 FLslider  "FLslider 4", 250, 5000, 1 ,11,-1, 300,30, 20,200
        FLscrollEnd
        FLpanelEnd
```

will show scroll bars, when the main window size is reduced:



FLscroll.

Examples

Here is an example of the FLscroll opcode. It uses the file *FLscroll.csd* [examples/FLscroll.csd].

Exemple 169. Example of the FLscroll opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLscroll.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flscroll opcode which enables
; the use of widget sizes and placings beyond the
; dimensions of the containing panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 420, 200, 50, 50
    iwidth = 420
    iheight = 200
    ix = 0
    iy = 0
    FLscroll iwidth, iheight, ix, iy
    ih3 FLbox "DRAG THE SCROLL BAR TO THE RIGHT IN ORDER TO READ THE REST OF THIS TEXT!", 1, 10, 20, 87
    FLscrollEnd
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin
```

```
</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLscrollEnd

FLscrollEnd — A FLTK opcode that marks the end of an area with scrollbars.

Description

A FLTK opcode that marks the end of an area with scrollbars.

Syntax

```
FLscrollEnd
```

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLtabs*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLscroll_end

FLscroll_end — A FLTK opcode that marks the end of an area with scrollbars.

Description

A FLTK opcode that marks the end of an area with scrollbars. This is another name for **FLscrollEnd** provided for compatibility. See *FLscrollEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetAlign

FLsetAlign — Sets the text alignment of a label of a FLTK widget.

Description

FLsetAlign sets the text alignment of the label of the target widget.

Syntax

```
FLsetAlign ialign, ihandle
```

Initialization

ialign -- sets the alignment of the label text of widgets.

The legal values for the *ialign* argument are:

- 1 - align center
- 2 - align top
- 3 - align bottom
- 4 - align left
- 5 - align right
- 6 - align top-left
- 7 - align top-right
- 8 - align bottom-left
- 9 - align bottom-right

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetBox

FLsetBox — Sets the appearance of a box surrounding a FLTK widget.

Description

FLsetBox sets the appearance of a box surrounding the target widget.

Syntax

```
FLsetBox itype, ihandle
```

Initialization

itype -- an integer number that modify the appearance of the target widget.

Legal values for the *itype* argument are:

- 1 - flat box
- 2 - up box
- 3 - down box
- 4 - thin up box
- 5 - thin down box
- 6 - engraved box
- 7 - embossed box
- 8 - border box
- 9 - shadow box
- 10 - rounded box
- 11 - rounded box with shadow
- 12 - rounded flat box
- 13 - rounded up box
- 14 - rounded down box
- 15 - diamond up box
- 16 - diamond down box
- 17 - oval box
- 18 - oval shadow box

- 19 - oval flat box

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetColor

FLsetColor — Sets the primary color of a FLTK widget.

Description

FLsetColor sets the primary color of the target widget.

Syntax

```
FLsetColor ired, igreen, iblue, ihandle
```

Initialization

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

Examples

Here is an example of the FLsetcolor opcode. It uses the file *FLsetcolor.csd* [examples/FLsetcolor.csd].

Exemple 170. Example of the FLsetcolor opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLsetcolor.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flsetcolor to change from the
; default colours for widgets
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Coloured Sliders", 900, 360, 50, 50
    gkfreq, ihandle FLslider "A Red Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 50
    ired1 = 255
    igreen1 = 0
    iblue1 = 0
    FLsetColor ired1, igreen1, iblue1, ihandle
```

```
    gkfreq, ihandle FLslider "A Green Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 150
    ired1 = 0
    igreen1 = 255
    iblue1 = 0
    FLsetColor ired1, igreen1, iblue1, ihandle

    gkfreq, ihandle FLslider "A Blue Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 250
    ired1 = 0
    igreen1 = 0
    iblue1 = 255
    FLsetColor ired1, igreen1, iblue1, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin


</CsInstruments>
<CsScore>

; 'Dummy' score event for 1 hour.
f 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLsetColor2

FLsetColor2 — Sets the secondary (or selection) color of a FLTK widget.

Description

FLsetColor2 sets the secondary (or selection) color of the target widget.

Syntax

```
FLsetColor2 ired, igreen, iblue, ihandle
```

Initialization

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetFont

FLsetFont — Sets the font type of a FLTK widget.

Description

FLsetFont sets the font type of the target widget.

Syntax

```
FLsetFont ifont, ihandle
```

Initialization

ifont -- sets the the font type of the label of a widget.

Legal values for ifont argument are:

- 1 - Helvetica (same as Arial under Windows)
- 2 - Helvetica Bold
- 3 - Helvetica Italic
- 4 - Helvetica Bold Italic
- 5 - Courier
- 6 - Courier Bold
- 7 - Courier Italic
- 8 - Courier Bold Italic
- 9 - Times
- 10 - Times Bold
- 11 - Times Italic
- 12 - Times Bold Italic
- 13 - Symbol
- 14 - Screen
- 15 - Screen Bold
- 16 - Dingbats

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value di-

rectly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetPosition

FLsetPosition — Sets the position of a FLTK widget.

Description

FLsetPosition sets the position of the target widget according to the *ix* and *iy* arguments.

Syntax

```
FLsetPosition ix, iy, ihandle
```

Initialization

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetSize

FLsetSize — Resizes a FLTK widget.

Description

FLsetSize resizes the target widget (not the size of its text) according to the *iwidth* and *iheight* arguments.

Syntax

```
FLsetSize iwidth, iheight, ihandle
```

Initialization

iwidth -- width of widget.

iheight -- height of widget.

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetsnap

FLsetsnap — Stores the current status of all FLTK valuators into a snapshot location.

Description

FLsetsnap stores the current status of all valuators present in the orchestra into a snapshot location (in memory).

Syntax

```
inumsnap, inumval FLsetsnap index [, ifn, igroup]
```

Initialization

inumsnap -- current number of snapshots.

inumval -- number of valuators (whose value is stored in a snapshot) present in current orchestra.

index -- a number referring unequivocally to a snapshot. Several snapshots can be stored in the same bank.

ifn (optional) -- optional argument referring to an already allocated table, to store values of a snapshot.

igroup -- (optional) an integer number referring to a snapshot-related group of widget. It allows to get/set, or to load/save the state of a subset of valuators. Default value is zero that refers to the first group. The group number is determined by the opcode *FLsetSnapGroup*.

Note

The *igroup* parameter has not been yet fully implemented in the current version of csound. Please do not rely on it yet.

Performance

The *FLsetsnap* opcode stores current status of all valuators present in the orchestra into a snapshot location (in memory). Any number of snapshots can be stored in the current bank. Banks are structures that only exist in memory, there are no other reference to them other that they can be accessed by *FLsetsnap*, *FLsavesnap*, *FLloadsnap* and *FLgetsnap* opcodes. Only a single bank can be present in memory.

If the optional *ifn* argument refers to an already allocated and valid table, the snapshot will be stored in the table instead of in the bank. So that table can be accessed from other Csound opcodes.

The *index* argument unequivocally refers to a determinate snapshot. If the value of *index* refers to a previously stored snapshot, all its old values will be replaced with current ones. If *index* refers to a snapshot that doesn't exist, a new snapshot will be created. If the *index* value is not adjacent with that of a previously created snapshot, some empty snapshots will be created. For example, if a location with *index* 0 contains the only and unique snapshot present in a bank and the user stores a new snapshot using *index* 5, all locations between 1 and 4 will automatically contain empty snapshots. Empty snapshots don't contain any data and are neutral.

FLsetsnap outputs the current number of snapshots (the *inumsnap* argument) and the total number of va-

lues stored in each snapshot (*inumval*). *inumval* is equal to the number of valuators present in the orchestra.

For purposes of snapshot saving, widgets can be grouped, so that snapshots affect only a defined group of widgets. The opcode *FLsetSnapGroup* is used to specify the group for all widgets declared after it, until the next *FLsetSnapGroup* statement.

See Also

FLgetsnap, *FLloadsnap*, *FLsetSnapGroup*, *FLrun*, *FLsavesnap*, *FLupdate*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetSnapGroup

FLsetSnapGroup — Determines the snapshot group for FL valuators.

Description

FLsetSnapGroup determines the snapshot group of valuators declared after it.

Syntax

```
FLsetSnapGroup igroup
```

Initialization

igroup -- (optional) an integer number referring to a snapshot-related group of widget. It allows to get/set, or to load/save the state of a subset of valuators.

Note

The *igroup* parameter has not been yet fully implemented in the current version of csound. Please do not rely on it yet.

Performance

For purposes of snapshot saving, widgets can be grouped, so that snapshots affect only a defined group of widgets. The opcode *FLsetSnapGroup* is used to specify the group for all widgets declared after it, until the next *FLsetSnapGroup* statement.

FLsetSnapGroup determines the snapshot group of a declared valuator. To make a valuator belong to a stated group, you have to place *FLsetSnapGroup* just before the declaration of the widget itself. The group stated by *FLsetSnapGroup* lasts for all valuators declared after it, until a new *FLsetSnapGroup* statement with a different group is encountered. If no *FLsetSnapGroup* statement are present in an orchestra, the default group for all widgets will be group zero.

See Also

FLgetsnap, *FLsetsnap*, *FLloadsnap*, *FLsavesnap*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetText

FLsetText — Sets the label of a FLTK widget.

Description

FLsetText sets the label of the target widget to the double-quoted text string provided with the *itext* argument.

Syntax

```
FLsetText "itext", ihandle
```

Initialization

« itext » -- a double-quoted string denoting the text of the label of the widget.

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

Examples

Here is an example of the FLsetText opcode. It uses the file *FLsetText.csd* [examples/FLsetText.csd].

Exemple 171. Example of the FLsetText opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLsetText.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

; Example by Giorgio Zucco and Andres Cabrera 2007

FLpanel "FLsetText",250,100,50,50

gk1,giha FLcount "", 1, 20, 1, 20, 1, 200, 40, 20, 20, 0, 1, 0, 1

FLpanelEnd
FLrun


    instr 1
; This instrument is triggered by FLcount above each time
```

```
; its value changes
iname = i(gk1)
print iname
; Must use FLsetText on the init pass!
if (iname == 1) igoto text1
if (iname == 2) igoto text2
if (iname == 3) igoto text3

igoto end

text1:
FLsetText "FM",giha
igoto end

text2:
FLsetText "GRANUL",giha
igoto end

text3:
FLsetText "PLUCK",giha
igoto end

end:
    endin


</CsInstruments>
<CsScore>

f 0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetTextColor

FLsetTextColor — Sets the color of the text label of a FLTK widget.

Description

FLsetTextColor sets the color of the text label of the target widget.

Syntax

```
FLsetTextColor ired, iblue, igreen, ihandle
```

Initialization

ired -- The red color of the target widget. The range for each RGB component is 0-255

igreen -- The green color of the target widget. The range for each RGB component is 0-255

iblue -- The blue color of the target widget. The range for each RGB component is 0-255

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetTextSize

FLsetTextSize — Sets the size of the text label of a FLTK widget.

Description

FLsetTextSize sets the size of the text label of the target widget.

Syntax

```
FLsetTextSize isize, ihandle
```

Initialization

isize -- size of the font of the target widget. Normal values are in the order of 15. Greater numbers enlarge font size, while smaller numbers reduce it.

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetTextType

FLsetTextType — Sets some font attributes of the text label of a FLTK widget.

Description

FLsetTextType sets some attributes related to the fonts of the text label of the target widget.

Syntax

```
FLsetTextType itype, ihandle
```

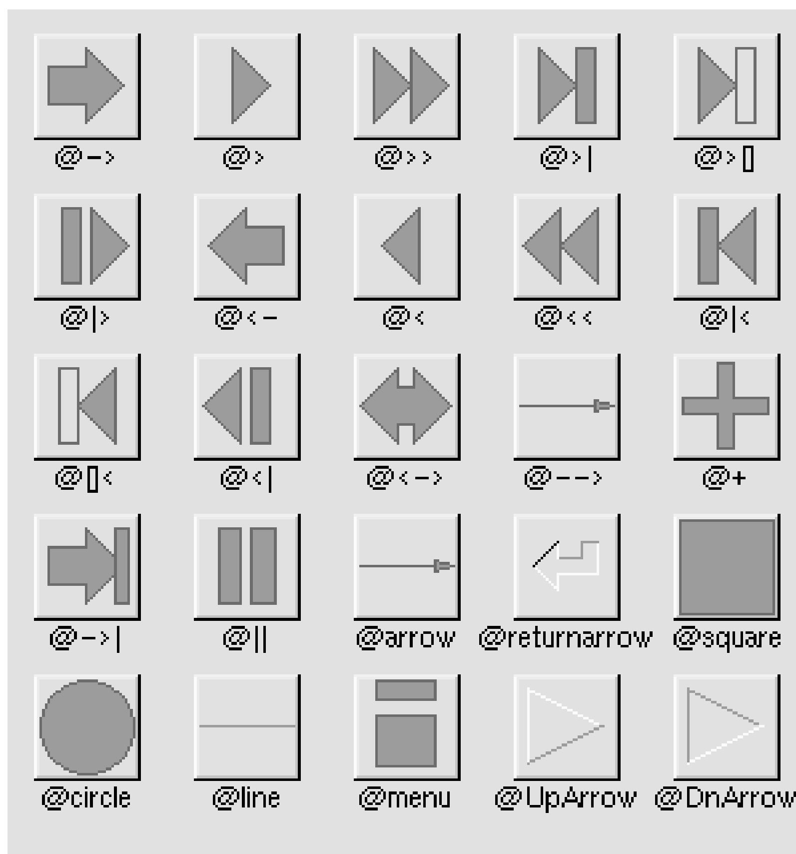
Initialization

itype -- an integer number that modify the appearance of the target widget.

The legal values of *itype* are:

- 0 - normal label
- 1 - no label (hides the text)
- 2 - symbol label (see below)
- 3 - shadow label
- 4 - engraved label
- 5- embossed label
- 6- bitmap label (not implemented yet)
- 7- pixmap label (not implemented yet)
- 8- image label (not implemented yet)
- 9- multi label (not implemented yet)
- 10- free-type label (not implemented yet)

When using *itype*=3 (symbol label), it is possible to assign a graphical symbol instead of the text label of the target widget. In this case, the string of the target label must always start with « @ ». If it starts with something else (or the symbol is not found), the label is drawn normally. The following symbols are supported:



FLTK label supported symbols.

The @ sign may be followed by the following optional « formatting » characters, in this order:

1. « # » forces square scaling rather than distortion to the widget's shape.
2. +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
3. [1-9] rotates by a multiple of 45 degrees. « 6 » does nothing, the others point in the direction of that key on a numeric keypad.

Notice that with *FLbox* and *FLbutton*, it is not necessary to call *FLsetTextType* opcode at all in order to use a symbol. In this case, it is sufficient to set a label starting with « @ » followed by the proper formatting string.

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetVal_i

FLsetVal_i — Sets the value of a FLTK valuator to a number provided by the user.

Description

FLsetVal_i forces the value of a valuator to a number provided by the user.

Syntax

```
FLsetVal_i ivalue, ihandle
```

Initialization

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

Performance

ivalue -- Value to set the widget to.

Note

FLsetVal is not fully implemented yet, and may crash in certain cases (e.g. when setting the value of a widget connected to a *FLvalue* widget- in this case use two separate *FLsetVal_i*).

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLsetVal

FLsetVal — Sets the value of a FLTK valuator at control-rate.

Description

FLsetVal is almost identical to *FLsetVal_i*. Except it operates at k-rate and it affects the target valuator only when *ktrig* is set to a non-zero value.

Syntax

```
FLsetVal ktrig, kvalue, ihandle
```

Initialization

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

Performance

ktrig -- triggers the opcode when different than 0.

kvalue -- Value to set the widget to.

Note

FLsetVal is not fully implemented yet, and may crash in certain cases (e.g. when setting the value of a widget connected to a *FLvalue* widget- in this case use two separate *FLsetVal*)

See Also

FLcolor, *FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLshow

FLshow — Restores the visibility of a previously hidden FLTK widget.

Description

FLshow restores the visibility of a previously hidden widget.

Syntax

```
FLshow ihandle
```

Initialization

ihandle -- an integer number (used as unique identifier) taken from the output of a previously located widget opcode (which corresponds to the target widget). It is used to unequivocally identify the widget when modifying its appearance with this class of opcodes. The user must not set the *ihandle* value directly, otherwise a Csound crash will occur.

See Also

FLcolor2, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal_i*, *FLsetVal*, *FLshow*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLslidBnk

FLslidBnk — A FLTK widget containing a bank of horizontal sliders.

Description

FLslidBnk is a widget containing a bank of horizontal sliders.

Syntax

```
FLslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
      [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

Initialization

« names » -- a double-quoted string containing the names of each slider. Each slider can have a different name. Separate each name with « @ » character, for example: « frequency@amplitude@cutoff ». It is possible to not provide any name by giving a single space «   ». In this case, the opcode will automatically assign a progressive number as a label for each slider.

inumsliders -- the number of sliders.

ioutable (optional, default=0) -- number of a previously-allocated table in which to store output values of each slider. The user must be sure that table size is large enough to contain all output cells, otherwise a segfault will crash Csound. By assigning zero to this argument, the output will be directed to the zak space in the k-rate zone. In this case, the zak space must be previously allocated with the *zakinit* opcode and the user must be sure that the allocation size is big enough to cover all sliders. The default value is zero (i.e. store output in zak space).

istart_index (optional, default=0) -- an integer number referring to a starting offset of output cell locations. It can be positive to allow multiple banks of sliders to output in the same table or in the zak space. The default value is zero (no offset).

iminmaxtable (optional, default=0) -- number of a previously-defined table containing a list of min-max pairs, referred to each slider. A zero value defaults to the 0 to 1 range for all sliders without necessity to provide a table. The default value is zero.

iexptable (optional, default=0) -- number of a previously-defined table containing a list of identifiers (i.e. integer numbers) provided to modify the behaviour of each slider independently. Identifiers can assume the following values:

- -1 -- exponential curve response
- 0 -- linear response
- number > than 0 -- follow the curve of a previously-defined table to shape the response of the corresponding slider. In this case, the number corresponds to table number.

You can assume that all sliders of the bank have the same response curve (exponential or linear). In this case, you can assign -1 or 0 to *iexptable* without worrying about previously defining any table. The default value is zero (all sliders have a linear response, without having to provide a table).

itypetable (optional, default=0) -- number of a previously-defined table containing a list of identifiers

(i.e. integer numbers) provided to modify the aspect of each individual slider independently. Identifiers can assume the following values:

- 0 = Nice slider
- 1 = Fill slider
- 3 = Normal slider
- 5 = Nice slider
- 7 = Nice slider with down-box

You can assume that all sliders of the bank have the same aspect. In this case, you can assign a negative number to *itypetable* without worrying about previously defining any table. Negative numbers have the same meaning of the corresponding positive identifiers with the difference that the same aspect is assigned to all sliders. You can also assign a random aspect to each slider by setting *itypetable* to a negative number lower than -7. The default value is zero (all sliders have the aspect of nice sliders, without having to provide a table).

You can add 20 to a value inside the table to make the slider "plastic", or subtract 20 if you want to set the value for all widgets without defining a table (e.g. -21 to set all sliders types to Plastic Fill slider).

iwidth (optional) -- width of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

iheight (optional) -- height of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

ix (optional) -- horizontal position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

iy (optional) -- vertical position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

Performance

There are no k-rate arguments, even if cells of the output table (or the zak space) are updated at k-rate.

FLslidBnk is a widget containing a bank of horizontal sliders. Any number of sliders can be placed into the bank (*inumsliders* argument). The output of all sliders is stored into a previously allocated table or into the zak space (*ioutable* argument). It is possible to determine the first location of the table (or of the zak space) in which to store the output of the first slider by means of *istart_index* argument.

Each slider can have an individual label that is placed to the left of it. Labels are defined by the *« names »* argument. The output range of each slider can be individually set by means of an external table (*iminmaxtable* argument). The curve response of each slider can be set individually, by means of a list of identifiers placed in a table (*iexptable* argument). It is possible to define the aspect of each slider independently or to make all sliders have the same aspect (*itypetable* argument).

The *iwidth*, *iheight*, *ix*, and *iy* arguments determine width, height, horizontal and vertical position of the rectangular area containing sliders. Notice that the label of each slider is placed to the left of them and is not included in the rectangular area containing sliders. So the user should leave enough space to the left of the bank by assigning a proper *ix* value in order to leave labels visible.

IMPORTANT!

Notice that the tables used by *FLslidBnk* must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. This is because tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

Examples

Here is an example of the FLslidBnk opcode. It uses the file *FLslidBnk.csd* [examples/FLslidBnk.csd].

Exemple 172. Example of the FLslidBnk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLslidBnk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gitypetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 380, 50, 50
    ;Number of sliders
    inum = 8
    ; Table to store output
    iouttable = giouttable
    ; Width of the slider bank in pixels
    iwidth = 350
    ; Height of the slider in pixels
    iheight = 160
    ; Distance of the left edge of the slider
    ; from the left edge of the panel
    ix = 30
    ; Distance of the top edge of the slider
    ; from the top edge of the panel
    iy = 10
    ; Table containing fader types
    itypetable  = gitypetable
    FLslidBnk "1@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
      , iy , itypetable
    FLslidBnk "1@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
      , iy + 200 , -23
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun


instr 1
;Dummy instrument
endin


</CsInstruments>
<CsScore>
```

```
; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLslider, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLslidBnk2

FLslidBnk2 — A FLTK widget containing a bank of horizontal sliders.

Description

FLslidBnk2 is a widget containing a bank of horizontal sliders.

Syntax

```
FLslidBnk2 "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart_index]
```

Initialization

« names » -- a double-quoted string containing the names of each slider. Each slider can have a different name. Separate each name with « @ » character, for example: « frequency@amplitude@cutoff ». It is possible to not provide any name by giving a single space «   ». In this case, the opcode will automatically assign a progressive number as a label for each slider.

inumsliders -- the number of sliders.

ioutable (optional, default=0) -- number of a previously-allocated table in which to store output values of each slider. The user must be sure that table size is large enough to contain all output cells, otherwise a segfault will crash Csound. By assigning zero to this argument, the output will be directed to the zak space in the k-rate zone. In this case, the zak space must be previously allocated with the *zakinit* opcode and the user must be sure that the allocation size is big enough to cover all sliders. The default value is zero (i.e. store output in zak space).

iconfigtable -- in the *FLslidBnk2* and FLvslidBnk2 opcodes, this table replaces *iminmaxtable*, *iexptable* and *istyletable*, all these parameters being placed into a single table. This table has to be filled with a group of 5 parameters for each slider in this way:

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

for example using GEN02 you can type:

inum *ftgen* 1,0,256, -2,   0,1,0,1,   100, 5000, -1, 3,   50, 200, -1, 5,….. [etcetera]

In this example the first slider will be affected by the [0,1,0,1] parameters (the range will be 0 to 1, it will have linear response, and its aspect will be a fill slider), the second slider will be affected by the [100,5000,-1,3] parameters (the range is 100 to 5000, the response is exponential and the aspect is a normal slider), the third slider will be affected by the [50,200,-1,5] parameters (the range is 50 to 200, the behavior exponential, and the aspect is a nice slider), and so on.

iwidth (optional) -- width of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

iheight (optional) -- height of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

ix (optional) -- horizontal position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make

sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

iy (optional) -- vertical position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

istart_index (optional, default=0) -- an integer number referring to a starting offset of output cell locations. It can be positive to allow multiple banks of sliders to output in the same table or in the zak space. The default value is zero (no offset).

Performance

There are no k-rate arguments, even if cells of the output table (or the zak space) are updated at k-rate.

FLslidBnk2 is a widget containing a bank of horizontal sliders. Any number of sliders can be placed into the bank (*inumsliders* argument). The output of all sliders is stored into a previously allocated table or into the zak space (*ioutable* argument). It is possible to determine the first location of the table (or of the zak space) in which to store the output of the first slider by means of *istart_index* argument.

Each slider can have an individual label that is placed to the left of it. Labels are defined by the *« names »* argument. The output range of each slider can be individually set by means of the *min* and *max* values inside the *iconfigtable* table. The curve response of each slider can be set individually, by means of a list of identifiers placed in the *iconfigtable* table (*exp* argument). It is possible to define the aspect of each slider independently or to make all sliders have the same aspect (*style* argument in the *iconfigtable* table).

The *iwidth*, *iheight*, *ix*, and *iy* arguments determine width, height, horizontal and vertical position of the rectangular area containing sliders. Notice that the label of each slider is placed to the left of them and is not included in the rectangular area containing sliders. So the user should leave enough space to the left of the bank by assigning a proper *ix* value in order to leave labels visible.

IMPORTANT!

Notice that the tables used by *FLslidBnk2* must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. This is because tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

Examples

Here is an example of the FLslidBnk2 opcode. It uses the file *FLslidBnk2.csd* [examples/FLslidBnk2.csd].

Exemple 173. Example of the FLslidBnk2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc       -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 100
nchnls = 2

;Example by Gabriel Maldonado

giElem init 8
giOutTab ftgen 1,0,128, 2,         0

              ;min1, max1, exp1, type1, min2, max2, exp2, type2, min3, max3, exp3, type3 etc.
giConfigTab ftgen 2,0,128,-2,         .1, 1000, -1, 3,      .1, 1000, -1, 3,     .1, 1000, -1, 3,     3
                                        .1, 5000, -1,  5,      .1, 5000, -1,  5,     .1, 5000, -1,  5,
giSine ftgen 3,0,256,10, 1

        FLpanel "This Panel contains a Slider Bank",600,600
        FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, giConfigTab, 400, 5
        FLpanel_end

        FLrun

        instr 1

kmodindex1         init 0
kmodindex2         init 0
kmodindex3         init 0
kamp                 init 0
kfreq1                 init 0
kfreq2           init 0
kfreq3          init 0
kfreq4                 init 0


       vtable1k  giOutTab, kmodindex1 , kmodindex2, kmodindex3, kamp, kfreq1, kfreq2 , kfreq3, kfreq4

amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,       kfreq4+amod1+amod2+amod3, giSine

        outs aout, aout
        endin


</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

FLslider, *FLslidBnk*, *FLvslidBnk*, *FLvslidBnk2*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslidBnkGetHandle

FLslidBnkGetHandle — gets the handle of last slider bank created.

Description

FLslidBnkGetHandle gets the handle of last slider bank created.

Syntax

```
ihandle FLslidBnkGetHandle
```

Initialization

ihandle - handle of the sliderBnk (to be used to set its values).

Performance

There are no k-rate arguments, even if cells of the output table (or the zak space) are updated at k-rate.

FLslidBnkGetHandle gets the handle of last slider bank created. This opcode must follow corresponding *FLslidBnk* (or *FLvslidBnk*, *FLslidBnk2* and *FLvslidBnk2*) immediately, in order to get its handle.

See the entry for *FLslidBnk2Setk* to see an example of usage.

See Also

FLslider, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnk2Set*, *FLslidBnk2Setk*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslidBnkSet

FLslidBnkSet — modify the values of a slider bank.

Description

FLslidBnkSet modifies the values of a slider bank according to an array of values stored in a table.

Syntax

```
FLslidBnkSet ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

Initialization

ihandle - handle of the sliderBnk (to be used to set its values).

ifn - number of a table containing an array of values to set each slider to.

istartIndex - (optional) starting index of the table element of to be evaluated firstly. Default value is zero

istartSlid - (optional) starting slider to be evaluated. Default 0, denoting the first slider.

inumSlid - (optional) number of sliders to be updated. Default 0, denoting all sliders.

Performance

FLslidBnkSet modifies the values of a slider bank (created with *FLslidBnk* or with *FLvslidBnk*) according to an array of values stored into table *ifn*. It actually allows to update an *FLslidBnk* (or *FLvslidBnk*) bank of sliders (for instance, using the *slider8table* opcode) to a set of values located in a table. User has to set *ihandle* argument to the handle got from *FLslidBnkGetHandle* opcode. It works at init-rate only. It is possible to reset only a range of sliders, by using the optional arguments *istartIndex*, *istartSlid*, *inumSlid*

There is a k-rate version of this opcode called *FLslidBnkSetk*.

See Also

FLslider, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2* *FLslidBnk2Set*, *FLslidBnkSetk*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslidBnkSetk

FLslidBnkSetk — modify the values of a slider bank.

Description

FLslidBnkSetk modifies the values of a slider bank according to an array of values stored in a table.

Syntax

```
FLslidBnkSetk  ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

Initialization

ihandle - handle of the sliderBnk (to be used to set its values).

ifn - number of a table containing an array of values to set each slider to.

istartIndex - (optional) starting index of the table element of to be evaluated firstly. Default value is zero

istartSlid - (optional) starting slider to be evaluated. Default 0, denoting the first slider.

inumSlid - (optional) number of sliders to be updated. Default 0, denoting all sliders.

Performance

ktrig – the output of *FLslidBnkSetk* consists of a trigger that informs if sliders have to be updated or not. A non-zero value forces the slider to be updated.

FLslidBnkSetk is similar to *FLslidBnkSet* but allows k-rate to modify the values of *FLslidBnk* (*FLslidBnkSetk* can also be used with *FLvslidBnk*, obtaining identical result). It also allows the slider bank to be joined with MIDI. If you are using MIDI (for instance, when using the *slider8table* opcode), *FLslidBnkSetk* changes the values of *FLslidBnk* bank of sliders to a set of values located in a table. This opcode is actually able to serve as a MIDI bridge to the *FLslidBnk* widget when used together with the *sliderXXtable* set of opcodes (see *slider8table* entry for more information). Notice, that, when you want to use table indexing as a curve response, it is not possible to do it directly in the *iconfigtable* configuration of *FLslidBnk2*, when you intend to use the *FLslidBnkSetk* opcode. In fact, corresponding *inputTable* element of *FLslidBnkSetk* must be set in linear mode and respect the 0 to 1 range. Even the corresponding elements of *sliderXXtable* must be set in linear mode and in the normalized range. You can do table indexing later, by using the *tab* and *tb* opcodes, and rescaling output according to max and min values. By the other hand, it is possible to use linear and exponential curve response directly, by setting the actual min-max range and flag both in the *iconfigtable* of corresponding *FLslidBnk2* and in *sliderXXtable*.

FLslidBnkSetk the k-rate version of *FLslidBnk2Set*.

See Also

FLslider, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2* *FLslidBnkSet*, *FLslidBnk2Set*, *slider8table*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslidBnk2Set

FLslidBnk2Set — modify the values of a slider bank.

Description

FLslidBnk2Set modifies the values of a slider bank according to an array of values stored in a table.

Syntax

```
FLslidBnk2Set ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

Initialization

ihandle - handle of the sliderBnk (to be used to set its values).

ifn - number of a table containing an array of values to set each slider to.

istartIndex - (optional) starting index of the table element of to be evaluated firstly. Default value is zero

istartSlid - (optional) starting slider to be evaluated. Default 0, denoting the first slider.

inumSlid - (optional) number of sliders to be updated. Default 0, denoting all sliders.

Performance

FLslidBnk2Set modifies the values of a slider bank (created with *FLslidBnk2* or with *FLvslidBnk2*) according to an array of values stored into table *ifn*. It actually allows to update an *FLslidBnk2* (or *FLvslidBnk2*) bank of sliders (for instance, using the *slider8table* opcode) to a set of values located in a table. User has to set *ihandle* argument to the handle got from *FLslidBnkGetHandle* opcode. It works at init-rate only. It is possible to reset only a range of sliders, by using the optional arguments *istartIndex*, *istartSlid*, *inumSlid*

FLslidBnk2Set is identical to *FLslidBnkSet*, but works on *FLslidBnk2* and *FLvslidBnk2* instead of *FLslidBnk* and *FLvslidBnk*.

There is a k-rate version of this opcode called *FLslidBnk2Setk*.

See Also

FLslider, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2* *FLslidBnkSet*, *FLslidBnk2Setk*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslidBnk2Setk

FLslidBnk2Setk — modify the values of a slider bank.

Description

FLslidBnk2Setk modifies the values of a slider bank according to an array of values stored in a table.

Syntax

```
FLslidBnk2Setk  ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

Initialization

ihandle - handle of the sliderBnk (to be used to set its values).

ifn - number of a table containing an array of values to set each slider to.

istartIndex - (optional) starting index of the table element of to be evaluated firstly. Default value is zero

istartSlid - (optional) starting slider to be evaluated. Default 0, denoting the first slider.

inumSlid - (optional) number of sliders to be updated. Default 0, denoting all sliders.

Performance

ktrig – the output of *FLslidBnk2Setk* consists of a trigger that informs if sliders have to be updated or not. A non-zero value forces the slider to be updated.

FLslidBnk2Setk is similar to *FLslidBnkSet* but allows k-rate to modify the values of *FLslidBnk2* (*FLslidBnk2Setk* can also be used with *FLvslidBnk2*, obtaining identical result). It also allows the slider bank to be joined with MIDI. If you are using MIDI (for instance, when using the *slider8table* opcode), *FLslidBnk2Setk* changes the values of *FLslidBnk2* bank of sliders to a set of values located in a table. This opcode is actually able to serve as a MIDI bridge to the *FLslidBnk2* widget when used together with the *sliderXXtable* set of opcodes (see *slider8table* entry for more information). Notice, that, when you want to use table indexing as a curve response, it is not possible to do it directly in the *iconfigtable* configuration of *FLslidBnk2*, when you intend to use the *FLslidBnk2Setk* opcode. In fact, corresponding *inputTable* element of *FLslidBnk2Setk* must be set in linear mode and respect the 0 to 1 range. Even the corresponding elements of *sliderXXtable* must be set in linear mode and in the normalized range. You can do table indexing later, by using the *tab* and *tb* opcodes, and rescaling output according to max and min values. By the other hand, it is possible to use linear and exponential curve response directly, by setting the actual min-max range and flag both in the *iconfigtable* of corresponding *FLslidBnk2* and in *sliderXXtable*.

FLslidBnk2Setk the k-rate version of *FLslidBnk2Set*.

Examples

Here is an example of the FLslidBnk2Setk opcode. It uses the file *FLslidBnk2Setk.csd* [examples/FLslidBnk2Setk.csd].

Exemple 174. Example of the FLslidBnk2Setk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr     = 44100
ksmps  = 10
nchnls = 2

;Example by Gabriel Maldonado 2007

giElem init 8
giOutTab ftgen 1,0,128, 2,         0
giSine ftgen 3,0,256,10, 1
giOutTab2 ftgen 4,0,128, 2,         0

itab ftgen 29, 0, 129, 5,  .002, 128, 1         ;** exponential ascending curve for slider mapping
giExpTab ftgen 30, 0, 129, -24, itab, 0, 1                     ;** rescaled curve for slider mapping


giConfigTab ftgen 2,0,128,-2,                 1,         500, -1,         13, \
                                             1,         500, -1,         13, \
                                          1,         500, -1,         13, \
                                         1,         5000, -1,         13, \
\
                                        1,         1000,         -1,          5, \
                                        1,         1000,         -1,          5, \
                                        1,         1000,         -1,          5, \
                                        1,         5000,         -1,          5

        FLpanel "Multiple FM",600,600
        FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab2, giConfigTab, 400,
giHandle FLslidBnkGetHandle

        FLpanel_end

        FLrun



        instr 1

ktrig slider8table  1, giOutTab, 0,\
\; ctl min max init func
        27, 1,         500, 3, -1, \;1 repeat rate
        28, 1,         500, 4, -1, \;2 random freq. amount
        29, 1,         500, 1, -1, \;3 random amp. amount
        30, 1, 5000, 1, -1, \;4 number of concurrent loop points
\
        31, 1,         1000, 1, -1, \;5 kloop1
        32, 1,         1000, 1, -1, \;6 kloop2
        33, 1,         1000, 1, -1, \;7 kloop3
        34, 1,         1000, 1, -1 \;8 kloop4


kmodindex1 init 0
kmodindex2 init 0
kmodindex3 init 0
kamp init 0
kfreq1 init 0
kfreq2 init 0
kfreq3 init 0
kfreq4 init 0

          vtable1k  giOutTab2, kmodindex1, kmodindex2, kmodindex3, kamp, kfreq1, kfreq2, kfreq3, kfreq4

;  *kflag, *ihandle, *ifn, *startInd, *startSlid, *numSlid;
```

```
        FLslidBnk2Setk  ktrig, giHandle, giOutTab, 0, 0, giElem
printk2 kmodindex1
printk2 kmodindex2,10
printk2 kmodindex3,20
printk2 kamp,30


amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,       kfreq4+amod1+amod2+amod3, giSine
        outs aout, aout

        endin
</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```

See Also

FLslider, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2* *FLslidBnkSet*, *FLslidBnk2Set*, *slider8table*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLslider

FLslider — Puts a slider into the corresponding FLTK container.

Description

FLslider puts a slider into the corresponding container.

Syntax

```
kout, ihandle FLslider "label", imin, imax, iexp, itype, idisp, iwidth, \
      iheight, ix, iy
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLslider* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

imin -- minimum value of output range (corresponds to the left value for horizontal sliders, and the top value for vertical sliders).

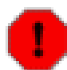
imax -- maximum value of output range (corresponds to the right value for horizontal sliders, and the bottom value for vertical sliders).

The *imin* argument may be greater than *imax* argument. This has the effect of « reversing » the object so the larger values are in the opposite direction. This also switches which end of the filled sliders is filled.

iexp -- an integer number denoting the behaviour of valuator:

- 0 = valuator output is linear
- -1 = valuator output is exponential

All other positive numbers for *iexp* indicate the number of an existing table that is used for indexing. Linear interpolation is provided in table indexing. A negative table number suppresses interpolation.



IMPORTANT!

Notice that the tables used by valuators must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. This is because tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

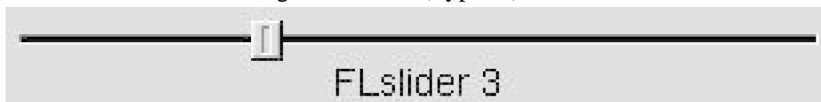
itype -- an integer number denoting the appearance of the valuator.

The *itype* argument can be set to the following values:

- 1 - shows a horizontal fill slider
- 2 - a vertical fill slider
- 3 - a horizontal engraved slider
- 4 - a vertical engraved slider
- 5 - a horizontal nice slider
- 6 - a vertical nice slider
- 7 - a horizontal up-box nice slider
- 8 - a vertical up-box nice slider

FLslider - a horizontal fill slider (itype=1).

FLslider - a horizontal engraved slider (itype=3).



FLslider - a horizontal nice slider (itype=5).
You can also create "plastic" looking sliders by adding 20 to *itype*.

idisp -- a handle value that was output from a previous instance of the *FLvalue* opcode to display the current value of the current valuator in the *FLvalue* widget itself. If the user doesn't want to use this feature that displays current values, it must be set to a negative number by the user.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

kout -- output value

FLsliders are created with the minimum value by default in the left/at the top. If you want to reverse the slider, reverse the values. See the example below.

Examples

Here is an example of the FLslider opcode. It uses the file *FLslider.csd* [examples/FLslider.csd].

Exemple 175. Example of the FLslider opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLslider.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flslider controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Slider", 900, 400, 50, 50
    ; Minimum value output by the slider
    imin = 200
    ; Maximum value output by the slider
    imax = 5000
    ; Logarithmic type slider selected
    iexp = -1
    ; Slider graphic type (5='nice' slider)
    itype = 5
    ; Display handle (-1=not used)
    idisp = -1
    ; Width of the slider in pixels
    iwidth = 750
    ; Height of the slider in pixels
    iheight = 30
    ; Distance of the left edge of the slider
    ; from the left edge of the panel
    ix = 125
    ; Distance of the top edge of the slider
    ; from the top edge of the panel
    iy = 50

    gkfreq, ihandle FLslider "Frequency", imin, imax, iexp, itype, idisp, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

;Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
    iamp = 15000
    ifn = 1
    kfreq portk gkfreq, 0.005  ;Smooth gkfreq to avoid zipper noise
    asig oscili iamp, kfreq, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the FLslider opcode, showing the slider types and other options. It also shows the usage of FLvalue to display a widget's contents. It uses the file *FLslider-2.csd* [examples/FLslider-2.csd].

Exemple 176. More complex example of the FLslider opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLslider-2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007

FLpanel "Slider Types", 410, 260, 50, 50
    ; Distance of the left edge of the slider
    ; from the left edge of the panel
    ix = 10
    ; Distance of the top edge of the slider
    ; from the top edge of the panel
    iy = 10
    ; Create boxes to display widget values
    givalue1 FLvalue "1", 60, 20, ix + 330, iy
    givalue3 FLvalue "3", 60, 20, ix + 330, iy + 40
    givalue5 FLvalue "5", 60, 20, ix + 330, iy + 80

    givalue2 FLvalue "2", 60, 20, ix + 60, iy + 140
    givalue4 FLvalue "4", 60, 20, ix + 195, iy + 140
    givalue6 FLvalue "6", 60, 20, ix + 320, iy + 140

    ;Horizontal sliders
    gkdummy1, gihandle1 FLslider "Type 1", 200, 5000, -1, 1, givalue1, 320, 20, ix, iy
    gkdummy3, gihandle3 FLslider "Type 3", 0, 15000, 0, 3, givalue3, 320, 20, ix, iy + 40
    ; Reversed slider
    gkdummy5, gihandle5 FLslider "Type 5", 1, 0, 0, 5, givalue5, 320, 20, ix, iy + 80

    ;Vertical sliders
    gkdummy2, gihandle2 FLslider "Type 2", 0, 1, 0, 2, givalue2, 20, 100, ix+ 30 , iy + 120
    ; Reversed slider
    gkdummy4, gihandle4 FLslider "Type 4", 1, 0, 0, 4, givalue4, 20, 100, ix + 165 , iy + 120
    gkdummy6, gihandle6 FLslider "Type 6", 0, 1, 0, 6, givalue6, 20, 100, ix + 290 , iy + 120
FLpanelEnd

FLpanel "Plastic Slider Types", 410, 300, 150, 150
    ; Distance of the left edge of the slider
    ; from the left edge of the panel
    ix = 10
    ; Distance of the top edge of the slider
    ; from the top edge of the panel
    iy = 10
    ; Create boxes to display widget values
    givalue21 FLvalue "21", 60, 20, ix + 330, iy
    givalue23 FLvalue "23", 60, 20, ix + 330, iy + 40
    givalue25 FLvalue "25", 60, 20, ix + 330, iy + 80

    givalue22 FLvalue "22", 60, 20, ix + 60, iy + 140
    givalue24 FLvalue "24", 60, 20, ix + 195, iy + 140
    givalue26 FLvalue "26", 60, 20, ix + 320, iy + 140

    ;Horizontal sliders
```

```
    gkdummy21, gihandle21 FLslider "Type 21", 200, 5000, -1, 21, givalue21, 320, 20, ix, iy
    gkdummy23, gihandle23 FLslider "Type 23", 0, 15000, 0, 23, givalue23, 320, 20, ix, iy + 40
    ; Reversed slider
    gkdummy25, gihandle25 FLslider "Type 25", 1, 0, 0, 25, givalue25, 320, 20, ix, iy + 80

    ;Vertical sliders
    gkdummy22, gihandle22 FLslider "Type 22", 0, 1, 0, 22, givalue22, 20, 100, ix+ 30 , iy + 120
    ; Reversed slider
    gkdummy24, gihandle24 FLslider "Type 24", 1, 0, 0, 24, givalue24, 20, 100, ix + 165 , iy + 120
    gkdummy26, gihandle26 FLslider "Type 26", 0, 1, 0, 26, givalue26, 20, 100, ix + 290 , iy + 120
    ;Button to add color to the sliders
    gkcolors, ihdummy FLbutton "Color", 1, 0, 21, 150, 30, 30, 260, 0, 10, 0, 1
FLpanelEnd
FLrun



;Set some widget's initial value
FLsetVal_i 500, gihandle1
FLsetVal_i 1000, gihandle3

instr 10
; Set the color of widgets
FLsetColor 200, 230, 0, gihandle1
FLsetColor 0, 123, 100, gihandle2
FLsetColor 180, 23, 12, gihandle3
FLsetColor 10, 230, 0, gihandle4
FLsetColor 0, 0, 0, gihandle5
FLsetColor 0, 0, 0, gihandle6

FLsetColor 200, 230, 0, givalue1
FLsetColor 0, 123, 100, givalue2
FLsetColor 180, 23, 12, givalue3
FLsetColor 10, 230, 0, givalue4
FLsetColor 255, 255, 255, givalue5
FLsetColor 255, 255, 255, givalue6

FLsetColor2 20, 23, 100, gihandle1
FLsetColor2 200,0 ,123 , gihandle2
FLsetColor2 180, 180, 100, gihandle3
FLsetColor2 180, 23, 12, gihandle4
FLsetColor2 180, 180, 100, gihandle5
FLsetColor2 180, 23, 12, gihandle6

FLsetColor 200, 230, 0, gihandle21
FLsetColor 0, 123, 100, gihandle22
FLsetColor 180, 23, 12, gihandle23
FLsetColor 10, 230, 0, gihandle24
FLsetColor 0, 0, 0, gihandle25
FLsetColor 0, 0, 0, gihandle26

FLsetColor 200, 230, 0, givalue21
FLsetColor 0, 123, 100, givalue22
FLsetColor 180, 23, 12, givalue23
FLsetColor 10, 230, 0, givalue24
FLsetColor 255, 255, 255, givalue25
FLsetColor 255, 255, 255, givalue26

FLsetColor2 20, 23, 100, gihandle21
FLsetColor2 200,0 ,123 , gihandle22
FLsetColor2 180, 180, 100, gihandle23
FLsetColor2 180, 23, 12, gihandle24
FLsetColor2 180, 180, 100, gihandle25
FLsetColor2 180, 23, 12, gihandle26

; Slider values must be updated for colors to change
FLsetVal_i 250, gihandle1
FLsetVal_i 0.5, gihandle2
FLsetVal_i 0, gihandle3
FLsetVal_i 0, gihandle4
FLsetVal_i 0, gihandle5
FLsetVal_i 0.5, gihandle6
FLsetVal_i 250, gihandle21
FLsetVal_i 0.5, gihandle22
FLsetVal_i 500, gihandle23
FLsetVal_i 0, gihandle24
FLsetVal_i 0, gihandle25
FLsetVal_i 0.5, gihandle26

endin
```

```
</CsInstruments>
<CsScore>
f 0 3600    ;Dumy table to make csound wait for realtime events

e

</CsScore>
</CsoundSynthesizer>
```

See Also

FLcount, *FLjoy*, *FLknob*, *FLroller*, *FLslidBnk*, *FLtext*

Credits

Author: Gabriel Maldonado

New in version 4.22

February 2004. Thanks to a note from Dave Phillips, deleted the extraneous istep parameter.

Example written by Iain McCurdy, edited by Kevin Conder.

FLtabs

FLtabs — Creates a tabbed FLTK interface.

Description

FLtabs is the « file card tabs » interface that allows useful to display several areas containing widgets in the same windows, alternatively. It must be used together with *FLgroup*, another container that groups child widgets.

Syntax

```
FLtabs iwidth, iheight, ix, iy
```

Initialization

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window. Expressed in pixels.

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window. Expressed in pixels.

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

FLtabs is a « file card tabs » interface that is useful to display several alternate areas containing widgets in the same window.

FLtabs.

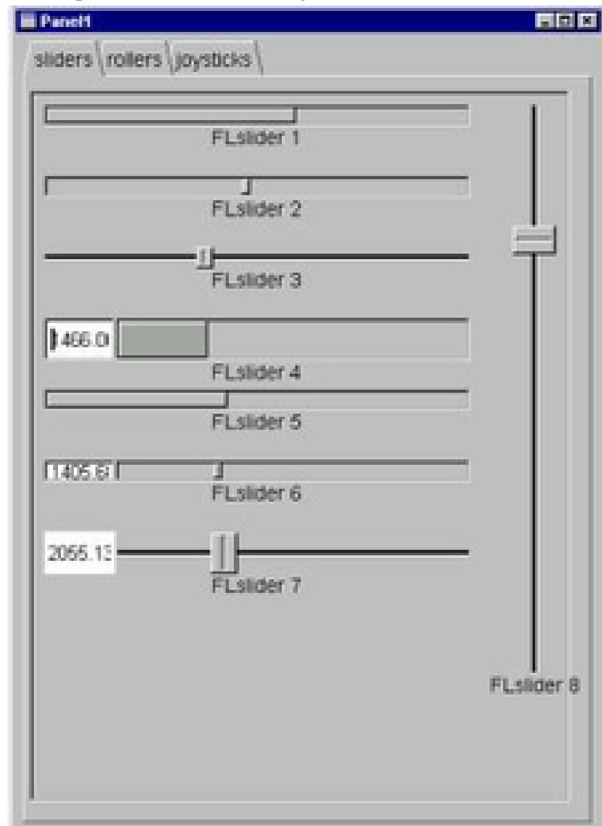
It must be used together with *FLgroup*, another FLTK container opcode that groups child widgets.

Examples

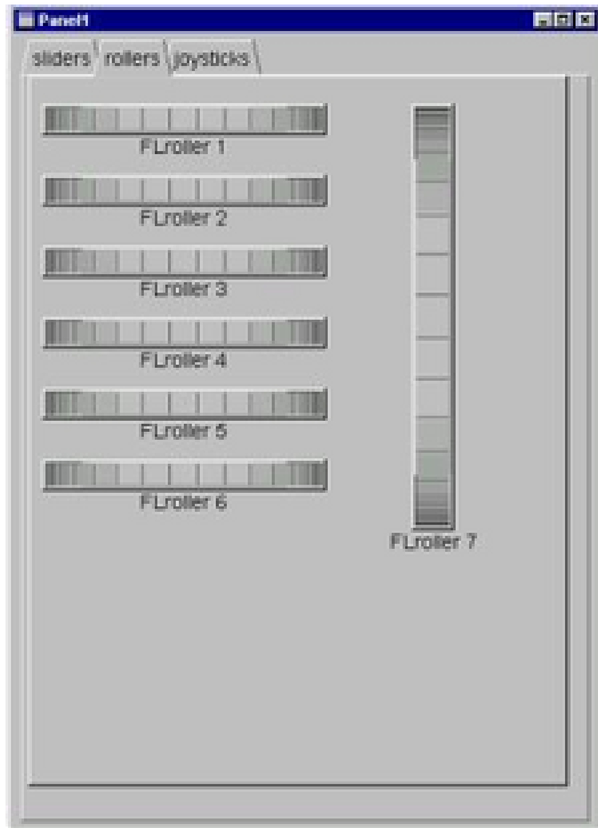
The following example code:

```
        FLpanel "Panel1",450,550,100,100
        FLscroll  450,550,0,0
        FLtabs  400,550, 5,5
        FLgroup "sliders",380,500, 10,40,1
gk1,ihs FLslider  "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2,ihs FLslider  "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3,ihs FLslider  "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4,ihs FLslider  "FLslider 4", 250, 5000, 1 ,11, -1, 300,30, 20,200
gk5,ihs FLslider  "FLslider 5", 220, 8000, 2 ,1, -1, 300,15, 20,250
gk6,ihs FLslider  "FLslider 6", 1, 5000, 1 ,13, -1, 300,15, 20,300
gk7,ihs FLslider  "FLslider 7", 870, 5000, 1 ,15, -1, 300,30, 20,350
gk8,ihs FLslider  "FLslider 8", 20, 20000, 2 ,6, -1, 30,400, 350,50
        FLgroupEnd

        FLgroup "rollers",380,500, 10,30,2
gk1,ihr FLroller  "FLroller 1", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,50
gk2,ihr FLroller  "FLroller 2", 80, 5000,1,2 ,1 ,-1, 200,22, 20,100
gk3,ihr FLroller  "FLroller 3", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,150
gk4,ihr FLroller  "FLroller 4", 80, 5000,1,2 ,1 ,-1, 200,22, 20,200
gk5,ihr FLroller  "FLroller 5", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,250
gk6,ihr FLroller  "FLroller 6", 80, 5000,1,2 ,1 ,-1, 200,22, 20,300
gk7,ihr FLroller  "FLroller 7",50, 5000,1,1 ,2 ,-1, 30,300, 280,50
        FLgroupEnd

        FLgroup "joysticks",380,500, 10,40,3
gk1,gk2,ihj1,ihj2 FLjoy "FLjoy", 50, 18000, 50, 18000,2,2,-1,-1,300,300,30,60
        FLgroupEnd

        FLtabsEnd
        FLscrollEnd
        FLpanelEnd
```

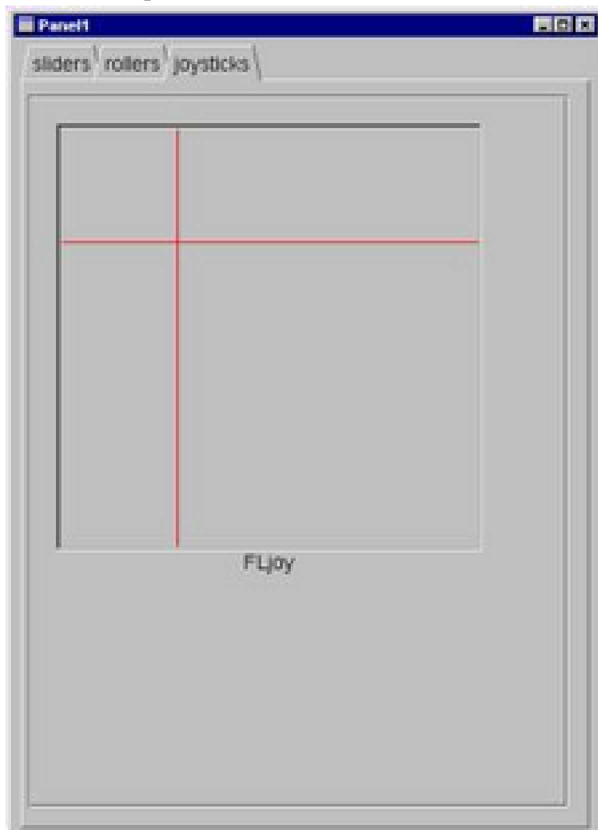
...will produce the following result:



FLtabs example, sliders tab.



FLtabs example, rollers tab.



FLtabs example, joysticks tab.
(Each picture shows a different tab selection inside the same window.)

Examples

Here is an example of the FLtabs opcode. It uses the file *FLtabs.csd* [examples/FLtabs.csd].

Exemple 177. Example of the FLtabs opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLtabs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; A single oscillator with frequency, amplitude and
; panning controls on separate file tab cards
sr = 44100
kr = 441
ksmps = 100
nchnls = 2

FLpanel "Tabs", 300, 350, 100, 100
itabswidth = 280
itabsheight = 330
ix = 5
iy = 5
FLtabs itabswidth,itabsheight, ix,iy

    itab1width = 280
    itab1height = 300
    itab1x = 10
    itab1y = 40
    FLgroup "Tab 1", itab1width, itab1height, itab1x, itab1y
        gkfreq, i1 FLknob "Frequency", 200, 5000, -1, 1, -1, 70, 70, 130
        FLsetVal_i 400, i1
    FLgroupEnd

    itab2width = 280
    itab2height = 300
    itab2x = 10
    itab2y = 40
    FLgroup "Tab 2", itab2width, itab2height, itab2x, itab2y
        gkamp, i2 FLknob "Amplitude", 0, 15000, 0, 1, -1, 70, 70, 130
        FLsetVal_i 15000, i2
    FLgroupEnd

    itab3width = 280
    itab3height = 300
    itab3x = 10
    itab3y = 40
    FLgroup "Tab 3", itab3width, itab3height, itab3x, itab3y
        gkpan, i3 FLknob "Pan position", 0, 1, 0, 1, -1, 70, 70, 130
        FLsetVal_i 0.5, i3
    FLgroupEnd
FLtabsEnd
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
    asig oscili gkamp, gkfreq, ifn
    outs asig*(1-gkpan), asig*gkpan
endin
```

```
</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLtabsEnd

FLtabsEnd — Marks the end of a tabbed FLTK interface.

Description

Marks the end of a tabbed FLTK interface.

Syntax

```
FLtabsEnd
```

Performance

Containers are useful to format the graphic appearance of the widgets. The most important container is *FLpanel*, that actually creates a window. It can be filled with other containers and/or valuators or other kinds of widgets.

There are no k-rate arguments in containers.

See Also

FLgroup, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLtabs_end

FLtabs_end — Marks the end of a tabbed FLTK interface.

Description

Marks the end of a tabbed FLTK interface. This is another name for **FLtabsEnd** provided for compatibility. See *FLtabsEnd*

Credits

Author: Gabriel Maldonado

New in version 4.22

FLtext

FLtext — A FLTK widget opcode that creates a textbox.

Description

FLtext *allows the user to modify a parameter value by directly typing it into a text field.*

Syntax

```
kout, ihandle FLtext "label", imin, imax, istep, itype, iwidth, \
      iheight, ix, iy
```

Initialization

ihandle -- a handle value (an integer number) that unequivocally references a corresponding widget. This is used by other opcodes that modify a widget's properties (see *Modifying FLTK Widget Appearance*). It is automatically output by *FLtext* and must not be set by the user label. (The user label is a double-quoted string containing some user-provided text placed near the widget.)

« label » -- a double-quoted string containing some user-provided text, placed near corresponding widget.

imin -- minimum value of output range.

imax -- maximum value of output range.

istep -- a floating-point number indicating the increment of valuator value corresponding to of each mouse click. The *istep* argument allows the user to arbitrarily slow roller's motion, enabling arbitrary precision.

itype -- an integer number denoting the appearance of the valuator.

The *itype* argument can be set to the following values:

- 1 - normal behaviour
- 2 - dragging operation is suppressed, instead it will appear two arrow buttons. A mouse-click on one of these buttons can increase/decrease the output value.
- 3 - text editing is suppressed, only mouse dragging modifies the output value.

iwidth -- width of widget.

iheight -- height of widget.

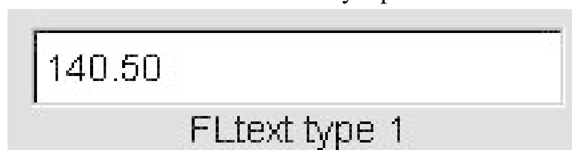
ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

kout -- output value

FLtext allows the user to modify a parameter value by directly typing it into a text field:



FLtext.
Its value can also be modified by clicking on it and dragging the mouse horizontally. The *istep* argument allows the user to arbitrarily set the response on mouse dragging.

Examples

Here is an example of the FLtext opcode. It uses the file *FLtext.csd* [examples/FLtext.csd].

Exemple 178. Example of the FLtext opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLtext.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with fltext box controlled
; frequency either click and drag or double click and
; type to change frequency value
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Text Box", 270, 600, 50, 50
    ; Minimum value output by the text box
    imin = 200
    ; Maximum value output by the text box
    imax = 5000
    ; Step size
    istep = 1
    ; Text box graphic type
    itype = 1
    ; Width of the text box in pixels
    iwidth = 70
    ; Height of the text box in pixels
    iheight = 30
    ; Distance of the left edge of the text box
    ; from the left edge of the panel
    ix = 100
    ; Distance of the top edge of the text box
    ; from the top edge of the panel
    iy = 300

    gkfreq,ihandle FLtext "Enter the frequency", imin, imax, istep, itype, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun
```

```
instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLcount, *FLjoy*, *FLknob*, *FLroller*, *FLslider*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLupdate

FLupdate — Same as the FLrun opcode.

Description

Same as the *FLrun* opcode.

Syntax

```
FLupdate
```

fluidAllOut

fluidAllOut — Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

Syntaxe

```
aleft, aright fluidAllOut
```

Description

Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

Exécution

aleft -- Canal de sortie audio gauche.

aright -- Canal de sortie audio droite.

Appelez fluidAllOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il n'y a aucun moyen d'activer ou d'arrêter de tels effets, ou de changer leurs paramètres, depuis Csound.

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluidAllOut.orc* [examples/fluidAllOut.orc].

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1      fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
             fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
             fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2      fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
             fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT

instr 1 ; FluidSynth Steinway Rev
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel    = 1
  ikey        = p4
  ivelocity   = p5
```

```
  istatus    = 144
             fluidControl gienginenum1, istatus, ichannel, ikey, ivelocity
endin

instr 2 ; GM soundfont
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel   = 2
  ikey       = p4
  ivelocity  = p5
  istatus    = 144
             fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel   = 3
  ikey       = p4
  ivelocity  = p5
  istatus    = 144
             fluidNote gienginenum1, ichannel, ikey, ivelocity
endin


; COLLECT AUDIO FROM ALL SOUNDFONTS

instr 100 ; Fluidsynth output
  ; INITIALIZATION
  ; Normalize so iamplitude for p5 of 80 == ampdb(80).
  iamplitude = ampdb(p5) * (10000.0 / 0.1)
  ; AUDIO
  aleft, aright fluidAllOut
             outs aleft * iamplitude, aright * iamplitude
endin
```

Voici un autre exemple plus complexe des opcodes fluidsynth écrit par Istvan Varga. Il utilise le fichier *fluidcomplex.csd* [examples/fluidcomplex.csd].

```
<CsoundSynthesizer>
<CsOptions>
-d -m229 -o dac -T -F midifile.mid
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
        massign   ichn, 0
        loop_le   ichn, 1, 16, lp1
        pgmassign 0, 0

; initialize FluidSynth

gifld   fluidEngine
gisf2   fluidLoad "07AcousticGuitar.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
  keng, kchn, ksf2, kbnk, kpre xin
        igoto     skipInit
  doInit:
        fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
        reinit    doInit
        rireturn
  skipInit:
```

```
endop

instr 1
  ; initialize channels
  kchn  init 1
  if (kchn == 1) then
lp2:
        fluidControl gifld, 192, kchn - 1, 0, 0
        fluidControl gifld, 176, kchn - 1, 7, 100
        fluidControl gifld, 176, kchn - 1, 10, 64
        loop_le   kchn, 1, 16, lp2
  endif

  ; send any MIDI events received to FluidSynth
nxt:
  kst, kch, kd1, kd2 midiin
  if (kst != 0) then
    if (kst != 192) then
        fluidControl gifld, kst, kch - 1, kd1, kd2
    else
        fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
    endif
      kgoto nxt
  endif

  ; get audio output from FluidSynth
  aL, aR fluidOut gifld
        outs      aL, aR
endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*

Crédits

Opcode par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.

fluidCCi

fluidCCi — Envoie un message de données de contrôleur MIDI à fluid.

Syntaxe

```
fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue
```

Description

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

Initialisation

iEngineNumber -- numéro du moteur affecté par fluidEngine

iChannelNumber -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

iControllerNumber -- numéro du contrôleur MIDI à utiliser pour ce message

iValue -- valeur à affecter au contrôleur (habituellement 0-127)

Exécution

Cet opcode est utilisé pour affecter des valeurs de contrôleur pendant l'initialisation. Pour des changements continus, utilisez fluidCCk.

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*, *fluidCCk*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

fluidCCk

fluidCCk — Envoie un message de données de contrôleur MIDI à fluid.

Syntaxe

```
fluidCCk iEngineNumber, iChannelNumber, iControllerNumber, kValue
```

Description

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

Initialisation

iEngineNumber -- numéro du moteur affecté par fluidEngine

iChannelNumber -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

iControllerNumber -- numéro du contrôleur MIDI à utiliser pour ce message

Exécution

kValue -- valeur à affecter au contrôleur (habituellement 0-127)

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*, *fluidCCi*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

fluidControl

fluidControl — Envoie un note on, un note off, et d'autres messages MIDI à un preset SoundFont.

Syntaxe

```
fluidControl ienginenum, kstatus, kchannel, kdata1, kdata2
```

Description

Les opcodes fluid fournissent une intégration simple dans des opcodes de Csound du synthétiseur Fluidsynth SoundFont2 de Peter Hanappe. Cette implémentation accepte les messages MIDI de note on, note off, de contrôleur, de pitch bend ou de changement de programme au taux-k. La polyphonie maximale est de 4096 voix simultanées. N'importe quel nombre de SoundFonts peuvent être chargés et joués simultanément.

Initialisation

ienginenum -- numéro du moteur affecté par fluidEngine

Exécution

kstatus -- octet d'état du message de canal MIDI : 128 pour note off, 144 pour note on, 176 pour control change, 192 for program change, ou 224 pour pitch bend.

kchannel -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

kdata1 -- Pour note on, numéro de touche MIDI : de 0 (le plus bas) à 127 (le plus haut), où 60 est le do médian. Pour les messages de contrôleur continu, le numéro du contrôleur.

kdata2 -- Pour note on, la vélocité de touche MIDI : de 0 (pas de son) à 127 (le plus fort). Pour les messages de contrôleur continu, la valeur du contrôleur.

Appelez fluidControl dans les définitions d'instrument qui jouent réellement des notes et qui envoient des messages de contrôle. Chaque définition d'instrument doit utiliser de manière cohérente un canal MIDI qui a été affecté à un programme Fluidsynth au moyen de fluidLoad.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il n'y a aucun moyen d'activer ou d'arrêter de tels effets, ou de changer leurs paramètres, depuis Csound.

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*

Crédits

Opcodes par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.

Nouveau dans Csound5.00

fluidEngine

fluidEngine — Crée une instance de moteur fluidsynth.

Syntaxe

```
ienginenum fluidEngine [iReverbEnabled] [, iChorusEnabled] [,iNumChannels] [, iPolyphony]
```

Description

Crée une instance de moteur fluidsynth, et retourne *ienginenum* pour identifier le moteur. *ienginenum* est passé à d'autres opcodes pour charger et jouer des SoundFonts et pour assembler le son généré.

Initialisation

ienginenum -- numéro du moteur affecté par fluidEngine

iReverbEnabled -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de réverbération dans les SoundFonts chargés.

iChorusEnabled -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de chorus dans les SoundFonts chargés.

iNumChannels -- nombre de canaux à utiliser ; de 16 à 256, la valeur par défaut de Csound est 256 (la valeur par défaut de Fluidsynth est 16).

iPolyphony -- nombre de voix à jouer en parallèle ; de 16 à 4096, la valeur par défaut de Csound est 4096 (la valeur par défaut de Fluidsynth est 256). Note : ce n'est pas le nombre de notes jouées simultanément car une seule note peut utiliser plusieurs voix en fonction des zones d'instrument et de la vélocité et/ou du numéro de touche de la note jouée.

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluidAllOut.orc* [examples/fluidAllOut.orc].

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1      fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
             fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
             fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2      fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
             fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT

instr 1 ; FluidSynth Steinway Rev
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
```

```
  ; Use channel assigned in fluidload.
  ichannel    = 1
  ikey        = p4
  ivelocity   = p5
  istatus     = 144
              fluidControl gienginenum1, istatus, ichannel, ikey, ivelocity
endin

instr 2 ; GM soundfont
  ; INITIALIZATION
              mididefault   60, p3 ; Default duration of 60 -- overridden by score.
              midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel    = 2
  ikey        = p4
  ivelocity   = p5
  istatus     = 144
              fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
  ; INITIALIZATION
              mididefault   60, p3 ; Default duration of 60 -- overridden by score.
              midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel    = 3
  ikey        = p4
  ivelocity   = p5
  istatus     = 144
              fluidNote gienginenum1, ichannel, ikey, ivelocity
endin


; COLLECT AUDIO FROM ALL SOUNDFONTS

instr 100 ; Fluidsynth output
  ; INITIALIZATION
  ; Normalize so iamplitude for p5 of 80 == ampdb(80).
  iamplitude = ampdb(p5) * (10000.0 / 0.1)
  ; AUDIO
  aleft, aright fluidAllOut
              outs aleft * iamplitude, aright * iamplitude
endin
```

Voici un exemple des opcodes fluidsynth qui fait appel à 2 moteurs. Il utilise le fichier *fluid-2.orc* [examples/fluid-2.orc].

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1      fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
             fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
             fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2      fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
             fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT

instr 1 ; FluidSynth Steinway Rev
  ; INITIALIZATION
              mididefault   60, p3 ; Default duration of 60 -- overridden by score.
              midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel    = 1
  ikey        = p4
  ivelocity   = p5
              fluidNote gienginenum1, ichannel, ikey, ivelocity
endin
```

```
instr 2 ; GM soundfont
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel   = 2
  ikey       = p4
  ivelocity  = p5
             fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
  ; INITIALIZATION
             mididefault   60, p3 ; Default duration of 60 -- overridden by score.
             midinoteonkey p4, p5 ; Channels MIDI input to pfields.
  ; Use channel assigned in fluidload.
  ichannel   = 3
  ikey       = p4
  ivelocity  = p5
             fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

; COLLECT AUDIO FROM ALL SOUNDFONTS

instr 100 ; Fluidsynth output
  ; INITIALIZATION
  ; Normalize so iamplitude for p5 of 80 == ampdb(80).
  iamplitude1 = ampdb(p5) * (10000.0 / 0.1)
  iamplitude2 = ampdb(p6) * (10000.0 / 0.1)

; AUDIO
aleft1, aright1 fluidOut   gienginenum1
aleft2, aright2 fluidOut   gienginenum2
                outs       (aleft1 * iamplitude1) + (aleft2 * iamplitude2),  \
                           (aright1 * iamplitude1) + (aright2 * iamplitude2)
endin
```

Voici un autre exemple plus complexe des opcodes fluidsynth écrit par Istvan Varga. Il utilise le fichier *fluidcomplex.csd* [examples/fluidcomplex.csd].

```
<CsoundSynthesizer>
<CsOptions>
-d -m229 -o dac -T -F midifile.mid
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
        massign   ichn, 0
        loop_le   ichn, 1, 16, lp1
        pgmassign 0, 0

; initialize FluidSynth

gifld   fluidEngine
gisf2   fluidLoad "07AcousticGuitar.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
  keng, kchn, ksf2, kbnk, kpre xin
        igoto     skipInit
  doInit:
        fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
        reinit    doInit
        rireturn
  skipInit:
endop

instr 1
```

```
  ; initialize channels
  kchn  init 1
  if (kchn == 1) then
lp2:
        fluidControl gifld, 192, kchn - 1, 0, 0
        fluidControl gifld, 176, kchn - 1, 7, 100
        fluidControl gifld, 176, kchn - 1, 10, 64
        loop_le   kchn, 1, 16, lp2
  endif

  ; send any MIDI events received to FluidSynth
nxt:
  kst, kch, kd1, kd2 midiin
  if (kst != 0) then
    if (kst != 192) then
        fluidControl gifld, kst, kch - 1, kd1, kd2
    else
        fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
    endif
      kgoto nxt
  endif

  ; get audio output from FluidSynth
  aL, aR fluidOut gifld
        outs      aL, aR
endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fluidNote, *fluidLoad*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Les paramètres facultatifs *iNumChannels* et *iPolyphony* ont été ajoutés dans la version 5.07

fluidLoad

fluidLoad — Charge un SoundFont dans un fluidEngine, en listant éventuellement le contenu du SoundFont.

Syntaxe

```
isfnum fluidLoad soundfont, ienginenum[, ilistpresets]
```

Description

Charge un SoundFont dans une instance d'un fluidEngine, en listant éventuellement les banques et les presets du SoundFont.

Initialisation

isfnum -- numéro affecté au soundfont qui vient d'être chargé.

soundfont -- chaîne spécifiant le nom de fichier d'un SoundFont. Notez que n'importe quel nombre de SoundFonts peuvent être chargés (évidemment, par différents appels de fluidLoad).

ienginenum -- numéro du moteur affecté par fluidEngine

ilistpresets -- facultatif, s'il est spécifié, tous les programmes Fluidsynth du SoundFont qui vient d'être chargé sont listés. Un programme FluidSynth est une combinaison d'ID de SoundFont, de numéro de banque, et de numéro de preset qui est affecté à un canal MIDI.

Exécution

Appelez fluidLoad dans l'en-tête de l'orchestre, autant de fois que vous voulez. Le même SoundFont peut être appelé pour affecter des programmes à des canaux MIDI autant de fois que l'on veut ; le SoundFont n'est chargé que la première fois.

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine  fluidEngine
isfnum    fluidLoad "07AcousticGuitar.sf2", giengine, 1
          fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
          mididefault     60, p3
          midinoteonkey   p4, p5

  ikey    init            p4
  ivel    init            p5

          fluidNote       giengine, 1, ikey, ivel
endin

instr 99
  imvol   init            70000
```

```
  asigl, asigr fluidOut   giengine
         outs             asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

Voir Aussi

fluidEngine, *fluidNote*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

fluidNote

fluidNote — Joue une note sur un canal dans un moteur fluidsynth.

Syntaxe

```
fluidNote ienginenum, ichannelnum, imidikey, imidivel
```

Description

Joue une note de hauteur *imidikey* et de vélocité *imidivel* sur le canal *ichannelnum* du fluidEngine numéro *ienginenum*.

Initialisation

ienginenum -- numéro du moteur affecté par fluidEngine

ichannelnum -- numéro de canal sur lequel jouer la note dans le fluidEngine donné

imidikey -- touche MIDI de la note (0-127)

imidivel -- vélocité MIDI de la note (0-127)

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine  fluidEngine
isfnum    fluidLoad "07AcousticGuitar.sf2", giengine, 1
          fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
          mididefault     60, p3
          midinoteonkey   p4, p5

  ikey    init            p4
  ivel    init            p5

          fluidNote       giengine, 1, ikey, ivel
endin

instr 99
  imvol   init            70000
  asigl, asigr fluidOut   giengine
          outs            asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

Voir Aussi

fluidEngine, *fluidLoad*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

fluidOut

fluidOut — Envoie en sortie le son d'un fluidEngine donné.

Syntaxe

```
aleft, aright fluidOut ienginenum
```

Description

Envoie en sortie le son d'un fluidEngine donné.

Initialisation

ienginenum -- numéro du moteur affecté par fluidEngine

Exécution

aleft -- Canal de sortie audio gauche.

aright -- Canal de sortie audio droite.

Appelez fluidOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts utilisés par le fluidEngine numéro *ienginenum* envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine  fluidEngine
isfnum    fluidLoad "07AcousticGuitar.sf2", giengine, 1
          fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
          mididefault     60, p3
          midinoteonkey   p4, p5

  ikey    init            p4
  ivel    init            p5

          fluidNote       giengine, 1, ikey, ivel
endin

instr 99
  imvol   init            70000
  asigl, asigr fluidOut   giengine
          outs            asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

fluidProgramSelect

fluidProgramSelect — Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

Syntaxe

```
fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum
```

Description

Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

Initialisation

ienginenum -- numéro du moteur affecté par fluidEngine

ichannelnum -- numéro du canal auquel affecter le preset dans le fluidEngine donné

isfnum -- numéro du SoundFont duquel le preset est issu

ibanknum -- numéro de la banque dans le SoundFont de laquelle le preset est issu

ipresetnum -- numéro du preset à affecter

Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine  fluidEngine
isfnum    fluidLoad "07AcousticGuitar.sf2", giengine, 1
          fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
          mididefault      60, p3
          midinoteonkey    p4, p5

  ikey    init             p4
  ivel    init             p5

          fluidNote        giengine, 1, ikey, ivel
endin

instr 99
  imvol   init             70000
  asigl, asigr fluidOut    giengine
          outs             asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

Voir Aussi

fluidEngine, *fluidNote*, *fluidLoad*

Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

fluidSetInterpMethod

fluidSetInterpMethod — Set interpolation method for channel in Fluid Engine

Syntax

```
fluidSetInterpMethod ienginenum, ichannelnum, iInterpMethod
```

Description

Set interpolation method for channel in Fluid Engine. Lower order interpolation methods will render faster at lower fidelity while higher order interpolation methods will render slower at higher fidelity. Default interpolation for a channel is 4th order interpolation.

Initialization

ienginenum -- engine number assigned from fluidEngine

ichannelnum -- which channel number to use for the preset in the given fluidEngine

iInterpMethod -- interpolation method, can be any of the following

- *0* -- No Interpolation
- *1* -- Linear Interpolation
- *4* -- 4th Order Interpolation (Default)
- *7* -- 7th Order Interpolation (Highest)

See Also

fluidEngine

Credits

Author: Steven Yi

New in version 5.07

FLvalue

FLvalue — Shows the current value of a FLTK valuator.

Description

FLvalue shows current the value of a valuator in a text field.

Syntax

```
ihandle FLvalue "label", iwidth, iheight, ix, iy
```

Initialization

ihandle -- handle value (an integer number) that unequivocally references the corresponding valuator. It can be used for the *idisp* argument of a valuator.

« label » -- a double-quoted string containing some user-provided text, placed near the corresponding widget.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the valuator, relative to the upper left corner of corresponding window (expressed in pixels).

Performance

FLvalue shows the current values of a valuator in a text field. It outputs *ihandle* that can then be used for the *idisp* argument of a valuator (see the *FLTK Valuators section*). In this way, the values of that valuator will be dynamically be shown in a text field.

Note

Note that *FLvalue* is not a valuator and its value cannot be modified.The value for an *FLvalue* widget should be set only by other widgets, and NOT from *FLsetVal* or *FLsetVal_i* since this can cause Csound to crash.

Examples

Here is an example of the FLvalue opcode. It uses the file *FLvalue.csd* [examples/FLvalue.csd].

Exemple 179. Example of the FLvalue opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command

line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLvalue.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flvalue to display the output of a slider
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Value Display Box", 900, 200, 50, 50
    ; Width of the value display box in pixels
    iwidth = 50
    ; Height of the value display box in pixels
    iheight = 20
    ; Distance of the left edge of the value display
    ; box from the left edge of the panel
    ix = 65
    ; Distance of the top edge of the value display
    ; box from the top edge of the panel
    iy = 55

    idisp FLvalue "Hertz", iwidth, iheight, ix, iy
    gkfreq, ihandle FLslider "Frequency", 200, 5000, -1, 5, idisp, 750, 30, 125, 50
    FLsetVal_i 500, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin


</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLbox, *FLbutBank*, *FLbutton*, *FLprintk*, *FLprintk2*

Credits

Author: Gabriel Maldonado

New in version 4.22

Example written by Iain McCurdy, edited by Kevin Conder.

FLvkeybd

FLvkeybd — An FLTK widget opcode that creates a virtual keyboard widget.

Description

An FLTK widget opcode that creates a virtual keyboard widget. This must be used in conjunction with the virtual midi keyboard driver for this to operate correctly. The purpose of this opcode is for making demo versions of MIDI orchestras with the virtual keyboard embedded within the main window.

Note

The widget version of the virtual keyboard does not include the MIDI sliders found in the full window version of the virtual keyboard.

Syntax

```
FLvkeybd "keyboard.map", iwidth, iheight, ix, iy
```

Initialization

« keyboard.map » -- a double-quoted string containing the keyboard map to use. An empty string ("") may be used to use the default bank/channel name values. See Virtual Midi Keyboard for more information on keyboard mappings.

iwidth -- width of widget.

iheight -- height of widget.

ix -- horizontal position of upper left corner of the keyboard, relative to the upper left corner of corresponding window (expressed in pixels).

iy -- vertical position of upper left corner of the keyboard, relative to the upper left corner of corresponding window (expressed in pixels).

Note

The standard width and height for the virtual keyboard is 624x120 for the dialog version that is shown when FLvkeybd is not used.

See Also

FLbutton, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

Credits

Author: Steven Yi

New in version 5.05

FLvslidBnk

FLvslidBnk — A FLTK widget containing a bank of horizontal sliders.

Description

FLvslidBnk is a widget containing a bank of horizontal sliders.

Syntax

```
FLvslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
      [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

Initialization

« names » -- a double-quoted string containing the names of each slider. Each slider can have a different name. Separate each name with « @ » character, for example: « frequency@amplitude@cutoff ». It is possible to not provide any name by giving a single space «   ». In this case, the opcode will automatically assign a progressive number as a label for each slider.

inumsliders -- the number of sliders.

ioutable (optional, default=0) -- number of a previously-allocated table in which to store output values of each slider. The user must be sure that table size is large enough to contain all output cells, otherwise a segfault will crash Csound. By assigning zero to this argument, the output will be directed to the zak space in the k-rate zone. In this case, the zak space must be previously allocated with the *zakinit* opcode and the user must be sure that the allocation size is big enough to cover all sliders. The default value is zero (i.e. store output in zak space).

istart_index (optional, default=0) -- an integer number referring to a starting offset of output cell locations. It can be positive to allow multiple banks of sliders to output in the same table or in the zak space. The default value is zero (no offset).

iminmaxtable (optional, default=0) -- number of a previously-defined table containing a list of min-max pairs, referred to each slider. A zero value defaults to the 0 to 1 range for all sliders without necessity to provide a table. The default value is zero.

iexptable (optional, default=0) -- number of a previously-defined table containing a list of identifiers (i.e. integer numbers) provided to modify the behaviour of each slider independently. Identifiers can assume the following values:

- -1 -- exponential curve response
- 0 -- linear response
- number > than 0 -- follow the curve of a previously-defined table to shape the response of the corresponding slider. In this case, the number corresponds to table number.

You can assume that all sliders of the bank have the same response curve (exponential or linear). In this case, you can assign -1 or 0 to *iexptable* without worrying about previously defining any table. The default value is zero (all sliders have a linear response, without having to provide a table).

itypetable (optional, default=0) -- number of a previously-defined table containing a list of identifiers

(i.e. integer numbers) provided to modify the aspect of each individual slider independently. Identifiers can assume the following values:

- 0 = Nice slider
- 1 = Fill slider
- 3 = Normal slider
- 5 = Nice slider
- 7 = Nice slider with down-box

You can assume that all sliders of the bank have the same aspect. In this case, you can assign a negative number to *itypetable* without worrying about previously defining any table. Negative numbers have the same meaning of the corresponding positive identifiers with the difference that the same aspect is assigned to all sliders. You can also assign a random aspect to each slider by setting *itypetable* to a negative number lower than -7. The default value is zero (all sliders have the aspect of nice sliders, without having to provide a table).

You can add 20 to a value inside the table to make the slider "plastic", or subtract 20 if you want to set the value for all widgets without defining a table (e.g. -21 to set all sliders types to Plastic Fill slider).

iwidth (optional) -- width of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

iheight (optional) -- height of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

ix (optional) -- horizontal position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

iy (optional) -- vertical position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

Performance

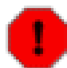
There are no k-rate arguments, even if cells of the output table (or the zak space) are updated at k-rate.

FLvslidBnk is a widget containing a bank of vertical sliders. Any number of sliders can be placed into the bank (*inumsliders* argument). The output of all sliders is stored into a previously allocated table or into the zak space (*ioutable* argument). It is possible to determine the first location of the table (or of the zak space) in which to store the output of the first slider by means of *istart_index* argument.

Each slider can have an individual label that is placed below it. Labels are defined by the *« names »* argument. The output range of each slider can be individually set by means of an external table (*iminmaxtable* argument). The curve response of each slider can be set individually, by means of a list of identifiers placed in a table (*iexptable* argument). It is possible to define the aspect of each slider independently or to make all sliders have the same aspect (*itypetable* argument).

The *iwidth*, *iheight*, *ix*, and *iy* arguments determine width, height, horizontal and vertical position of the rectangular area containing sliders. Notice that the label of each slider is placed below them and is not included in the rectangular area containing sliders. So the user should leave enough space below the bank by assigning a proper *ix* value in order to leave labels visible.

FLvslidBnk is identical to *FLslidBnk* except it contains vertical sliders instead of horizontal. Since the width of each single slider is often small, it is recommended to leave only a single space in the names string (" "), in this case each slider will be automatically numbered.



IMPORTANT!

Notice that the tables used by *FLvslidBnk* must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. This is because tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

Examples

Here is an example of the FLvslidBnk opcode. It uses the file *FLvslidBnk.csd* [examples/FLvslidBnk.csd].

Exemple 180. Example of the FLvslidBnk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gitypetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 400, 50, 50
    ;Number of sliders
    inum = 8
    ; Table to store output
    iouttable = giouttable
    ; Width of the slider bank in pixels
    iwidth = 350
    ; Height of the slider in pixels
    iheight = 160
    ; Distance of the left edge of the slider
    ; from the left edge of the panel
    ix = 30
    ; Distance of the top edge of the slider
    ; from the top edge of the panel
    iy = 10
    ; Table containing fader types
    itypetable  = gitypetable
    FLvslidBnk "1@2@3@4@5@6@7@8@9@10@11@12@13@14@15@16", 16 , iouttable , iwidth , iheight , ix \
      , iy , itypetable
    FLvslidBnk " ", inum , iouttable , iwidth , iheight , ix \
      , iy + 200 , -23
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun


instr 1
;Dummy instrument
```

```
endin


</CsInstruments>
<CsScore>

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e


</CsScore>
</CsoundSynthesizer>
```

See Also

FLslider, *FLslidBnk*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLvslidBnk2

FLvslidBnk2 — A FLTK widget containing a bank of horizontal sliders.

Description

FLvslidBnk2 is a widget containing a bank of horizontal sliders.

Syntax

```
FLvslidBnk2 "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart_index]
```

Initialization

« names » -- a double-quoted string containing the names of each slider. Each slider can have a different name. Separate each name with « @ » character, for example: « frequency@amplitude@cutoff ». It is possible to not provide any name by giving a single space «   ». In this case, the opcode will automatically assign a progressive number as a label for each slider.

inumsliders -- the number of sliders.

ioutable (optional, default=0) -- number of a previously-allocated table in which to store output values of each slider. The user must be sure that table size is large enough to contain all output cells, otherwise a segfault will crash Csound. By assigning zero to this argument, the output will be directed to the zak space in the k-rate zone. In this case, the zak space must be previously allocated with the *zakinit* opcode and the user must be sure that the allocation size is big enough to cover all sliders. The default value is zero (i.e. store output in zak space).

iconfigtable -- in the *FLslidBnk2* and FLvslidBnk2 opcodes, this table replaces *iminmaxtable*, *iexptable* and *istyletable*, all these parameters being placed into a single table. This table has to be filled with a group of 5 parameters for each slider in this way:

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

for example using GEN02 you can type:

inum *ftgen* 1,0,256, -2,   0,1,0,1,   100, 5000, -1, 3,   50, 200, -1, 5,….. [etcetera]

In this example the first slider will be affected by the [0,1,0,1] parameters (the range will be 0 to 1, it will have linear response, and its aspect will be a fill slider), the second slider will be affected by the [100,5000,-1,3] parameters (the range is 100 to 5000, the response is exponential and the aspect is a normal slider), the third slider will be affected by the [50,200,-1,5] parameters (the range is 50 to 200, the behavior exponential, and the aspect is a nice slider), and so on.

iwidth (optional) -- width of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

iheight (optional) -- height of the rectangular area containing all sliders of the bank, excluding text labels, that are placed to the left of that area.

ix (optional) -- horizontal position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make

sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

iy (optional) -- vertical position of the upper left corner of the rectangular area containing all sliders belonging to the bank. You have to leave enough space, at the left of that rectangle, in order to make sure labels of sliders to be visible. This is because the labels themselves are external to the rectangular area.

istart_index (optional, default=0) -- an integer number referring to a starting offset of output cell locations. It can be positive to allow multiple banks of sliders to output in the same table or in the zak space. The default value is zero (no offset).

Performance

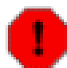
There are no k-rate arguments, even if cells of the output table (or the zak space) are updated at k-rate.

FLvslidBnk2 is a widget containing a bank of vertical sliders. Any number of sliders can be placed into the bank (*inumsliders* argument). The output of all sliders is stored into a previously allocated table or into the zak space (*ioutable* argument). It is possible to determine the first location of the table (or of the zak space) in which to store the output of the first slider by means of *istart_index* argument.

Each slider can have an individual label that is placed to the left of it. Labels are defined by the *« names »* argument. The output range of each slider can be individually set by means of the *min* and *max* values inside the *iconfigtable* table. The curve response of each slider can be set individually, by means of a list of identifiers placed in the *iconfigtable* table (*exp* argument). It is possible to define the aspect of each slider independently or to make all sliders have the same aspect (*style* argument in the *iconfigtable* table).

The *iwidth*, *iheight*, *ix*, and *iy* arguments determine width, height, horizontal and vertical position of the rectangular area containing sliders. Notice that the label of each slider is placed below them and is not included in the rectangular area containing sliders. So the user should leave enough space below the bank by assigning a proper *ix* value in order to leave labels visible.

FLvslidBnk2 is identical to *FLslidBnk2* except it contains vertical sliders instead of horizontal. Since the width of each single slider is often small, it is recommended to leave only a single space in the names string (“ “), in this case each slider will be automatically numbered.



IMPORTANT!

Notice that the tables used by *FLvslidBnk2* must be created with the *ftgen* opcode and placed in the orchestra before the corresponding valuator. They can not placed in the score. This is because tables placed in the score are created later than the initialization of the opcodes placed in the header section of the orchestra.

See Also

FLslider, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk2*

Credits

Author: Gabriel Maldonado

New in version 5.06

FLxyin

FLxyin — Senses the mouse cursor position in a user-defined area inside an FLpanel.

Description

Similar to *xyin*, sense the mouse cursor position in a user-defined area inside an FLpanel.

Syntax

```
koutx, kouty, kinside  FLxyin ioutx_min, ioutx_max, iouty_min, iouty_max, \
      iwindx_min, iwindx_max, iwindy_min, iwindy_max [, iexpx, iexpy, ioutx, iouty]
```

Initialization

ioutx_min, ioutx_max - the minimum and maximum limits of the interval to be output (X or horizontal axis).

iouty_min, iouty_max - the minimum and maximum limits of the interval to be output (Y or vertical axis).

iwindx_min, iwindx_max - the X coordinate of the horizontal edges of the sensible area, relative to the FLpanel ,in pixels.

iwindy_min, iwindy_max - the Y coordinates of the vertical edges of the sensible area, relative to the FL-panel, in pixels.

iexpx, iexpy - (optional) integer numbers denoting the behavior of the x or y output: 0 -> output is linear; -1 -> output is exponential; any other number indicates the number of an existing table that is used for indexing. With a positive value for table number, linear interpolation is provided in table indexing. A negative table number suppresses interpolation. Notice that in normal operations, the table should be normalized and unipolar (i.e. all table elements should be in the zero to one range). In this case all table elements will be rescaled according to imax and imin. Anyway, it is possible to use non-normalized tables (created with a negative table number, that can contain elements of any value), in order to access the actual values of table elements, without rescaling, by assigning 0 to iout_min and 1 to iout_max.

ioutx, iouty – (optional) initial output values.

Performance

koutx, kouty - output values, scaled according to user choices.

kinside - a flag that informs if the mouse cursor falls out of the rectangle of the user-defined area. If it is out of the area, kinside is set to zero.

FLxyin senses the mouse cursor position in a user-defined area inside an *FLpanel*. When *FLxyin* is called, the position of the mouse within the chosen area is returned at k-rate. It is possible to define the sensible area, as well the minimum and maximum values corresponding to the minimum and maximum mouse positions. Mouse buttons don’t need to be pressed to make *FLxyin* to operate. It is able to function correctly even if other widgets (present in the *FLpanel*) overlap the sensible area.

FLxyin unlike most other FLTK opcodes can't be used inside the header, since it is not a widget. It is just a definition of an area for mouse sensing within an FLTK panel.

Examples

Here is an example of the FLxyin opcode. It uses the file *FLxyin.csd* [examples/FLxyin.csd].

Exemple 181. Example of the FLxyin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Andres Cabrera 2007

FLpanel "FLxyin", 200, 100, -1, -1, 3
FLpanelEnd
FLrun

instr 1
  koutx, kouty, kinside FLxyin 0, 10, 100, 1000, 10, 190, 10, 90
  aout buzz 10000, kouty, koutx, 1
  printk2 koutx
  outs aout, aout
endin


</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 3600

e

</CsScore>
</CsoundSynthesizer>
```

Here is another example of the FLxyin opcode. It uses the file *FLxyin-2.csd* [examples/FLxyin-2.csd].

Exemple 182. Example of the FLxyin opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
kr=441
ksmps=100
nchnls=2

; Example by Gabriel Maldonado

        FLpanel "Move the mouse inside this panel to hear the effect",400,400
        FLpanel_end
        FLrun
```

```
        instr 1

k1, k2, kinside FLxyin   50, 1000, 50, 1000, 100, 300, 50, 250, -2,-3
;if k1 <= 50 || k1 >=5000 || k2 <=100 || k2 >= 8000 kgoto end ; if cursor is outside bounds, then don't t

a1 oscili 3000, k1, 1
a2 oscili 3000, k2, 1

        outs a1,a2
printk2 k1
printk2 k2, 10
printk2 kinside, 20
end:
        endin

</CsInstruments>
<CsScore>

f1 0 1024 10 1
f2 0 17 19 1 1 90 1
f3 0 17 19 2 1 90 1
i1 0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

FLpanel

Credits

Author: Gabriel Maldonado

New in version 5.06

fmb3

fmb3 — Utilise la synthèse FM pour créer un son d'orgue Hammond B3.

Description

Utilise la synthèse FM pour créer un son d'orgue Hammond B3. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn
```

Initialisation

fmb3 prend 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmb3. Il utilise le fichier *fmb3.csd* [examples/fmb3.csd].

Exemple 183. Exemple de l'opcode fmb3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmb3.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 15000
  kfreq = 440
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  a1 fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate,  \
          ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fmbell, *fmmetal*, *fmpercfl*, *fmrhode*, *fmwurlie*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.

Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fmbell

fmbell — Utilise la synthèse FM pour créer un son de cloche tube.

Description

Utilise la synthèse FM pour créer un son de cloche tube. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn
```

Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmbell. Il utilise le fichier *fmbell.csd* [examples/fmbell.csd].

Exemple 184. Exemple de l'opcode fmbell.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmbell.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10000
  kfreq = 880
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  a1 fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fmb3, *fmmetal*, *fmpercfl*, *fmrhode*, *fmwurlie*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fmmetal

fmmetal — Utilise la synthèse FM pour créer un son de « Heavy Metal ».

Description

Utilise la synthèse FM pour créer un son de « Heavy Metal ». Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn
```

Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn3* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn4* -- onde sinus

Note

Le fichier « twopeaks.aiff » est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 3

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmmetal. Il utilise les fichiers *fmmetal.csd* [examples/fmmetal.csd] et *two-peaks.aiff* [examples/twopeaks.aiff].

Exemple 185. Exemple de l'opcode fmmetal.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmmetal.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10000
  kfreq = 440
  kc1 = 6
  kc2 = 5
  kvdepth = 0
  kvrate = 0
  ifn1 = 1
  ifn2 = 2
  ifn3 = 2
  ifn4 = 1
  ivfn = 1

  a1 fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a normal sine wave.
f 1 0 32768 10 1
; Table #2, the "twopeaks.aiff" audio file.
f 2 0 256 1 "twopeaks.aiff" 0 0 0

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

voir Aussi

fmb3, *fmbell*, *fmpercfl*, *fmrhode*, *fmwurlie*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fmpercfl

fmpercfl — Utilise la synthèse FM pour créer un son de flûte percussive.

Description

Utilise la synthèse FM pour créer un son de flûte percussive. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
      ifn3, ifn4, ivfn
```

Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmpercfl. Il utilise le fichier *fmpercfl.csd* [examples/fmpercfl.csd].

Exemple 186. Exemple de l'opcode fmpercfl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmpercfl.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  a1 fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fmb3, *fmbell*, *fmmetal*, *fmrhode*, *fmwurlie*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fmrhode

fmrhode — Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes.

Description

Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
      ifn3, ifn4, ivfn
```

Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]

Note

Le fichier « fwavblnk.aiff » est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmrhode. Il utilise les fichiers *fmrhode.csd* [examples/fmrhode.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

Exemple 187. Exemple de l'opcode fmrhode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmrhode.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kc1 = 6
  kc2 = 0
  kvdepth = 0.01
  kvrate = 3
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 2
  ivfn = 1

  a1 fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1
; Table #2, the "fwavblnk.aiff" audio file.
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fmb3, *fmbell*, *fmmetal*, *fmpercfl*, *fmwurlie*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fmvoice

fmvoice — Synthèse FM d'une Voix de Chanteur

Description

Synthèse FM d'une Voix de Chanteur

Syntaxe

```
ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, \
      ifn2, ifn3, ifn4, ivibfn
```

Initialisation

ifn1, ifn2, ifn3,ifn3 -- Tables, normalement des formes d'onde sinus.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kvowel -- La voyelle chantée, dans l'intervalle 0-64

ktilt -- La pente spectrale du son dans l'intervalle 0 à 99

kvibamt -- Largeur du vibrato

kvibrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmvoice. Il utilise le fichier *fmvoice.csd* [examples/fmvoice.csd].

Exemple 188. Exemple de l'opcode fmvoice.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmvoice.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 110
  ; Use the fourth p-field for the vowel.
  kvowel = p4
  ktilt = 0
  kvibamt = 0.005
  kvibrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivibfn = 1

  a1 fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, ifn2, ifn3, ifn4, ivibfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = vowel (a value from 0 to 64)
; Play Instrument #1 for one second, vowel=1.
i 1 0 1 1
; Play Instrument #1 for one second, vowel=2.
i 1 1 1 2
; Play Instrument #1 for one second, vowel=3.
i 1 2 1 3
; Play Instrument #1 for one second, vowel=4.
i 1 3 1 4
; Play Instrument #1 for one second, vowel=5.
i 1 4 1 5
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 3.47 de Csound

fmwurlie

fmwurlie — Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer.

Description

Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

Syntaxe

```
ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn
```

Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]

Note

Le fichier « fwavblnk.aiff » est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note jouée.

kc1, kc2 -- Contrôles pour le synthériseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

kvdepth -- Largeur du vibrato

kvrate -- Vitesse du vibrato

Exemples

Voici un exemple de l'opcode fmwurlie. Il utilise les fichiers *fmwurlie.csd* [examples/fmwurlie.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

Exemple 189. Exemple de l'opcode fmwurlie.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmwurlie.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 440
  kc1 = 6
  kc2 = 1
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 2
  ivfn = 1

  a1 fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1
; Table #2, the "fwavblnk.aiff" audio file.
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fmb3, *fmbell*, *fmmetal*, *fmpercfl*, *fmrhode*

Crédits

Auteur : John ffitch (d'après Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

fof

fof — Produit des grains FOF (sinusoïde amortie) pour la synthèse par formant et la synthèse granulaire.

Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-k. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

Syntaxe

```
ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]
```

Initialisation

iolaps -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund * kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

ifna, ifnb -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

itotdur -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

iphs (facultatif, par défaut 0) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

ifmode (facultatif, par défaut 0) -- mode fréquentiel du formant. S'il est nul, chaque grain FOF garde la fréquence *xform* avec laquelle il a été amorcé. Sinon, chaque grain FOF est influencé par *xform* en continu. La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

Exécution

xamp -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons Q < 10) et/ou quand les grains FOF se superposent en partie.

xfund -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

xform -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu (voir *ifmode*).

koct -- indice d'octaviation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

kband -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

kris, kdur, kdec -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur, xfund*, et de la densité des chevauchements.

Exemples

Voici un exemple de l'opcode fof. Il utilise le fichier *fof.csd* [examples/fof.csd].

Exemple 190. Exemple de l'opcode fof.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Combine five formants together to create
  ; an alto-"a" sound.

  ; Values common to all of the formants.
  kfund init 261.659
  koct init 0
  kris init 0.003
  kdur init 0.02
  kdec init 0.007
  iolaps = 14850
  ifna = 1
  ifnb = 2
  itotdur = p3

  ; First formant.
  k1amp = ampdb(0)
```

```
  k1form init 800
  k1band init 80

  ; Second formant.
  k2amp = ampdb(-4)
  k2form init 1150
  k2band init 90

  ; Third formant.
  k3amp = ampdb(-20)
  k3form init 2800
  k3band init 120

  ; Fourth formant.
  k4amp = ampdb(-36)
  k4form init 3500
  k4band init 130

  ; Fifth formant.
  k5amp = ampdb(-60)
  k5form init 4950
  k5band init 140

  a1 fof k1amp, kfund, k1form, koct, k1band, kris, \
         kdur, kdec, iolaps, ifna, ifnb, itotdur
  a2 fof k2amp, kfund, k2form, koct, k2band, kris, \
         kdur, kdec, iolaps, ifna, ifnb, itotdur
  a3 fof k3amp, kfund, k3form, koct, k3band, kris, \
         kdur, kdec, iolaps, ifna, ifnb, itotdur
  a4 fof k4amp, kfund, k4form, koct, k4band, kris, \
         kdur, kdec, iolaps, ifna, ifnb, itotdur
  a5 fof k5amp, kfund, k5form, koct, k5band, kris, \
         kdur, kdec, iolaps, ifna, ifnb, itotdur

  ; Combine all of the formants together.
  out (a1+a2+a3+a4+a5) * 16384
endin


</CsInstruments>
<CsScore>


; Table #1, a sine wave.
f 1 0 4096 10 1
; Table #2.
f 2 0 1024 19 0.5 0.5 270 0.5

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Les valeurs de formant pour le "a" en voix d'alto proviennent de l'*Appendice Valeurs de Formant*.

Voir Aussi

fof2, *Appendice Valeurs de Formant*

Crédits

Ajouté dans la version 1 (1990)

fof2

fof2 — Produit des grains FOF (sinusoïde amortie) incluant une indexation incrémentielle de taux-k avec chaque grain.

Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-k. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

fof2 implémente une indexation incrémentielle de taux-k dans la fonction *ifna* ave chaque grain successif.

Syntaxe

```
ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kgliss [, iskip]
```

Initialisation

iolaps -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund * kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

ifna, ifnb -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

itotdur -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

iskip (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

Exécution

xamp -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons Q < 10) et/ou quand les grains FOF se superposent en partie.

xfund -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

xform -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu (voir *ifmode*).

koct -- indice d'octaviation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

kband -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

kris, kdur, kdec -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

kphs -- permet d'indexer au taux-k la table de fonction *ifna* avec chaque grain successif, ce qui permet d'appliquer le recalage temporel. Les valeurs de *kphs* sont normalisées entre 0 et 1, 1 étant la fin de la table de fonction *ifna*.

kgliss -- fixe la hauteur finale de chaque grain en fontion de sa hauteur initiale, en octaves. Ainsi *kgliss* = 2 signifie que le grain se termine deux octaves plus haut que sa hauteur initiale, tandis qu'avec *kgliss* = -3/4 le grain se termine une sixte majeure plus bas. Chaque 1/12 ajouté à *kgliss* élève la hauteur finale d'un demi-ton. Si vous ne voulez pas de glissando, fixez *kgliss* à 0.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur, xfund*, et de la densité des chevauchements.

Note

La fréquence finale de chaque grain étant égale à *kform* * (2 ^ *kgliss*), des valeurs trop importantes de *kgliss* pourront provoquer un repliement. Par exemple, *kform* = 3000 et *kgliss* = 3 placent la fréquence finale au-delà de la fréquence de Nyquist si *sr* = 44100. Il est prudent de pondérer *kgliss* en conséquence.

Exemples

Voici un exemple de l'opcode fof2. Il utilise le fichier *fof2.csd* [examples/fof2.csd].

Exemple 191. Exemple de l'opcode fof2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

;By Andres Cabrera 2007

    instr 1             ;table-lookup vocal synthesis
```

```
kris init p12
kdur init p13
kdec init p14

iolaps init p15

ifna init 1  ; Sine wave
ifnb init 2  ; Straight line rise shape

itotdur init p3

kphs init 0  ; No phase modulation (constant kphs)


kfund line p4, p3, p5
kform line p6, p3, p7
koct line p8, p3, p9
kband line p10, p3, p11
kgliss line p16, p3, p17

kenv linen 5000, 0.03, p3, 0.03  ;to avoid clicking


aout    fof2    kenv, kfund, kform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kgliss


    outs    aout, aout
    endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;              kfund1  kfund2  kform1  kform2  koct1  koct2  kband1 kband2 kris  kdur  kdec  iolaps  kg
i1    0    4     220     220     510     510     0      0      30     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     910     0      0      30     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     510     0      0     100     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     510     0      1      30     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     510     0      0      30     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     510     0      0      30     30   0.01  0.03  0.01   20
i1    +    .     220     220     510     510     0      0      30     30   0.01  0.05  0.01   100
i1    +    .     220     440     510     510     0      0      30     30   0.01  0.05  0.01   100

e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode fof2, qui produit des sons de voyelle en utilisant des formants générés par fof2 avec les valeurs de l'appendice *Valeurs de Formant*. Il utilise le fichier *fof2-2.csd* [examples/fof2-2.csd].

Exemple 192. Exemple de l'opcode fof2 pour produire des sons de voyelle.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

; Example by Chuckk Hubbard 2007
```

```
    instr 1             ;table-lookup vocal synthesis

iolaps    =    120
ifna    =    1        ;f1 - sine wave
ifnb    =    2        ;f2 - linear rise shape
itotdur    =    p3
iamp    =    p4 * 0dbfs
ifreq1    =    p5        ;starting frequency
ifreq2    =    p6        ;ending frequency

kamp    linseg    0, .003, iamp, itotdur-.007, iamp, .003, 0, .001, 0
kfund    expseg    ifreq1, itotdur, ifreq2
koct    init    0
kris    init    .003
kdur    init    .02
kdec    init    .007
kphs    init    0
kgliss    init    0

iforma    =    p7        ;starting spectrum
iformb    =    p8        ;ending spectrum

iform1a    tab_i    0, iforma        ;read values of 5 formants for 1st spectrum
iform2a    tab_i    1, iforma
iform3a    tab_i    2, iforma
iform4a    tab_i    3, iforma
iform5a    tab_i    4, iforma
idb1a    tab_i    5, iforma        ;read decibel levels for same 5 formants
idb2a    tab_i    6, iforma
idb3a    tab_i    7, iforma
idb4a    tab_i    8, iforma
idb5a    tab_i    9, iforma
iband1a    tab_i    10, iforma    ;read bandwidths for same 5 formants
iband2a    tab_i    11, iforma
iband3a    tab_i    12, iforma
iband4a    tab_i    13, iforma
iband5a    tab_i    14, iforma
iamp1a    =    ampdb(idb1a)    ;convert db to linear multipliers
iamp2a    =    ampdb(idb2a)
iamp3a    =    ampdb(idb3a)
iamp4a    =    ampdb(idb4a)
iamp5a    =    ampdb(idb5a)

iform1b    tab_i    0, iformb        ;values of 5 formants for 2nd spectrum
iform2b    tab_i    1, iformb
iform3b    tab_i    2, iformb
iform4b    tab_i    3, iformb
iform5b    tab_i    4, iformb
idb1b    tab_i    5, iformb        ;decibel levels for 2nd set of formants
idb2b    tab_i    6, iformb
idb3b    tab_i    7, iformb
idb4b    tab_i    8, iformb
idb5b    tab_i    9, iformb
iband1b    tab_i    10, iformb    ;bandwidths for 2nd set of formants
iband2b    tab_i    11, iformb
iband3b    tab_i    12, iformb
iband4b    tab_i    13, iformb
iband5b    tab_i    14, iformb
iamp1b    =    ampdb(idb1b)    ;convert db to linear multipliers
iamp2b    =    ampdb(idb2b)
iamp3b    =    ampdb(idb3b)
iamp4b    =    ampdb(idb4b)
iamp5b    =    ampdb(idb5b)

kform1    line    iform1a, itotdur, iform1b    ;transition between formants
kform2    line    iform2a, itotdur, iform2b
kform3    line    iform3a, itotdur, iform3b
kform4    line    iform4a, itotdur, iform4b
kform5    line    iform5a, itotdur, iform5b
kband1    line    iband1a, itotdur, iband1b    ;transition of bandwidths
kband2    line    iband2a, itotdur, iband2b
kband3    line    iband3a, itotdur, iband3b
kband4    line    iband4a, itotdur, iband4b
kband5    line    iband5a, itotdur, iband5b
kamp1    line    iamp1a, itotdur, iamp1b    ;transition of amplitudes of formants
kamp2    line    iamp2a, itotdur, iamp2b
kamp3    line    iamp3a, itotdur, iamp3b
kamp4    line    iamp4a, itotdur, iamp4b
kamp5    line    iamp5a, itotdur, iamp5b

;5 formants for each spectrum
```

```
a1    fof2    kamp1, kfund, kform1, koct, kband1, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a2    fof2    kamp2, kfund, kform2, koct, kband2, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a3    fof2    kamp3, kfund, kform3, koct, kband3, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a4    fof2    kamp4, kfund, kform4, koct, kband4, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a5    fof2    kamp5, kfund, kform5, koct, kband5, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,

aout    =    (a1+a2+a3+a4+a5) * kamp/5    ;sum and scale

aenv linen 1, 0.05, p3, 0.05  ;to avoid clicking

    outs    aout*aenv, aout*aenv
    endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;*****************************************************************
; tables of formant values adapted from MiscFormants.html
; 100's: soprano    200's: alto    300's: countertenor        400's: tenor    500's: bass
; -01: "a" sound    -02: "e" sound    -03: "i" sound    -04: "o" sound    -05: "u" sound
; p-5 through p-9: frequencies of 5 formants
; p-10 through p-14: decibel levels of 5 formants
; p-15 through p-19: bandwidths of 5 formants

;        formant frequencies            decibel levels                bandwidths
;soprano
f101 0 16 -2    800      1150      2900      3900      4950      0.001     -6      -32      -20      -50
f102 0 16 -2    350      2000      2800      3600      4950      0.001      -20      -15      -40      -56      6
f103 0 16 -2    270      2140      2950      3900      4950      0.001      -12      -26      -26      -44      60
f104 0 16 -2    450      800      2830      3800      4950      0.001      -11      -22      -22      -50      40
f105 0 16 -2    325      700      2700      3800      4950      0.001      -16      -35      -40      -60      50
;alto
f201 0 16 -2    800      1150      2800      3500      4950      0.001     -4      -20      -36      -60     8
f202 0 16 -2    400      1600      2700      3300      4950      0.001     -24      -30      -35      -60
f203 0 16 -2     350      1700      2700      3700      4950      0.001     -20      -30      -36      -60
f204 0 16 -2    450      800      2830      3500      4950      0.001     -9      -16      -28      -55
f205 0 16 -2    325      700      2530      3500      4950      0.001     -12      -30      -40      -64
;countertenor
f301 0 16 -2    660      1120      2750      3000      3350      0.001     -6      -23      -24      -38
f302 0 16 -2    440      1800      2700      3000      3300      0.001      -14      -18      -20      -20      70
f303 0 16 -2    270      1850      2900      3350      3590      0.001      -24      -24      -36      -36      40
f304 0 16 -2    430      820      2700      3000      3300      0.001      -10      -26      -22      -34      40
f305 0 16 -2    370      630      2750      3000      3400      0.001      -20      -23      -30      -34      40
;tenor
f401 0 16 -2    650      1080      2650      2900      3250      0.001     -6      -7      -8      -22      8
f402 0 16 -2    400      1700      2600      3200      3580      0.001      -14      -12      -14      -20      70
f403 0 16 -2    290      1870      2800      3250      3540      0.001      -15      -18      -20      -30      40
f404 0 16 -2    400      800      2600      2800      3000      0.001      -10      -12      -12      -26      70
f405 0 16 -2    350      600      2700      2900      3300      0.001      -20      -17      -14      -26      40
;bass
f501 0 16 -2    600      1040      2250      2450      2750      0.001     -7      -9      -9      -20      6
f502 0 16 -2    400      1620      2400      2800      3100      0.001      -12      -9      -12      -18
f503 0 16 -2    250      1750      2600      3050      3340      0.001     -30      -16      -22      -28
f504 0 16 -2    400      750      2400      2600      2900      0.001     -11      -21      -20      -40
f505 0 16 -2    350      600      2400      2675      2950      0.001     -20      -32      -28      -36
;*****************************************************************

;    start dur   amp    start freq    end freq    start formant    end formant
i1    0    1    .8       440         412.5         201         203
i1    +    .    .8       412.5         550         201         204
i1    +    .    .8       495         330         202         205

i1    +    .    .8       110         103.125         501         503
i1    +    .    .8       103.125 137.5               501         504
i1    +    .    .8       123.75 82.5         502         505

i1    7    .    .4       440         412.5         201         203
i1    8    .    .4       412.5         550         201         204
i1    9    .    .4       495         330         202         205
i1    7    .    .4       110         103.125         501         503
i1    8    .    .4       103.125 137.5         501         504
i1    9    .    .4       123.75 82.5         502         505
i1    +    .    .4       440         412.5         101         103
i1    +    .    .4       412.5         550         101         104
i1    +    .    .4       495         330         102         105
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fof

Crédits

Auteur : Rasmus Ekman
fof2 est une modification de *fof* par Rasmus Ekman

Nouveau dans Csound 3.45

fofilter

fofilter — Formant filter.

Description

Fofilter generates a stream of overlapping sinewave grains, when fed with a pulse train. Each grain is the impulse response of a combination of two BP filters. The grains are defined by their attack time (determining the skirtwidth of the formant region at -60dB) and decay time (-6dB bandwidth). Overlapping will occur when 1/freq < decay, but, unlike FOF, there is no upper limit on the number of overlaps. The original idea for this opcode came from J McCartney's formlet class in SuperCollider, but this is possibly implemented differently(?).

Syntax

```
asig fofilter ain, kcf, kris, kdec[, istor]
```

Initialization

istor --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal.

kcf -- filter centre frequency

kris -- impulse response attack time (secs).

kdec -- impulse response decay time (secs).

Examples

Exemple 193. Example

```
kfe         expseg 10, p3*0.9, 180, p3*0.1, 175
kenv        linen  1000, 0.05, p3, 0.05
asig        buzz   kenv, kfe, sr/(2*kfe), 1
afil        fofilter  asig, 900, 0.007, 0.04

            out afil
```

Credits

Author: Victor Lazzarini;
January 2005

New plugin in version 5

January 2005.

fog

fog — La sortie audio est une succession de grains obtenus à partir des données d'une table de fonction.

Description

La sortie audio est une succession de grains obtenus à partir des données de la table de fonction *ifna*. L'enveloppe locale de ces grains et leur distribution temporelle sont basées sur le modèle de la synthèse *fof* et permettent un contrôle détaillé de la synthèse granulaire.

Syntaxe

```
ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \
      iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]
```

Initialisation

iolaps -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la densité, et l'espace requis dépend de la valeur maximale de *xdens * kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

ifna, ifnb -- numéros de table de deux fonctions. La première contient les données utilisées pour la granulation, provenant habituellement d'un fichier son (*GEN01*). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains ; cette forme est normalement une sigmoïde (*GEN19*) mais elle peut être aussi linéaire (*GEN05*).

itotdur -- durée totale durant laquelle ce *fog* sera actif. Fixée normalement à p3. Aucun nouveau grain n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

iphs (facultatif) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

itmode (facultatif) -- type de transposition. S'il est nul, chaque grain garde la valeur *xtrans* avec laquelle il a été amorcé. Sinon, chaque grain est influencé par *xtrans* de manière continue. La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

Exécution

xamp -- facteur d'amplitude. L'amplitude dépend également du nombre de grains se chevauchant, de l'interaction de la forme montante (*ifnb*) et de la chute exponentielle (*kband*), et des valeurs de la forme d'onde du grain (*ifna*). L'amplitude réelle peut ainsi dépasser *xamp*.

xdens -- densité. Nombre de grains par seconde.

xtrans -- facteur de transposition. Le taux de lecture des données de la table de fonction *ifna* dans chaque grain. Il a pour effet de tranposer le matériel original. Une valeur de 1 produit la hauteur originale. Les valeurs supérieures à 1 transposent vers le haut tandis que les valeurs inférieures à 1 le font vers le bas. Les valeurs négatives provoquent une lecture à l'envers de la table.

aspd -- indice de lecture initial. *aspd* est l'indice de lecture normalisé (0 à 1) dans la table *ifna* qui détermine le mouvement d'un pointeur à partir duquel commence la lecture dans chaque grain. (*xtrans* détermine le taux de lecture des données à partir de ce pointeur.)

koct -- indice d'octaviation. Ce paramètre fonctionne de manière identique à celui qui est décrit dans *fof*.

kband, *kris*, *kdur*, *kdec* -- forme de l'enveloppe du grain. Ces paramètres déterminent les temps de décroissance exponentielle (*kband*), et d'attaque (*kris*), la durée totale (*kdur*), et celle de la chute (*kdec*) de l'enveloppe du grain. Leur mode opératoire est identique à celui des paramètres d'enveloppe locale dans *fof*.

Exemples

```
;p4 = facteur de transposition
;p5 = facteur de vitesse
;p6 = table de fonction pour les données du grain
i1  = sr/ftlen(p6) ; prise en compte du taux d'échantillonnage et de la longueur de la table
a1 phasor i1*p5 ; indice pour la vitesse
a2 fog    5000, 100, p4, a1, 0, 0, , .01, .02, .01, 2, p6, 1, p3, 0, 1
```

Crédits

Auteur : Michael Clark
Huddersfield
Mai 1997

Nouveau dans la version 3.46

Le générateur *fog* de Csound a été écrit par Michael Clarke, comme suite à ses travaux antérieurs basés sur l'algorithme FOF de l'IRCAM.

Notes ajoutées par Rasmus Ekman en septembre 2002.

fold

fold — Adds artificial foldover to an audio signal.

Description

Adds artificial foldover to an audio signal.

Syntax

```
ares fold asig, kincr
```

Performance

asig -- input signal

kincr -- amount of foldover expressed in multiple of sampling rate. Must be >= 1

fold is an opcode which creates artificial foldover. For example, when *kincr* is equal to 1 with sr=44100, no foldover is added. When *kincr* is set to 2, the foldover is equivalent to a downsampling to 22050, when it is set to 4, to 11025 etc. Fractional values of *kincr* are possible, allowing a continuous variation of foldover amount. This can be used for a wide range of special effects.

Examples

Here is an example of the fold opcode. It uses the file *fold.csd* [examples/fold.csd].

Exemple 194. Example of the fold opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fold.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use an ordinary sine wave.
  asig oscils 30000, 100, 1

  ; Vary the fold-over amount from 1 to 200.
  kincr line 1, p3, 200
  a1 fold asig, kincr

  out a1
```

```
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for four seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

follow

follow — Envelope follower unit generator.

Description

Envelope follower unit generator.

Syntax

```
ares follow asig, idt
```

Initialization

idt -- This is the period, in seconds, that the average amplitude of *asig* is reported. If the frequency of *asig* is low then *idt* must be large (more than half the period of *asig*)

Performance

asig -- This is the signal from which to extract the envelope.

Examples

Here is an example of the follow opcode. It uses the file *follow.csd* [examples/follow.csd], and *beats.wav* [examples/beats.wav].

Exemple 195. Example of the follow opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o follow.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play a WAV file.
instr 1
  a1 soundin "beats.wav"
  out a1
endin

; Instrument #2 - have another waveform follow the WAV file.
instr 2
  ; Follow the WAV file.
  as soundin "beats.wav"
```

```
  af follow as, 0.01

  ; Use a sine waveform.
  as oscil 4000, 440, 1
  ; Have it use the amplitude of the followed WAV file.
  a1 balance as, af

  out a1
endin


</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

To avoid zipper noise, by discontinuities produced from complex envelope tracking, a lowpass filter could be used, to smooth the estimated envelope.

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

follow2

follow2 — Another controllable envelope extractor.

Description

A controllable envelope extractor using the algorithm attributed to Jean-Marc Jot.

Syntax

```
ares follow2 asig, katt, krel
```

Performance

asig -- the input signal whose envelope is followed

katt -- the attack rate (60dB attack time in seconds)

krel -- the decay rate (60dB decay time in seconds)

The output tracks the amplitude envelope of the input signal. The rate at which the output grows to follow the signal is controlled by the *katt*, and the rate at which it decreases in response to a lower amplitude, is controlled by the *krel*. This gives a smoother envelope than *follow*.

Examples

Here is an example of the follow2 opcode. It uses the file *follow2.csd* [examples/follow2.csd], and *beats.wav* [examples/beats.wav].

Exemple 196. Example of the follow2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o follow2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play a WAV file.
instr 1
  a1 soundin "beats.wav"
  out a1
endin

; Instrument #2 - have another waveform follow the WAV file.
```

```
instr 2
  ; Follow the WAV file.
  as soundin "beats.wav"
  af follow2 as, 0.01, 0.1

  ; Use a noise waveform.
  ar rand 44100
  ; Have it use the amplitude of the followed WAV file.
  a1 balance ar, af

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch
The algorithm for the *follow2* is attributed to Jean-Marc Jot.
University of Bath, Codemist Ltd.
Bath, UK
February 2000

Example written by Kevin Conder.

New in Csound version 4.03

Added notes by Rasmus Ekman on September 2002.

foscil

foscil — Un oscillateur élémentaire modulé en fréquence.

Description

Un oscillateur élémentaire modulé en fréquence.

Syntaxe

```
ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

iphs (faculatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

xamp -- l'amplitude du signal de sortie.

kcps -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

xcar -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

xmod -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

kndx -- l'indice de modulation.

foscil est une unité composée qui assemble deux opcodes *oscil* dans la configuration familière de synthèse FM de Chowning, où la sortie au taux audio de l'un des générateurs est utilisée pour moduler l'entrée en fréquence de l'autre (la « porteuse »). Fréquence de la porteuse = *kcps* * *xcar* et fréquence modulante = *kcps* * *xmod*. Pour des valeurs entières de *xcar* et de *xmod*, la fondamentale perçue sera la valeur positive minimale de *kcps* * (*xcar* - *n* * *xmod*), *n* = 0,1,2,... L'entrée *kndx* est l'indice de modulation (habituellement variant dans le temps approximativement dans l'intervalle de 0 à 4) qui détermine la distribution de l'energie acoustique parmi les positions des partiels données par *n* = 0,1,2,.., etc. *ifn* doit pointer sur une onde sinus stockée. Avant la version 3.50, *xcar* et *xmod* ne pouvaient être que de taux-k.

La formule utilisée pour cette implémentation de la synthèse FM est *xamp* * cos(2# * *t* * *kcps* * *xcar* + *kndx* * sin(2# * *t* * *kcps* * *xmod*) - #), en supposant que la table est une onde sinus.

Exemples

Voici un exemple de l'opcode foscil. Il utilise le fichier *foscil.csd* [examples/foscil.csd].

Exemple 197. Exemple de l'opcode foscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o foscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic FM waveform.
instr 1
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  a1 foscil kamp, kcps, kcar, kmod, kndx, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Exemple écrit par Kevin Conder.

foscili

foscili — Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

Description

Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

Syntaxe

```
ares foscili xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

iphs (faculatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

xamp -- l'amplitude du signal de sortie.

kcps -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

xcar -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

xmod -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

kndx -- l'indice de modulation.

foscili diffère de *foscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture des échantillons est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

Exemples

Voici un exemple de l'opcode foscili. Il utilise le fichier *foscili.csd* [examples/foscili.csd].

Exemple 198. Exemple de l'opcode foscili.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o foscili.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic FM waveform.
instr 1
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  a1 foscil kamp, kcps, kcar, kmod, kndx, ifn
  out a1
endin

; Instrument #2 - the basic FM waveform with extra interpolation.
instr 2
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  a1 foscili kamp, kcps, kcar, kmod, kndx, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 4096 10 1

; Play Instrument #1, the basic FM instrument, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the interpolated FM instrument, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Exemple écrit par Kevin Conder.

fout

fout — Outputs a-rate signals to an arbitrary number of channels.

Description

fout outputs *N* a-rate signals to a specified file of *N* channels.

Syntax

```
fout ifilename, iformat, aout1 [, aout2, aout3,...,aoutN]
```

Initialization

ifilename -- the output file's name (in double-quotes).

iformat -- a flag to choose output file format (note: Csound versions older than 5.0 may only support formats 0, 1, and 2):

- 0 - 32-bit floating point samples without header (binary PCM multichannel file)
- 1 - 16-bit integers without header (binary PCM multichannel file)
- 2 - 16-bit integers with a header. The header type depends on the render (-o) format. For example, if the user chooses the AIFF format (using the *-A flag*), the header format will be AIFF type.
- 3 - u-law samples with a header (see iformat=2).
- 4 - 16-bit integers with a header (see iformat=2).
- 5 - 32-bit integers with a header (see iformat=2).
- 6 - 32-bit floats with a header (see iformat=2).
- 7 - 8-bit unsigned integers with a header (see iformat=2).
- 8 - 24-bit integers with a header (see iformat=2).
- 9 - 64-bit floats with a header (see iformat=2).

In addition, Csound versions 5.0 and later allow for explicitly selecting a particular header type by specifying the format as 10 * fileType + sampleFormat, where fileType may be 1 for WAV, 2 for AIFF, 3 for raw (headerless) files, and 4 for IRCAM; sampleFormat is one of the above values in the range 0 to 9, except sample format 0 is taken from the command line (-o), format 1 is 8-bit signed integers, and format 2 is a-law. So, for example, iformat=25 means 32-bit integers with AIFF header.

Performance

aout1,... aoutN -- signals to be written to the file. In the case of raw files, the expected range of audio signals is determined by the selected sample format; for sound files with a header like WAV and AIFF, the audio signals should be in the range -0dbfs to 0dbfs.

fout (file output) writes samples of audio signals to a file with any number of channels. Channel number

depends by the number of *aoutN* variables (i.e. a mono signal with only an a-rate argument, a stereo signal with two a-rate arguments etc.) Maximum number of channels is fixed to 64. Multiple *fout* opcodes can be present in the same instrument, referring to different files.

Notice that, unlike *out*, *outs* and *outq*, *fout* does not zero the audio variable so you must zero it after calling it. If polyphony is to be used, you can use *vincr* and *clear* opcodes for this task.

Notice that *fout* and *foutk* can use either a string containing a file pathname, or a handle-number generated by *fiopen*. Whereas, with *fouti* and *foutir*, the target file can be only specified by means of a handle-number.

Note

If you are usinf *fout* to generate an audio file for the global output of csound, you might want to use the *monitor* opcode, which can tap the output buffer, to avoid having to route

Examples

Here is a simple example of the fout opcode. It uses the file *fout.csd* [examples/fout.csd].

Exemple 199. Example of the fout opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fout.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  ; Create an audio signal.
  asig oscils iamp, icps, iphs

  ; Write the audio signal to a headerless audio file
  ; called "fout.raw".
  fout "fout.raw", 1, asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Here is an example of the fout opcode with a polyphonic score. It uses the file *fout_poly.csd* [examples/fout_poly.csd] and *beats.wav* [examples/beats.wav].

Exemple 200. Example of the fout opcode with a polyphonic score.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fout_poly.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - Play an audio file.
instr 1
  ; Generate an audio signal using
  ; the audio file "beats.wav".
  asig soundin "beats.wav"

  out asig
endin

; Instrument #2 - Create a basic tone.
instr 2
  iamp = 5000
  icps = 440
  iphs = 0

  ; Create an audio signal.
  asig oscils iamp, icps, iphs

  out asig
endin

; Instrument #99 - Save the global signal to a file.
instr 99
  ; Read the csound output buffer
  aoutput monitor
  ; Write the output of csound to a headerless
  ; audio file called "fout_poly.raw".
  fout "fout_poly.raw", 1, aoutput

endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2

; Play Instrument #2 every quarter-second.
i 2 0.00 0.1
i 2 0.25 0.1
i 2 0.50 0.1
i 2 0.75 0.1
i 2 1.00 0.1
i 2 1.25 0.1
i 2 1.50 0.1
i 2 1.75 0.1

; Make sure the global instrument, #99, is running
; during the entire performance (2 seconds).
i 99 0 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Here is another example of *fout*, using it to save the contents of a table to an audio file. It uses the file *fout_ftable.csd* [examples/fout_ftable.csd] and *beats.wav* [examples/beats.wav].

Exemple 201. Example of the fout opcode to save the contents of an f-table.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac          -iadc      -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fout_ftable.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

  gilen         =         131072
  gicps         =         sr/gilen
  gitab         ftgen         1, 0, gilen, 10, 1

                instr 1

  /******** write file to table ********/

  ain         diskin         "beats.wav", 1, 0, 1
  aphs         phasor         gicps
  andx         =         aphs * gilen
               tablew         ain, andx, gitab

  /******** write table to file ********/

  aosc         table         aphs, gitab, 1
               out         aosc
               fout         "beats_copy.wav", 6, aosc

                endin

</CsInstruments>
<CsScore>
i1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

See Also

fiopen, *fouti*, *foutir*, *foutk*, *monitor*

Credits

Author: Gabriel Maldonado
Italy
1999

The simple example was written by Kevin Conder.

New in Csound version 3.56

October 2002. Added a note from Richard Dobson.

fouti

fouti — Outputs i-rate signals of an arbitrary number of channels to a specified file.

Description

fouti output *N* i-rate signals to a specified file of *N* channels.

Syntax

```
fouti ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]
```

Initialization

ihandle -- a number which specifies this file.

iformat -- a flag to choose output file format:

- 0 - floating point in text format
- 1 - 32-bit floating point in binary format

iflag -- choose the mode of writing to the ASCII file (valid only in ASCII mode; in binary mode *iflag* has no meaning, but it must be present anyway). *iflag* can be a value chosen among the following:

- 0 - line of text without instrument prefix
- 1 - line of text with instrument prefix (see below)
- 2 - reset the time of instrument prefixes to zero (to be used only in some particular cases. See below)

iout,..., ioutN -- values to be written to the file

Performance

fouti and *foutir* write i-rate values to a file. The main use of these opcodes is to generate a score file during a realtime session. For this purpose, the user should set *iformat* to 0 (text file output) and *iflag* to 1, which enable the output of a prefix consisting of the strings *inum*, *actiontime*, and *duration*, before the values of *iout1...ioutN* arguments. The arguments in the prefix refer to instrument number, action time and duration of current note.

Notice that *fout* and *foutk* can use either a string containing a file pathname, or a handle-number generated by *fiopen*. Whereas, with *fouti* and *foutir*, the target file can be only specified by means of a handle-number.

See Also

fiopen, *fout*, *foutir*, *foutk*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

foutir

foutir — Outputs i-rate signals from an arbitrary number of channels to a specified file.

Description

foutir output *N* i-rate signals to a specified file of *N* channels.

Syntax

```
foutir ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]
```

Initialization

ihandle -- a number which specifies this file.

iformat -- a flag to choose output file format:

- 0 - floating point in text format
- 1 - 32-bit floating point in binary format

iflag -- choose the mode of writing to the ASCII file (valid only in ASCII mode; in binary mode *iflag* has no meaning, but it must be present anyway). *iflag* can be a value chosen among the following:

- 0 - line of text without instrument prefix
- 1 - line of text with instrument prefix (see below)
- 2 - reset the time of instrument prefixes to zero (to be used only in some particular cases. See below)

iout,..., ioutN -- values to be written to the file

Performance

fouti and *foutir* write i-rate values to a file. The main use of these opcodes is to generate a score file during a realtime session. For this purpose, the user should set *iformat* to 0 (text file output) and *iflag* to 1, which enable the output of a prefix consisting of the strings *inum*, *actiontime*, and *duration*, before the values of *iout1...ioutN* arguments. The arguments in the prefix refer to instrument number, action time and duration of current note.

The difference between *fouti* and *foutir* is that, in the case of *fouti*, when *iflag* is set to 1, the duration of the first opcode is undefined (so it is replaced by a dot). Whereas, *foutir* is defined at the end of note, so the corresponding text line is written only at the end of the current note (in order to recognize its duration). The corresponding file is linked by the *ihandle* value generated by the *fiopen* opcode. So *fouti* and *foutir* can be used to generate a Csound score while playing a realtime session.

Notice that *fout* and *foutk* can use either a string containing a file pathname, or a handle-number generated by *fiopen*. Whereas, with *fouti* and *foutir*, the target file can be only specified by means of a handle-number.

See Also

fiopen, *fout*, *fouti*, *foutk*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

foutk

foutk — Outputs k-rate signals of an arbitrary number of channels to a specified file, in raw (headerless) format.

Description

foutk outputs *N* k-rate signals to a specified file of *N* channels.

Syntax

```
foutk ifilename, iformat, kout1 [, kout2, kout3,....,koutN]
```

Initialization

ifilename -- the output file's name (in double-quotes).

iformat -- a flag to choose output file format (note: Csound versions older than 5.0 may only support formats 0 and 1):

- 0 - 32-bit floating point samples without header (binary PCM multichannel file)
- 1 - 16-bit integers without header (binary PCM multichannel file)
- 2 - 16-bit integers without header (binary PCM multichannel file)
- 3 - u-law samples without header
- 4 - 16-bit integers without header
- 5 - 32-bit integers without header
- 6 - 32-bit floats without header
- 7 - 8-bit unsigned integers without header
- 8 - 24-bit integers without header
- 9 - 64-bit floats without header

Performance

kout1,...koutN -- control-rate signals to be written to the file. The expected range of the signals is determined by the selected sample format.

foutk operates in the same way as *fout*, but with k-rate signals. *iformat* can be set only in the range 0 to 9, or 0 to 1 with an old version of Csound.

Notice that *fout* and *foutk* can use either a string containing a file pathname, or a handle-number generated by *fiopen*. Whereas, with *fouti* and *foutir*, the target file can be only specified by means of a handle-number.

See Also

fiopen, *fout*, *fouti*, *foutir*

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

fprintks

fprintks — Similar to printks but prints to a file.

Description

Similar to *printks* but prints to a file.

Syntax

```
fprintks "filename", "string", [, kval1] [, kval2] [...]
```

Initialization

"filename" -- name of the output file.

"string" -- the text string to be printed. Can be up to 8192 characters and must be in double quotes.

Performance

kval1, kval2, ... (optional) -- The k-rate values to be printed. These are specified in « *string* » with the standard C value specifier (%f, %d, etc.) in the order given.

fprintks is similar to the *printks* opcode except it outputs to a file and doesn't have a *itime* parameter. For more information about output formatting, please look at *printks's documentation*.

Examples

Here is an example of the fprintks opcode. It uses the file *fprintks.csd* [examples/fprintks.csd].

Exemple 202. Example of the fprintks opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fprintks.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a score generator example.
instr 1
  ; K-rate stuff.
  kstart init 0
```

```
  kdur linrand 10
  kpitch linrand 8

  ; Printing to to a file called "my.sco".
  fprintks "my.sco", "i1\\t%2.2f\\t%2.2f\\t%2.2f\\n", kstart, kdur, 4+kpitch

  knext linrand 1
  kstart = kstart + knext
endin


</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001


</CsScore>
</CsoundSynthesizer>
```

This example will generate a file called « my.sco ». It should contain lines like this:

```
i1      0.00    3.94    10.26
i1      0.20    3.35    6.22
i1      0.67    3.65    11.33
i1      1.31    1.42    4.13
```

Here is an example of the fprintks opcode, which converts a standard MIDI file to a csound score. It uses the file *fprintks-2.csd* [examples/fprintks-2.csd].

Exemple 203. Example of the fprintks opcode to convert a MIDI file to a csound score.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
; -odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n -Fmidichn_advanced.mid
;Don't write audio ouput to disk and use the file midichn_advanced.mid as MIDI input
</CsOptions>
<CsInstruments>

  sr     =  48000
  ksmps     =  16
  nchnls    =  2

  ;Example by Jonathan Murphy 2007

            ; assign all midi events to instr 1000
            massign   0, 1000
            pgmassign 0, 1000

    instr 1000

  ktim timeinsts

  kst, kch, kd1, kd2  midiin
if (kst != 0) then
;  p4 = MIDI event type   p5 = channel   p6= data1    p7= data2
            fprintks  "MIDI2cs.sco", "i1\\t%f\\t%f\\t%d\\t%d\\t%d\\t%d\\n", ktim, 1/kr, kst, kch, kd1,
endif

    endin


</CsInstruments>
<CsScore>
```

```
i1000 0 10000
e
</CsScore>
</CsoundSynthesizer>
```

This example will generate a file called « MIDI2cs.sco » containing i-events according to the MIDI file

Here is an advanced example of the fprintks opcode, which generates scores for Csound. It uses the file *scogen-2.csd* [examples/scogen.csd].

Exemple 204. Example of the fprintks opcode to create a Csound score file generator using Csound.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
; -odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n
;Don't write audio ouput to disk
</CsOptions>
<CsInstruments>
;============================================================
;        scogen.csd       by: Matt Ingalls
;
;    a "port" of sorts
;      of the old "mills" score generator (scogen)
;
;    this instrument creates a schottstaedt.sco file
;    to be used with the schottstaedt.orc file
;
;    as long as you dont save schottstaedt.orc as a .csd
;    file, you should be able to keep it open in MacCsound
;    and render each newly generated .sco file.
;
;============================================================

gScoName = "/Users/matt/Desktop/schottstaedt.sco"     ; the name of the file to be generated

    sr    =    100     ; this defines our temporal resolution,
                ; an sr of 100 means we will generate p2 and p3 values
                ; to the nearest 1/100th of a second

    ksmps =    1     ; set kr=sr so we can do everything at k-rate


; some print opcodes
opcode PrintInteger, 0, k
    kval    xin
        fprintks    gScoName, "%d", kval
endop

opcode PrintFloat, 0, k
    kval    xin
        fprintks    gScoName, "%f", kval
endop

opcode PrintTab, 0, 0
    fprintks    gScoName, "%n"
endop

opcode PrintReturn, 0, 0
    fprintks    gScoName, "%r"
endop


; recursively calling opcode to handle all the optional parameters
opcode ProcessAdditionalPfields, 0, ikio
    iPtable, kndx, iNumPfields, iPfield xin

    ; additional pfields start at 5, we use a default 0 to identify the first call
    iPfield = (iPfield == 0 ? 5 : iPfield)
```

```
    if (iPfield > iNumPfields) goto endloop
        ; find our tables
        iMinTable table    2*iPfield-1, iPtable
        iMaxTable table    2*iPfield, iPtable

        ; get values from our tables
        kMin tablei    kndx, iMinTable
        kMax tablei    kndx, iMaxTable

        ; find a random value in the range and write it to the score
        fprintks gScoName, "%t%f", kMin + rnd(kMax-kMin)

        ; recursively call for any additional pfields.
        ProcessAdditionalPfields iPtable, kndx, iNumPfields, iPfield + 1
    endloop:

endop


/* ============================================================
    Generate a gesture of i-statements

    p2 = start of the gesture
    p3 = duration of the gesture
    p4 = number of a function that contains a list of all
        function table numbers used to define the
        pfield random distribution
    p5 = scale generated p4 values according to density (0=off, 1=on) [todo]
    p6 = let durations overlap gesture duration (0=off, 1=on) [todo]
    p7 = seed for random number generator seed [todo]
  ============================================================
*/
instr Gesture

    ; initialize
    iResolution = 1/sr

    kNextStart init p2
    kCurrentTime init p2

    iNumPfields table        0, p4
    iInstrMinTable table    1, p4
    iInstrMaxTable table    2, p4
    iDensityMinTable table    3, p4
    iDensityMaxTable table    4, p4
    iDurMinTable table    5, p4
    iDurMaxTable table    6, p4
    iAmpMinTable table    7, p4
    iAmpMaxTable table    8, p4

    ; check to make sure there is enough data
    print iNumPfields
    if iNumPfields < 4 then
        prints "%dError: At least 4 p-fields (8 functions) need to be specified.%n", iNumPfields
        turnoff
    endif

    ; initial comment
    fprints    gScoName, "%!Generated Gesture from %f to %f seconds%n %!%t%twith a p-max of %d%n%n", p2

    ; k-rate stuff
    if (kCurrentTime >= kNextStart) then ; write a new note!

        kndx = (kCurrentTime-p2)/p3

        ; get the required pfield ranges
        kInstMin tablei    kndx, iInstrMinTable
        kInstMax tablei    kndx, iInstrMaxTable
        kDensMin tablei    kndx, iDensityMinTable
        kDensMax tablei    kndx, iDensityMaxTable
        kDurMin tablei    kndx, iDurMinTable
        kDurMax tablei    kndx, iDurMaxTable
        kAmpMin tablei    kndx, iAmpMinTable
        kAmpMax tablei    kndx, iAmpMaxTable

        ; find random values for all our required parametrs and print the i-statement
        fprintks gScoName, "i%d%t%f%t%f%t%f", kInstMin + rnd(kInstMax-kInstMin), kNextStart, kDurMin +

        ; now any additional pfields
        ProcessAdditionalPfields p4, kndx, iNumPfields
```

```
        PrintReturn

        ; calculate next starttime
        kDensity = kDensMin + rnd(kDensMax-kDensMin)
        if (kDensity < iResolution) then
            kDensity = iResolution
        endif
        kNextStart = kNextStart + kDensity
    endif

    kCurrentTime = kCurrentTime + iResolution
endin


</CsInstruments>
<CsScore>
/*
===========================================================
  scogen.sco

    this csound module generates a score file
    you specify a gesture of notes by giving
    the "gesture" instrument a number to a
    (negative) gen2 table.

    this table stores numbers to pairs of functions.
    each function-pair represents a range (min-max)
    of randomness for every pfield for the notes to
    be generated.
===========================================================
*/


; common tables for pfield ranges
f100    0    2    -7    0 2 0    ; static 0
f101    0    2    -7    1 2 1    ; static 1
f102    0    2    -7    0 2 1 ; ramp 0->1
f103    0    2    -7    1 2 0 ; ramp 1->0
f105    0    2    -7    10 2 10 ; static 10
f106    0    2    -7    .1 2 .1 ; static .1

; specific pfield ranges
f10    0    2    -7       .8 2 .01 ; density
f11    0    2    -7       8 2 4 ; pitchmin
f12    0    2    -7       8 2 12 ; pitchmax


;=== table containing the function numbers used for all the p-field distributions
;
;    p1 -     table number
;    p2 -     time table is instantiated
;    p3 -     size of table (must be >= p5!)
;    p4 -     gen# (should be = -2)
;    p5 -     number of pfields of each note to be generated
;    p6 -     table number of the function representing the minimum possible note number (p1) of a gene
;    p7 -     table number of the function representing the maximum possible note number (p1) of a gene
;    p8 -     table number of the function representing the minimum possible noteon-to-noteon time (p2
;    p9 -     table number of the function representing the maximum possible noteon-to-noteon time (p2
;    p10 -    table number of the function representing the minimum possible duration (p3) of a generat
;    p11 -    table number of the function representing the maximum possible duration (p3) of a generat
;    p12 -    table number of the function representing the maximum possible amplitude (p4) of a genera
;    p13 -    table number of the function representing the maximum possible amplitude (p5) of a genera
;    p14,p16.. -    table number of the function representing the minimum possible value for additional
;    p15,p17.. -    table number of the function representing the maximum possible value for additional

;        siz    2    #pds p1min    p1max p2min    p2max p3min    p3max p4min    p4max p5min    p5max p6
f1    0    32    -2    6    101    101    10    10 101    105    100    106    11    12    100    101


;gesture definitions
;        start    dur    pTble    scale    overlap seed
i"Gesture"     0    60    1 ;todo-->0    0    123
</CsScore>
</CsoundSynthesizer>
```

This example will generate a file called « schottstaedt.sco » which can be used as a score together with *schottstaedt.orc* [examples/schottstaedt.orc]

See Also

printks

Credits

Author: Matt Ingalls
January 2003

fprints

fprints — Similar to prints but prints to a file.

Description

Similar to *prints* but prints to a file.

Syntax

```
fprints "filename", "string" [, ival1] [, ival2] [...]
```

Initialization

"filename" -- name of the output file.

"string" -- the text string to be printed. Can be up to 8192 characters and must be in double quotes.

ival1, ival2, ... (optional) -- The i-rate values to be printed. These are specified in « *string* » with the standard C value specifier (%f, %d, etc.) in the order given.

Performance

fprints is similar to the *prints* opcode except it outputs to a file. For more information about output formatting, please look at *printks's documentation*.

Examples

Here is an example of the fprints opcode. It uses the file *fprints.csd* [examples/fprints.csd].

Exemple 205. Example of the fprints opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fprints.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a score generator example.
instr 1
  ; Print to the file "my.sco".
  fprints "my.sco", "%!Generated score by ma++\\n \\n"
```

```
endin

</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001


</CsScore>
</CsoundSynthesizer>
```

This example will generate a file called « my.sco ». It should contain a line like this:

```
;Generated score by ma++
```

See Also

prints

Credits

Author: Matt Ingalls
January 2003

frac

frac — Retourne la partie fractionnaire d'un nombre décimal.

Description

Retourne la partie fractionnaire de *x*.

Syntaxe

```
frac(x) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode frac. Il utilise le fichier *frac.csd* [examples/frac.csd].

Exemple 206. Exemple de l'opcode frac.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o frac.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 16 / 5
  i2 = frac(i1)

  print i2
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i2 = 0.200
```

Voir Aussi

abs, *exp*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Exemple écrit par Kevin Conder.

freeverb

freeverb — Opcode version of Jezar's Freeverb

Description

freeverb is a stereo reverb unit based on Jezar's public domain C++ sources, composed of eight parallel comb filters on both channels, followed by four allpass units in series. The filters on the right channel are slightly detuned compared to the left channel in order to create a stereo effect.

Syntax

```
aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]
```

Initialization

iSRate (optional, defaults to 44100): adjusts the reverb parameters for use with the specified sample rate (this will affect the length of the delay lines in samples, and, as of the latest CVS version, the high frequency attenuation). Only integer multiples of 44100 will reproduce the original character of the reverb exactly, so it may be useful to set this to 44100 or 88200 for an orchestra sample rate of 48000 or 96000 Hz, respectively. While iSRate is normally expected to be close to the orchestra sample rate, different settings may be useful for special effects.

iSkip (optional, defaults to zero): if non-zero, initialization of the opcode will be skipped, whenever possible.

Performance

ainL, ainR -- input signals; usually both are the same, but different inputs can be used for special effect

Note

It is recommended to process the input signal(s) with the denorm opcode in order to avoid denormalized numbers which could significantly increase CPU usage in some cases

aoutL, aoutR -- output signals for left and right channel

kRoomSize (range: 0 to 1) -- controls the length of the reverb, a higher value means longer reverb. Settings above 1 may make the opcode unstable.

kHFDamp (range: 0 to 1): high frequency attenuation; a value of zero means all frequencies decay at the same rate, while higher settings will result in a faster decay of the high frequency range.

Examples

Here is an example of the *freeverb* opcode. It uses the file *freeverb.csd* [examples/freeverb.csd].

Exemple 207. An example of the freeverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o freeverb.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr      =  44100
ksmps   =  32
nchnls  =  2
0dbfs   =  1

        instr 1
a1      vco2 0.75, 440, 10
kfrq    port 100, 0.008, 20000
a1      butterlp a1, kfrq
a2      linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1      =  a1 * a2
        denorm a1
aL, aR  freeverb a1, a1, 0.9, 0.35, sr, 0
        outs a1 + aL, a1 + aR
        endin

</CsInstruments>
<CsScore>
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Istvan Varga
2005

ftchnls

ftchnls — Returns the number of channels in a stored function table.

Description

Returns the number of channels in a stored function table.

Syntax

```
ftchnls(x) (init-rate args only)
```

Performance

Returns the number of channels of a *GEN01* table, determined from the header of the original file. If the original file has no header or the table was not created by these GEN01, *ftchnls* returns -1.

Examples

Here is an example of the ftchnls opcode. It uses the file *ftchnls.csd* [examples/ftchnls.csd], and *mary.wav* [examples/mary.wav].

Exemple 208. Example of the ftchnls opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftchnls.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the number of channels in Table #1.
  ichnls = ftchnls(1)
  print ichnls
endin


</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
```

```
e

</CsScore>
</CsoundSynthesizer>
```

Since the audio file « mary.wav » is monophonic (1 channel), its output should include a line like this:

```
instr 1:  ichnls = 1.000
```

See Also

ftlen, *ftlptim*, *ftsr*, *nsamp*

Credits

Author: Chris McCormick
Perth, Australia
December 2001

Example written by Kevin Conder.

ftconv

ftconv — Low latency multichannel convolution, using a function table as impulse response source.

Description

Low latency multichannel convolution, using a function table as impulse response source. The algorithm is to split the impulse response to partitions of length determined by the 'iplen' parameter, and delay and mix partitions so that the original, full length impulse response is reconstructed without gaps. The output delay (latency) is 'iplen' samples, and does not depend on the control rate, unlike in the case of other convolve opcodes.

Syntax

```
a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskipsamples \
      [, iirlen[, iskipinit]]]
```

Initialization

ift -- source ftable number. The table is expected to contain interleaved multichannel audio data, with the number of channels equal to the number of output variables (a1, a2, etc.). An interleaved table can be created from a set of mono tables with GEN52.

iplen -- length of impulse response partitions, in sample frames; must be an integer power of two. Lower settings allow for shorter output delay, but will increase CPU usage.

iskipsamples (optional, defaults to zero) -- number of sample frames to skip at the beginning of the table. Useful for reverb responses that have some amount of initial delay. If this delay is not less than 'iplen' samples, then setting iskipsamples to the same value as iplen will eliminate any additional latency by ftconv.

iirlen (optional) -- total length of impulse response, in sample frames. The default is to use all table data (not including the guard point).

iskipinit (optional, defaults to zero) -- if set to any non-zero value, skip initialization whenever possible without causing an error.

Performance

ain -- input signal.

a1 ... a8 -- output signal(s).

Example

Here is an example of the ftconv opcode. It uses the file *ftconv.csd* [examples/ftconv.csd].

Exemple 209. Example of the ftconv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftconv.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr      =  48000
ksmps   =  32
nchnls  =  2
0dbfs   =  1

garvb   init 0
gaW     init 0
gaX     init 0
gaY     init 0

itmp    ftgen   1, 0, 64, -2, 2, 40, -1, -1, -1, 123,           \
               1, 13.000, 0.05, 0.85, 20000.0, 0.0, 0.50, 2,   \
               1,  2.000, 0.05, 0.85, 20000.0, 0.0, 0.25, 2,   \
               1, 16.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,   \
               1,  9.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,   \
               1, 12.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,   \
               1,  8.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2

itmp    ftgen 2, 0, 262144, -2, 0
       spat3dt 2, -0.2, 1, 0, 1, 1, 2, 0.005

itmp    ftgen 3, 0, 262144, -52, 3, 2, 0, 4, 2, 1, 4, 2, 2, 4

       instr 1

a1      vco2 1, 440, 10
kfrq    port 100, 0.008, 20000
a1      butterlp a1, kfrq
a2      linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1      =  a1 * a2 * 2
       denorm a1
       vincr garvb, a1
aw, ax, ay, az  spat3di a1, p4, p5, p6, 1, 1, 2
       vincr gaW, aw
       vincr gaX, ax
       vincr gaY, ay

       endin

       instr 2

       denorm garvb
; skip as many samples as possible without truncating the IR
arW, arX, arY   ftconv garvb, 3, 2048, 2048, (65536 - 2048)
aW      =  gaW + arW
aX      =  gaX + arX
aY      =  gaY + arY
garvb   =  0
gaW     =  0
gaX     =  0
gaY     =  0

aWre, aWim      hilbert aW
aXre, aXim      hilbert aX
aYre, aYim      hilbert aY
aWXr    =  0.0928*aXre + 0.4699*aWre
aWXiYr  =  0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aL      =  aWXr + aWXiYr
aR      =  aWXr - aWXiYr

       outs aL, aR

       endin

</CsInstruments>
<CsScore>

i 1 0 0.5  0.0  2.0 -0.8
i 1 1 0.5  1.4  1.4 -0.6
i 1 2 0.5  2.0  0.0 -0.4
```

```
i 1 3 0.5  1.4 -1.4 -0.2
i 1 4 0.5  0.0 -2.0  0.0
i 1 5 0.5 -1.4 -1.4  0.2
i 1 6 0.5 -2.0  0.0  0.4
i 1 7 0.5 -1.4  1.4  0.6
i 1 8 0.5  0.0  2.0  0.8
i 2 0 10
e

</CsScore>
</CsoundSynthesizer>
```

See also

pconvolve, *convolve*, *convolve*.

Credits

Author: Istvan Varga
2005

ftfree

ftfree — Deletes function table.

Description

Deletes function table.

Syntax

```
ftfree ifno, iwhen
```

Initialization

ifno -- the number of the table to be deleted

iwhen -- if zero the table is deleted at init time; otherwise the table number is registered for being deleted at note deactivation.

Credits

Authors: Steven Yi, Istvan Varga
2005

ftgen

ftgen — Generate a score function table from within the orchestra.

Description

Generate a score function table from within the orchestra.

Syntax

```
gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]
```

Initialization

gir -- either a requested or automatically assigned table number above 100.

ifn -- requested table number If *ifn* is zero, the number is assigned automatically and the value placed in *gir*. Any other value is used as the table number

itime -- is ignored, but otherwise corresponds to p2 in the score *f statement*.

isize -- table size. Corresponds to p3 of the score *f statement*.

igen -- function table *GEN* routine. Corresponds to p4 of the score *f statement*.

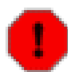
iarga, iargb, ... -- function table arguments. Correspond to p5 through p*n* of the score *f statement*.

Performance

This is equivalent to table generation in the score with the *f statement*.

Note

Csound was originally designed to support tables with power of two sizes only. Though this has changed in recent versions (you can use any size by using a negative number), many opcodes will not accept them.



Warning

Although Csound will not protest if ftgen is used inside instr-endin statements, this is not the intended or supported use, and must be handled with care as it has global effects. (In particular, a different size usually leads to relocation of the table, which may cause a crash or otherwise erratic behaviour.

Examples

Here is an example of the ftgen opcode. It uses the file *ftgen.csd* [examples/ftgen.csd].

Exemple 210. Example of the ftgen opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftgen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a sine wave using the GEN10 routine.
gitemp ftgen 1, 0, 16384, 10, 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ; Use Table #1.
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Here is another example of the ftgen opcode. It uses the file *ftgen-2.csd* [examples/ftgen-2.csd].

Exemple 211. Example of the ftgen opcode.

This example queries a file for it length to create an f-table of the appropriate size.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftgen-2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

  sr     =  48000
  ksmps     =  16
  nchnls    =  2

;Example by Jonathan Murphy 2007

  0dbfs     =  1

    instr 1
```

```
  Sfile     =    "beats.wav"

  ilen     filelen   Sfile  ; Find length
  isr     filesr    Sfile  ; Find sample rate

  isamps    =  ilen * isr  ; Total number of samples
  isize     init      1

loop:
  isize     =  isize * 2
; Loop until isize is greater than number of samples
if (isize < isamps) igoto loop

  itab     ftgen     0, 0, isize, 1, Sfile, 0, 0, 0
            print     isize
            print     isamps

  turnoff
    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

See also

GEN routine overview

Credits

Author: Barry L. Vercoe
M.I.T., Cambridge, Mass
1997

Example written by Kevin Conder.

Added warning April 2002 by Rasmus Ekman

ftgentmp

ftgentmp — Generate a score function table from within the orchestra, which is deleted at the end of the note.

Description

Generate a score function table from within the orchestra, which is optionally deleted at the end of the note.

Syntax

```
ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...
```

Initialization

ifno -- either a requested or automatically assigned table number above 100.

ip1 -- the number of the table to be generated or 0 if the number is to be assigned, in which case the table is deleted at the end of the note activation.

ip2dummy -- ignored.

isize -- table size. Corresponds to p3 of the score *f statement*.

igen -- function table *GEN* routine. Corresponds to p4 of the score *f statement*.

iarga, iargb, ... -- function table arguments. Correspond to p5 through p*n* of the score *f statement*.

Note

Csound was originally designed to support tables with power of two sizes only. Though this has changed in recent versions (you can use any size by using a negative number), many opcodes will not accept them.

Credits

Authors: Istvan Varga
2005

ftlen

ftlen — Returns the size of a stored function table.

Description

Returns the size of a stored function table.

Syntax

```
ftlen(x) (init-rate args only)
```

Performance

Returns the size (number of points, excluding guard point) of stored function table, number *x*. While most units referencing a stored table will automatically take its size into account (so tables can be of arbitrary length), this function reports the actual size if that is needed. Note that *ftlen* will always return a power-of-2 value, i.e. the function table guard point (see *f Statement*) is not included.As of Csound version 3.53, *ftlen* works with deferred function tables (see *GEN01*).

ftlen differs from *nsamp* in that *nsamp* gives the number of sample frames loaded, while *ftlen* gives the total number of samples without the guard point. For example, with a stereo sound file of 10000 samples, ftlen() would return 19999 (i.e. a total of 20000 mono samples, not including a guard point), but nsamp() returns 10000.

Examples

Here is an example of the ftlen opcode. It uses the file *ftlen.csd* [examples/ftlen.csd], and *mary.wav* [examples/mary.wav].

Exemple 212. Example of the ftlen opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftlen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the size of Table #1.
  ; The size will be the number of points excluding the guard point.
  ilen = ftlen(1)
```

```
  print ilen
endin


</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The audio file « mary.wav » is 154390 samples long. The ftlen opcode reports it as 154389 samples long because it reserves 1 point for the guard point. Its output should include a line like this:

```
instr 1:  ilen = 154389.000
```

See Also

ftchnls, *ftlptim*, *ftsr*, *nsamp*

Credits

Author: Barry L. Vercoe
MIT
Cambridge, Massachussetts
1997

Example written by Kevin Conder.

ftload

ftload — Load a set of previously-allocated tables from a file.

Description

Load a set of previously-allocated tables from a file.

Syntax

```
ftload "filename", iflag, ifn1 [, ifn2] [...]
```

Initialization

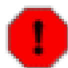
"filename" -- A quoted string containing the name of the file to load.

iflag -- Type of the file to load/save. (0 = binary file, Non-zero = text file)

ifn1, ifn2, ... -- Numbers of tables to load.

Performance

ftload loads a list of tables from a file. (The tables have to be already allocated though.) The file's format can be binary or text.



Warning

The file's format is not compatible with a WAV-file and is not endian-safe.

Examples

See the example for *ftsave*.

See Also

ftloadk, *ftsavek*, *ftsave*

Credits

Author: Gabriel Maldonado

New in version 4.21

ftloadk

ftloadk — Load a set of previously-allocated tables from a file.

Description

Load a set of previously-allocated tables from a file.

Syntax

```
ftloadk "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

Initialization

"filename" -- A quoted string containing the name of the file to load.

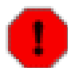
iflag -- Type of the file to load/save. (0 = binary file, Non-zero = text file)

ifn1, ifn2, ... -- Numbers of tables to load.

Performance

ktrig -- The trigger signal. Load the file each time it is non-zero.

ftloadk loads a list of tables from a file. (The tables have to be already allocated though.) The file's format can be binary or text. Unlike *ftload*, the loading operation can be repeated numerous times within the same note by using a trigger signal.



Warning

The file's format is not compatible with a WAV-file and is not endian-safe.

See Also

ftload, *ftsavek*, *ftsave*

Credits

Author: Gabriel Maldonado

New in version 4.21

ftlptim

ftlptim — Returns the loop segment start-time of a stored function table number.

Description

Returns the loop segment start-time of a stored function table number.

Syntax

```
ftlptim(x) (init-rate args only)
```

Performance

Returns the loop segment start-time (in seconds) of stored function table number *x*. This reports the duration of the direct recorded attack and decay parts of a sound sample, prior to its looped segment. Returns zero (and a warning message) if the sample does not contain loop points.

Examples

Here is an example of the ftlptim opcode. It uses the file *ftlptim.csd* [examples/ftlptim.csd], and *mary.wav* [examples/mary.wav].

Exemple 213. Example of the ftlptim opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftlptim.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the loop-segment start time in Table #1.
  itim = ftlptim(1)
  print itim
endin


</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
```

```
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Since the audio file « mary.wav » is non-looping, its output should include lines like this:

```
WARNING: non-looping sample
instr 1:  itim = 0.000
```

See Also

ftchnls, *ftlen*, *ftsr*, *nsamp*

Credits

Author: Barry L. Vercoe
MIT
Cambridge, Massachussetts
1997

Example written by Kevin Conder.

ftmorf

ftmorf — Morphs between multiple ftables as specified in a list.

Description

Uses an index into a table of ftable numbers to morph between adjacent tables in the list.This morphed function is written into the table referenced by *iresfn* on every k-cycle.

Syntax

```
ftmorf kftndx, iftfn, iresfn
```

Initialization

iftfn -- The ftable function. The list of values are expected to be pre-existing ftable numbers.

iresfn -- Table number of the morphed function

The length of all the tables in *iftfn* must equal the length of *iresfn*.

Performance

kftndx -- the index into the *iftfn* table.

If *iftfn* contains (6, 4, 6, 8, 7, 4):

- *kftndx=4* will write the contents of f7 into *iresfn*.
- *kftndx=4.5* will write the average of the contents of f7 and f4 into *iresfn*.

Examples

Here is an example of the ftmorf opcode. It uses the file *ftmorf.csd* [examples/ftmorf.csd].

Exemple 214. Example of the ftmorf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftmorf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

instr 1
kndx    line    0, p3, 7
        ftmorf  kndx, 1, 2
asig    oscili  30000, 440, 2
        out     asig
endin


</CsInstruments>
<CsScore>

f1 0 8 -2 3 4 5 6 7 8 9 10
f2 0 1024 10 1 /*contents of f2 dont matter */
f3 0 1024 10 1
f4 0 1024 10 0 1
f5 0 1024 10 0 0 1
f6 0 1024 10 0 0 0 1
f7 0 1024 10 0 0 0 0 1
f8 0 1024 10 0 0 0 0 0 1
f9 0 1024 10 0 0 0 0 0 0 1
f10 0 1024 10 1 1 1 1 1 1 1

i1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: William « Pete » Moss
University of Texas at Austin
Austin, Texas USA
Jan. 2002

New in version 4.18

ftsave

ftsave — Save a set of previously-allocated tables to a file.

Description

Save a set of previously-allocated tables to a file.

Syntax

```
ftsave "filename", iflag, ifn1 [, ifn2] [...]
```

Initialization

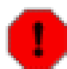
"filename" -- A quoted string containing the name of the file to save.

iflag -- Type of the file to save. (0 = binary file, Non-zero = text file)

ifn1, ifn2, ... -- Numbers of tables to save.

Performance

ftsave saves a list of tables to a file. The file's format can be binary or text.



Warning

The file's format is not compatible with a WAV-file and is not endian-safe.

Examples

Here is an example of the ftsave opcode. It uses the file *ftsave.csd* [examples/ftsave.csd].

Exemple 215. Example of the ftsave opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftsave.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Table #1, make a sine wave using the GEN10 routine.
gitmp1 ftgen 1, 0, 32768, 10, 1
; Table #2, create an empty table.
gitmp2 ftgen 2, 0, 32768, 7, 0, 32768, 0

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 20000
  kcps = 440
  ; Use Table #1.
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin


; Instrument #2 - Load Table #1 into Table #2.
instr 2
  ; Save Table #1 to a file called "table1.ftsave".
  ftsave "table1.ftsave", 0, 1

  ; Load the "table1.ftsave" file into Table #2.
  ftload "table1.ftsave", 0, 2

  kamp = 20000
  kcps = 440
  ; Use Table #2, it should contain Table #1's sine wave now.
  ifn = 2

  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
; Play Instrument #2 for 1 second.
i 2 2 1
e


</CsScore>
</CsoundSynthesizer>
```

See Also

ftloadk, *ftload*, *ftsavek*

Credits

Author: Gabriel Maldonado

Example written by Kevin Conder.

New in version 4.21

ftsavek

ftsavek — Save a set of previously-allocated tables to a file.

Description

Save a set of previously-allocated tables to a file.

Syntax

```
ftsavek "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

Initialization

"filename" -- A quoted string containing the name of the file to save.

iflag -- Type of the file to save. (0 = binary file, Non-zero = text file)

ifn1, ifn2, ... -- Numbers of tables to save.

Performance

ktrig -- The trigger signal. Save the file each time it is non-zero.

ftsavek saves a list of tables to a file. The file's format can be binary or text. Unlike *ftsave*, the saving operation can be repeated numerous times within the same note by using a trigger signal.

Warning

The file's format is not compatible with a WAV-file and is not endian-safe.

See Also

ftloadk, *ftload*, *ftsave*

Credits

Author: Gabriel Maldonado

New in version 4.21

ftsr

ftsr — Returns the sampling-rate of a stored function table.

Description

Returns the sampling-rate of a stored function table.

Syntax

```
ftsr(x) (init-rate args only)
```

Performance

Returns the sampling-rate of a *GEN01* generated table. The sampling-rate is determined from the header of the original file. If the original file has no header or the table was not created by these GEN01, *ftsr* returns 0. New in Csound version 3.49.

Examples

Here is an example of the ftsr opcode. It uses the file *ftsr.csd* [examples/ftsr.csd], and *mary.wav* [examples/mary.wav].

Exemple 216. Example of the ftsr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftsr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the sampling rate of Table #1.
  isr = ftsr(1)
  print isr
endin


</CsInstruments>
<CsScore>

; Table #1: Use an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
```

```
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Since the audio file « mary.wav » uses a 44.1 Khz sampling rate, its output should a line like this:

```
instr 1:  isr = 44100.000
```

See Also

ftchnls, *ftlen*, *ftlptim*, *nsamp*

Credits

Author: Gabriel Maldonado
Italy
October 1998

Example written by Kevin Conder.

gain

gain — Adjusts the amplitude audio signal according to a root-mean-square value.

Description

Adjusts the amplitude audio signal according to a root-mean-square value.

Syntax

```
ares gain asig, krms [, ihp] [, iskip]
```

Initialization

ihp (optional, default=10) -- half-power point (in Hz) of a special internal low-pass filter. The default value is 10.

iskip (optional, default=0) -- initial disposition of internal data space (see *reson*). The default value is 0.

Performance

asig -- input audio signal

gain provides an amplitude modification of *asig* so that the output *ares* has rms power equal to *krms*. *rms* and *gain* used together (and given matching *ihp* values) will provide the same effect as *balance*.

Examples

```
asrc buzz    10000,440, sr/440, 1 ; band-limited pulse train
a1   reson   asrc, 1000,100       ; sent through
a2   reson   a1,3000,500          ; 2 filters
afin balance a2, asrc             ; then balanced with source
```

See Also

balance, *rms*

gainslider

gainslider — Une implémentation de courbe de gain logarithmique qui est semblable à l'objet gainslider~ de Cycling 74 Max / MSP.

Description

Cet opcode sert à être multiplié par un signal audio pour donner la même impression qu'avec une console de mixage. Il n'y a pas de limites dans le code source si bien que l'on peut par exemple donner des valeurs supérieures à 127 pour obtenir un signal audio plus fort mais avec un risque d'écrêtage.

Syntaxe

```
kout gainslider kindex
```

Exécution

kindex -- Valeur d'indice. Intervalle nominal de 0 à 127. Par exemple un intervalle de 0 à 152 donnera un intervalle de -# à +18,0 dB.

kout -- Sortie pondérée.

Exemples

Voici un exemple de l'opcode gainslider. Il utilise le fichier *gainslider.csd* [examples/gainslider.csd].

Exemple 217. Exemple de l'opcode gainslider.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent
-odac          -idac      -d     ;;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

/*--- ---*/

                instr 1 ; gainslider test

; uncomment for realtime midi
;kmod ctrl7 1, 1, 0, 127

; uncomment for non realtime
km0d phasor 1/10
kmod scale km0d, 127, 0

kout gainslider kmod

        printk2 kmod
        printk2 kout
```

```
aout diskin "soundfile.aiff", 1, 0, 1

aout = aout*kout

        outs aout, aout

                endin

/*--- ---*/
</CsInstruments>
<CsScore>

i1 0 8888

e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

scale, *logcurve*, *expcurve*

Crédits

Auteur : David Akbari
Octobre
2006

gauss

gauss — Générateur de nombres aléatoires de distribution gaussienne.

Description

Générateur de nombres aléatoires de distribution gaussienne. C'est un générateur de bruit de classe x.

Syntaxe

```
ares gauss krange
```

```
ires gauss krange
```

```
kres gauss krange
```

Exécution

krange -- l'intervalle des nombres aléatoires (-*krange* à +*krange*). Produit des nombres positifs et négatifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode gauss. Il utilise le fichier *gauss.csd* [examples/gauss.csd].

Exemple 218. Exemple de l'opcode gauss.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gauss.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number between -1 and 1.
  ; krange = 1

  i1 gauss 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.252
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

Existait dans la version 3.30

gbuzz

gbuzz — La sortie est un ensemble de partiels cosinus en relation harmonique.

Description

La sortie est un ensemble de partiels cosinus en relation harmonique.

Syntaxe

```
ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]
```

Initialisation

ifn -- numéro de table d'une fonction stockée contenant une onde cosinus. Une grande table d'au moins 8192 points est recommandée.

iphs (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

Exécution

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

knh -- nombre total d'harmoniques demandés. Si *knh* est négatif, sa valeur absolue est utilisée. Si *knh* vaut zéro, une valeur de 1 est utilisée.

klh -- harmonique présent le plus bas. Peut être positif, nul ou négatif. Dans *gbuzz* l'ensemble de partiels peut commencer à n'importe quel numéro de partiel et se complète vers le haut ; si *klh* est négatif, tous les partiels en-dessous de zéro seront repliés comme des partiels positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux partiels positifs de l'ensemble.

kmul -- spécifie la raison de la série des coefficients d'amplitude. C'est une série entière : si le *klh*ème partiel a pour coefficient A, le (*klh* + n)ème partiel aura pour coefficient A * (*kmul* ** n), c'est-à-dire que les valeurs d'intensité dessinent une courbe exponentielle. *kmul* peut être positif, nul ou négatif, et n'est pas restreint aux valeurs entières.

buzz et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel *klh* = *kmul* = 1 ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire *knh* = int (sr / 2 / fréq. fondamentale), le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* et *klh* durant l'exécution, leurs valeurs internes sont nécessairement entières ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. Cependant, la variation de *kmul* durant l'exécution produit un bon effet. *gbuzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GEN11*, dans lequel le même ensemble de cosinus peut être

stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul, le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

Exemples

Voici un exemple de l'opcode gbuzz. Il utilise le fichier *gbuzz.csd* [examples/gbuzz.csd].

Exemple 219. Exemple de l'opcode gbuzz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gbuzz.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  knh = 3
  klh = 2
  kmul = 0.7
  ifn = 1

  a1 gbuzz kamp, kcps, knh, klh, kmul, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a simple cosine waveform.
f 1 0 16384 11 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

buzz

Crédits

Exemple écrit par Kevin Conder.

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre *kmul*.

getcfg

getcfg — Return Csound settings.

Description

Return various configuration settings in Svalue as a string at init time.

Syntax

```
Svalue getcfg iopt
```

Initialization

iopt -- The parameter to be returned, can be one of:

- 1: the maximum length of string variables in characters; this is at least the value of the -+max_str_len command line option - 1
- 2: the input sound file name (-i), or empty if there is no input file
- 3: the output sound file name (-o), or empty if there is no output file
- 4: return "1" if real time audio input or output is being used, and "0" otherwise
- 5: return "1" if running in beat mode (-t command line option), and "0" otherwise
- 6: the host operating system name
- 7: return "1" if a callback function for the chnrecv and chnsend opcodes has been set, and "0" otherwise (which means these opcodes do nothing)

Credits

Author: Istvan Varga
2006

New in version 5.02

gogobel

gogobel — La sortie audio est un son tel que celui produit lorque l'on frappe une cloche à vache.

Description

La sortie audio est un son tel que celui produit lorque l'on frappe une cloche à vache. Il s'agit d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

Syntaxe

```
ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn
```

Initialisation

ihrd -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

ipos -- le point d'impact sur le bloc, compris entre 0 et 1.

imp -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

ivfn -- forme du vibrato, habituellement une table sinus, créée par une fonction.

Exécution

Une note est jouée sur une instrument de type cloche à vache, avec les arguments suivants.

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note.

kvibf -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de de 0 à 12.

kvamp -- Amplitude du vibrato.

Exemples

Voici un exemple de l'opcode gogobel. Il utilise les fichiers *gogobel.csd* [examples/gogobel.csd] et *marmstk1.wav* [examples/marmstk1.wav]

Exemple 220. Exemple de l'opcode gogobel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gogobel.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; kamp = 31129.60
  ; kfreq = 440
  ; ihrd = 0.5
  ; ipos = 0.561
  ; imp = 1
  ; kvibf = 6.0
  ; kvamp = 0.3
  ; ivfn = 2

  a1 gogobel 31129.60, 440, 0.5, 0.561, 1, 6.0, 0.3, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, the "marmstk1.wav" audio file.
f 1 0 256 1 "marmstk1.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 3.47 de Csound

goto

goto — Transfer control on every pass.

Description

Transfer control to *label* on every pass. (Combination of *igoto* and *kgoto*)

Syntax

```
goto label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the goto opcode. It uses the file *goto.csd* [examples/goto.csd].

Exemple 221. Example of the goto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o goto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  a1 oscil 10000, 440, 1
  goto playit

  ; The goto will go to the playit label.
  ; It will skip any code in between like this comment.

playit:
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
```

```
e

</CsScore>
</CsoundSynthesizer>
```

See Also

cggoto, *cigoto*, *ckgoto*, *if*, *igoto*, *kgoto*, *tigoto*, *timout*

Credits

Example written by Kevin Conder.

Added a note by Jim Aikin.

grain

grain — Génère des textures de synthèse granulaire.

Description

Génère des textures de synthèse granulaire.

Syntaxe

```
ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \
      iwfn, imgdur [, igrnd]
```

Initialisation

igfn -- numéro de la ftable de la forme d'onde du grain. Peut être une onde sinus ou un son échantillonné.

iwfn -- numéro de la ftable de l'enveloppe d'amplitude utilisée pour les grains (voir aussi *GEN20*).

imgdur -- durée maximum du grain en secondes. C'est la plus grande valeur que l'on peut affecter à *kgdur*.

igrnd (facultatif) -- s'il est non nul, le décalage aléatoire du grain est désactivé. Cela signifie que tous les grains commenceront à lire la table *igfn* depuis son début. S'il vaut zéro (par défaut), les grains commenceront leur lecture dans la table *igfn* à partir de positions aléatoires.

Exécution

xamp -- amplitude de chaque grain.

xpitch -- hauteur du grain. Pour utiliser la fréquence originale du son en entrée, on se sert de la formule :

sndsr / ftlen(*igfn*)

où sndsr est le taux d'échantillonnage original du son *igfn*.

xdens -- densité des grains mesurée en grains par seconde. Si elle est constante la sortie sera une synthèse granulaire synchrone, très semblable à *fof*. Si *xdens* a une composante aléatoire (comme du bruit ajouté), alors le résultat ressemblera plus à une synthèse granulaire asynchrone.

kampoff -- déviation d'amplitude maximale par rapport à *xamp*. Cela signifie que l'amplitude maximale possible pour un grain est *xamp* + *kampoff* et l'amplitude minimale est *xamp*. Si *kampoff* est nul alors il n'y a pas d'amplitude aléatoire pour chaque grain.

kpitchoff -- déviation de hauteur maximale par rapport à *xpitch* en Hz. Semblable à *kampoff*.

kgdur -- durée du grain en secondes. Sa valeur maximale doit être déclarée dans *imgdur*. Si *kgdur* dépasse *imgdur* en un point, sa valeur sera tronquée à celle de *imgdur*.

Le générateur *grain* est principalement basé sur les travaux et les écrits de Barry Truax et de Curtis Roads.

Exemples

Cet exemple génère une texture avec des grains de plus en plus courts, une amplitude de plus en plus large et une dispersion de hauteur. Il utilise les fichiers *grain.csd* [examples/grain.csd] et *mary.wav* [examples/mary.wav].

Exemple 222. Exemple de l'opcode grain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
    insnd = 10
    ibasfrq = 44100 / ftlen(insnd)   ; Use original sample rate of insnd file

    kamp   expseg 220, p3/2, 600, p3/2, 220
    kpitch line ibasfrq, p3, ibasfrq * .8
    kdens  line 600, p3, 200
    kaoff  line 0, p3, 5000
    kpoff  line 0, p3, ibasfrq * .5
    kgdur  line .4, p3, .1
    imaxgdur =  .5

    ar  grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, 5, imaxgdur, 0.0
    out ar
endin


</CsInstruments>
<CsScore>

f5  0 512  20 2                  ; Hanning window
f10 0 262144 1  "mary.wav" 0 0 0
i1 0 6
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Paris Smaragdis
MIT
Mai 1997

grain2

grain2 — Générateur de textures par synthèse granulaire facile à utiliser.

Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser, mais *grain3* offre plus de contrôle.

Syntaxe

```
ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \
      [, iseed] [, imode]
```

Initialisation

iovrlp -- (constant) nombre de grains se chevauchant.

iwfn -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

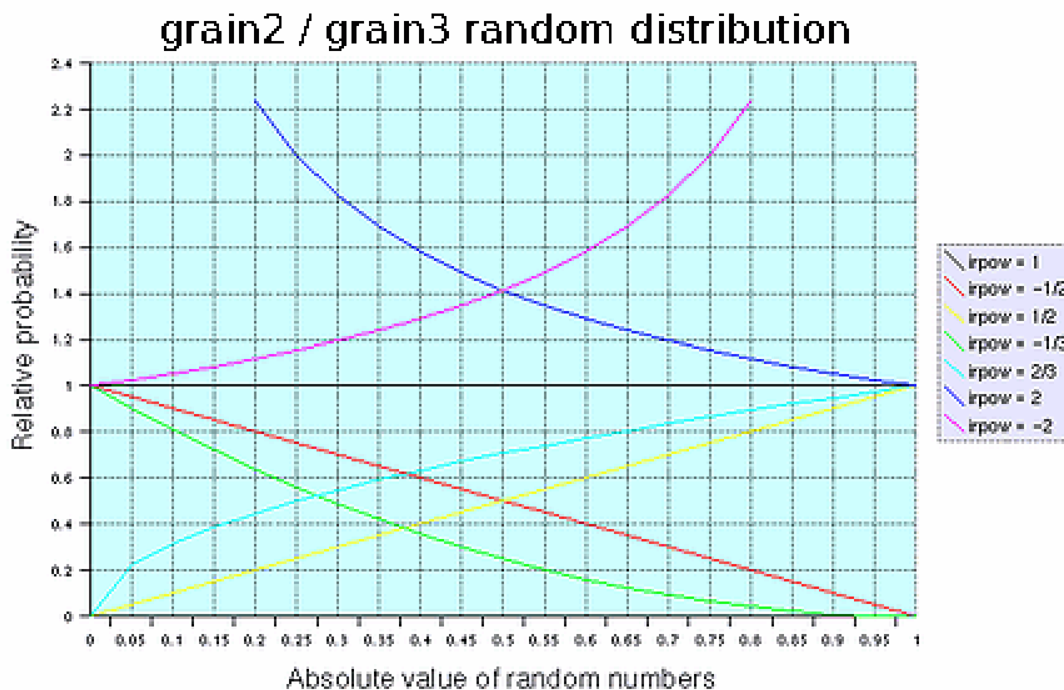
irpow (facultatif, par défaut 0) -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *irpow* est positif, la distribution aléatoire (*x* est compris entre -1 et 1) est

abs(x) ^ ((1 / irpow) - 1) ;

pour des valeurs négatives de *irpow*, elle est

(1 - abs(x)) ^ ((-1 / irpow) - 1)

En fixant *irpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *irpow*. La valeur par défaut de *irpow* est 0.



Un graphique des distributions pour différentes valeurs de *irpow*.

iseed (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 (2^{31} - 2)). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

imode (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- *8* : forme d'onde de la fenêtre avec interpolation (plus lent).
- *4* : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- *2* : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus.
- *1* : ignorer l'initialisation.

Exécution

ares -- signal de sortie.

kcps -- fréquence du grain en Hz.

kfmd -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

kgdur -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

kfn -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k

(on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).

Note

grain2 utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

Exemples

Voici un exemple de l'opcode grain2. Il utilise le fichier *grain2.csd* [examples/grain2.csd].

Exemple 223. Exemple de l'opcode grain2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr =  48000
kr =  750
ksmps =  64
nchnls =  2

/* square wave */
i_ ftgen 1, 0, 4096, 7, 1, 2048, 1, 0, -1, 2048, -1
/* window */
i_ ftgen 2, 0, 16384, 7, 0, 4096, 1, 4096, 0.3333, 8192, 0
/* sine wave */
i_ ftgen 3, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11,                 \
                            1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
                            1,  1.753, 0.05, 0.85,  5000, 0.8, 0.5, 2, \
                            1, 39.451, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 33.503, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 36.151, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 29.633, 0.05, 0.85,  7000, 0.8, 0.5, 2

ga01 init 0

/* generate bandlimited square waves */

i0 =  0
loop1:
imaxh =  sr / (2 * 440.0 * exp (log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 =  i0 + 1
        if (i0 < 127.5) igoto loop1

        instr 1

p3 =  p3 + 0.2

/* note velocity */
iamp =  0.0039 + p5 * p5 / 16192
/* vibrato */
kcps oscili 1, 8, 3
kenv linseg 0, 0.05, 0, 0.1, 1, 1, 1
/* frequency */
kcps =  (kcps * kenv * 0.01 + 1) * 440 * exp(log(2) * (p4 - 69) / 12)
```

```
/* grain ftable */
kfn =  int(256 + 69 + 0.5 + 12 * log(kcps / 440) / log(2))
/* grain duration */
kgdur port 100, 0.1, 20
kgdur =  kgdur / kcps

a1 grain2 kcps, kcps * 0.02, kgdur, 50, kfn, 2, -0.5, 22, 2
a1 butterlp a1, 3000
a2 grain2 kcps, kcps * 0.02, 4 / kcps, 50, kfn, 2, -0.5, 23, 2
a2 butterbp a2, 12000, 8000
a2 butterbp a2, 12000, 8000
aenv1 linseg 0, 0.01, 1, 1, 1
aenv2 linseg 3, 0.05, 1, 1, 1
aenv3 linseg 1, p3 - 0.2, 1, 0.07, 0, 1, 0

a1 =  aenv1 * aenv3 * (a1 + a2 * 0.7 * aenv2)

ga01 =  ga01 + a1 * 10000 * iamp

        endin

/* output instr */

        instr 81

i1 =  0.000001
aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, 3.0, 4.0, 0.0, 0.5, 7, 4
ga01 =  0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

        outs aLl + aLh, aRl + aRh

        endin


</CsInstruments>
<CsScore>

t 0 60

i 1 0.0 1.3 60 127
i 1 2.0 1.3 67 127
i 1 4.0 1.3 64 112
i 1 4.0 1.3 72 112

i 81 0 6.4

e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

grain3

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga

grain3

grain3 — Générateur de textures par synthèse granulaire avec plus de contrôle.

Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser mais *grain3* offre plus de contrôle.

Syntaxe

```
ares grain3 kcps, kphs, kfmd, kpmd, kgdur, kdens, imaxovr, kfn, iwfn, \
      kfrpow, kprpow [, iseed] [, imode]
```

Initialisation

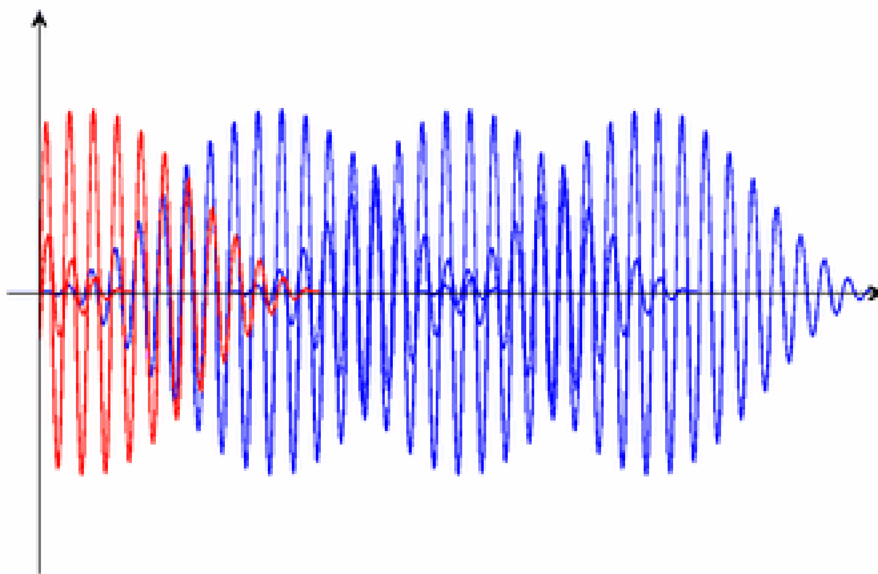
imaxovr -- nombre maximum de grains se chevauchant. Le nombre de chevauchements peut être calculé par (*kdens* * *kgdur*) ; cependant, il peut être surestimé sans coût supplémentaire lors de l'exécution, et un simple chevauchement utilise (selon le système) de 16 à 32 octets en mémoire.

iwfn -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

iseed (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 (2^{31} - 2)). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

imode (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- *64* : synchronise la phase au démarrage des grains sur *kcps*.
- *32* : démarre tous les grains sur une position d'échantillon entière. Ceci peut être plus rapide dans certains cas, tout en rendant moins précis le déroulement temporel des enveloppes de grain.
- *16* : ne pas générer de grains ayant une date de démarrage inférieure à zéro. (Voir la figure ci-dessous ; cette option désactive les grains marqués en rouge sur l'image).
- *8* : forme d'onde de la fenêtre avec interpolation (plus lent).
- *4* : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- *2* : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus. Contrôle aussi la modulation de phase (*kphs*)
- *1* : ignorer l'initialisation.



Graphique montrant des grains avec une date de démarrage inférieure à zéro en rouge.

Exécution

ares -- signal de sortie.

kcps -- fréquence du grain en Hz.

kphs -- phase du grain. C'est une position dans la forme d'onde du grain, exprimée comme une fraction (entre 0 et 1) de la longueur de la table.

kfmd -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

kpmd -- variation aléatoire (bipolaire) de la phase au démarrage.

kgdur -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

kdens -- nombre de grains par seconde.

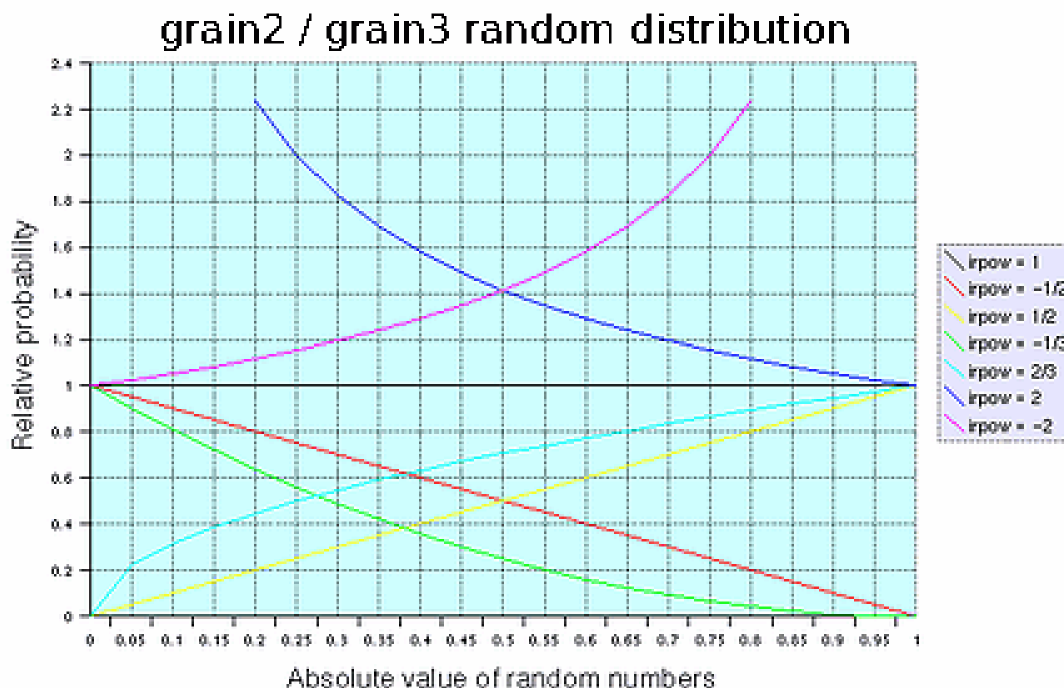
kfrpow -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *kfrpow* est positif, la distribution aléatoire (*x* est compris entre -1 et 1) est

abs(x) ^ ((1 / kfrpow) - 1) ;

pour des valeurs négatives de *kfrpow*, elle est

(1 - abs(x)) ^ ((-1 / kfrpow) - 1)

En fixant *kfrpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *kfrpow*. La valeur par défaut de *kfrpow* est 0.



Un graphique des distributions pour différentes valeurs de *kfrpow*.

kprpow -- variation de la distribution de phase aléatoire (voir *kfrpow*). En fixant *kphs* et *kpmd* à 0.5, et *kprpow* à 0 on émulera *grain2*.

kfn -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k (on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).

Note

grain3 utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

Exemples

Voici un exemple de l'opcode grain3. Il utilise le fichier *grain3.csd* [examples/grain3.csd].

Exemple 224. Exemple de l'opcode grain3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain3.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr =  48000
kr =  1000
ksmps   =  48
nchnls =  1

/* Bartlett window */
itmp ftgen 1, 0, 16384, 20, 3, 1
/* sawtooth wave */
itmp ftgen 2, 0, 16384, 7, 1, 16384, -1
/* sine */
itmp ftgen 4, 0, 1024, 10, 1
/* window for "soft sync" with 1/32 overlap */
itmp ftgen 5, 0, 16384, 7, 0, 256, 1, 7936, 1, 256, 0, 7936, 0
/* generate bandlimited sawtooth waves */
itmp ftgen 3, 0, 4096, -30, 2, 1, 2048
icnt =  0
loop01:
; 100 tables for 8 octaves from 30 Hz
ifrq =  30 * exp(log(2) * 8 * icnt / 100)
itmp ftgen icnt + 100, 0, 4096, -30, 3, 1, sr / (2 * ifrq)
icnt =  icnt + 1
        if (icnt < 99.5) igoto loop01
/* convert frequency to table number */
#define FRQ2FNUM(xout'xcps'xbsfn) #

$xout =  int(($xbsfn) + 0.5 + (100 / 8) * log(($xcps) / 30) / log(2))
$xout limit $xout, $xbsfn, $xbsfn + 99

#

/* instr 1: pulse width modulated grains */

        instr 1

kfrq =  523.25          ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kfmd =  kfrq * 0.02          ; random variation in frequency
kgdur =  0.2                  ; grain duration
kdens =  200                  ; density
iseed =  1                  ; random seed

kphs oscili 0.45, 1, 4 ; phase

a1 grain3 kfrq, 0, kfmd, 0.5, kgdur, kdens, 100,          \
                kfnum, 1, -0.5, 0, iseed, 2
a2 grain3 kfrq, 0.5 + kphs, kfmd, 0.5, kgdur, kdens, 100, \
                kfnum, 1, -0.5, 0, iseed, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

        out aenv * 2250 * (a1 - a2)

        endin

/* instr 2: phase variation */

        instr 2

kfrq =  220                  ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kgdur =  0.2                  ; grain duration
kdens =  200                  ; density
iseed =  2                  ; random seed

kprdst expon 0.5, p3, 0.02 ; distribution

a1 grain3 kfrq, 0.5, 0, 0.5, kgdur, kdens, 100,          \
                kfnum, 1, 0, -kprdst, iseed, 64

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

        out aenv * 1500 * a1

        endin

/* instr 3: "soft sync" */
```

```
        instr 3

kdens =  130.8          ; base frequency
kgdur =  2 / kdens          ; grain duration

kfrq expon 880, p3, 220 ; oscillator frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number

a1 grain3 kfrq, 0, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2
a2 grain3 kfrq, 0.667, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

        out aenv * 10000 * (a1 - a2)

        endin


</CsInstruments>
<CsScore>

t 0 60
i 1 0 3
i 2 4 3
i 3 8 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

grain2

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga

granule

granule — Un générateur de texture par synthèse granulaire plus complexe.

Description

Le générateur unitaire *granule* est plus complexe que *grain*, mais il ajoute de nouvelles possibilités.

granule est un générateur unitaire de Csound qui emploie une table d'onde en entrée pour produire une sortie audio par synthèse granulaire. Les données de la table d'onde peuvent être générée par n'importe laquelle des routines GEN telle que *GEN01* qui lit un fichier audio. On peut ainsi utiliser un son échantillonné comme source pour les grains. L'implémentation interne comprend jusqu'à 128 voix. Le nombre maximum de voix peut être augmenté en redéfinissant la variable MAXVOICE dans le fichier grain4.h. *granule* possède son propre générateur de nombres aléatoires pour produire toutes les fluctuations aléatoires des paramètres. Il comprend aussi une fonction de seuil pour scanner la table de fonction source lors de la phase d'initialisation. On peut ainsi facilement ignorer les passages de silence entre les phrases.

Les caractéristiques de la synthèse sont contrôlées par 22 paramètres. *xamp* est l'amplitude de la sortie et elle peut varier aussi bien au taux audio qu'au taux de contrôle.

Syntaxe

```
ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \
      igskip_os, ilength, kgap, igap_os, kgsize, igsize_os, iatt, idec \
      [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]
```

Initialisation

ivoice -- nombre de voix.

iratio -- rapport entre la vitesse du pointeur de lecture et le taux d'échantillonnage de la sortie, par exemple 0,5 donnera une vitesse de lecture moitié de la vitesse originale.

imode -- +1, le pointeur de lecture progresse en avant (direction naturelle du fichier source), -1, en arrière (direction opposée à la direction naturelle du fichier source), ou 0, pour une direction aléatoire.

ithd -- seuil ; lorsque le signal échantillonné dans la table est plus petit que *ithd*, il est ignoré.

ifn -- numéro de la table de fonction de la source sonore.

ipshift -- contrôle de la transposition. Si *ipshift* vaut 0, la hauteur sera fixée aléatoirement dans un ambitus d'une octave de part et d'autre de la hauteur de chaque grain. Si *ipshift* vaut 1, 2, 3 ou 4, on peut fixer jusqu'à quatre hauteurs différentes pour le nombre de voix défini dans *ivoice*. Les paramètres facultatifs *ipitch1*, *ipitch2*, *ipitch3* et *ipitch4* servent à quantifier les transpositions.

igskip -- décalage initial depuis le début de la table de fonction en sec.

igskip_os -- fluctuation aléatoire du pointeur de lecture en sec, 0 signifiant pas de décalage.

ilength -- longueur de la partie de la table à utiliser à partir de *igskip* en sec.

igap_os -- fluctuation aléatoire de l'écart en % de la taille de l'écart, 0 signifiant pas de décalage.

igsize_os -- fluctuation aléatoire de la taille du grain en % de la taille du grain, 0 signifiant pas de décalage.

iatt -- attaque de l'enveloppe du grain en % de la taille du grain.

idec -- chute de l'enveloppe du grain en % de la taille du grain.

iseed (facultatif, par défaut 0,5) -- graine pour le générateur de nombre aléatoire.

ipitch1, ipitch2, ipitch3, ipitch4 (facultatif, par défaut 1) -- paramètre de transposition, utilisé lorsque *ipshift* vaut 1, 2, 3 ou 4. La transposition est réalisée par une technique de pondération temporelle avec interpolation linéaire entre les points. La valeur par défaut de 1 signifie la hauteur originale.

ifnenv (facultatif, par défaut 0) -- numéro de la table de fonction utilisée pour générer la forme de l'enveloppe.

Exécution

xamp -- amplitude.

kgap -- écart entre les grains en sec.

kgsize -- taille du grain en sec.

Exemples

Voici un exemple de l'opcode granule. Il utilise le fichier *granule.csd* [examples/granule.csd], and *mary.wav* [examples/mary.wav].

Exemple 225. Exemple de l'opcode granule.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o granule.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
instr 1
;
k1      linseg 0,0.5,1,(p3-p2-1),1,0.5,0
a1      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19,p20,p21,p22,p23,p24
a2      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19, p20+0.17,p21,p22,p23,p24
outs a1,a2
endin


</CsInstruments>
<CsScore>

; f statement read sound file mary.wav in the SFDIR
```

```
; directory into f-table 1
f1      0 262144 1 "mary.wav" 0 0 0
i1      0 10 2000 64 0.5 0 0 1 4 0 0.005 5 0.01 50 0.02 50 30 30 0.39 \
        1 1.42 0.29 2
e

</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus lit un fichier son nommé *mary.wav* dans la table de fonction numéro 1 en gardant 262 144 échantillons. Il génère 10 secondes de sortie stéréo à partir de la table de fonction. Dans le fichier orchestre, tous les paramètres nécessaires au contrôle de la synthèse proviennent du fichier partition. Un générateur de fonction *linseg* est utilisé pour produire une enveloppe avec une attaque et une chute linéaires de 0,5 secondes. On obtient un effet stéréo par l'utilisation de différentes graines pour les deux appels de la fonction *granule*. Dans l'exemple, on ajoute 0,17 à p20 avant de le passer au second appel de *granule* pour s'assurer que toutes les fluctuations aléatoires seront différentes de celles du premier appel.

Voici la signification des paramètres dans le fichier partition :

Parameter	**Interpreted As**
p5 (*ivoice*)	le nombre de voix est fixé à 64
p6 (*iratio*)	fixé à 0,5, on lit la table d'onde deux fois moins vite que le taux de la sortie audio
p7 (*imode*)	fixé à 0, le pointeur du grain ne se déplace qu'en avant
p8 (*ithd*)	fixé à 0, pas de détection de seuil
p9 (*ifn*)	fixé à 1, on utilise la table de fonction numéro 1
p10 (*ipshift*)	fixé à 4, quatre hauteurs différentes seront générées
p11 (*igskip*)	fixé à 0 et p12 (*igskip_os*) est fixé à 0,005, pas de décalage par rapport au début de table d'onde et on utilise une fluctuation aléatoire de 5 ms
p13 (*ilength*)	fixé à 5, on n'utilise que 5 secondes de la table d'onde
p14 (*kgap*)	fixé à 0,01 et p15 (*igap_os*) est fixé à 50, on utilise un écart de 10 ms avec une fluctuation aléatoire de 50%
p16 (*kgsize*)	fixé à 0,02 et p17 (*igsize_os*) est fixé à 50, la durée du grain est de 20 ms avec une fluctation aléatoire de 50%
p18 (*iatt*) et p19 (*idec*)	fixés à 30, on applique une attaque et une chute linéaires de 30% au grain
p20 (*iseed*)	la graine pour le générateur de nombre aléatoire est fixée à 0,39
p21 - p24	les hauteurs sont fixées à 1, soit la hauteur originale, 1,42 soit une quinte plus haut, 0,29 soit une septième plus bas et enfin 2 soit une octave plus haut.

Crédits

Auteur : Allan Lee

Belfast

1996

Nouveau dans la version 3.35

guiro

guiro — Modèle semi-physique d'un son de guiro.

Description

guiro est un modèle semi-physique d'un son de guiro. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]
```

Initialisation

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

idamp (facultatif) -- le facteur d'amortissement de l'instrument. *Inutilisé.*

imaxshake (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

ifreq (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500.

ifreq1 (facultatif) -- la première fréquence de résonance.

Exécution

kamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

Exemples

Voici un exemple de l'opcode guiro. Il utilise le fichier *guiro.csd* [examples/guiro.csd].

Exemple 226. Exemple de l'opcode guiro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o guiro.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

    instr 01  ;example of a guiro
a1  guiro p4, 0.01
    out a1
    endin


</CsInstruments>
<CsScore>

i1 0 1 20000
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

bamboo, *dripwater*, *sleighbells*, *tambourine*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

harmon

harmon — Analyze an audio input and generate harmonizing voices in synchrony.

Description

Analyze an audio input and generate harmonizing voices in synchrony.

Syntax

```
ares harmon asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \
      iminfrq, iprd
```

Initialization

imode -- interpreting mode for the generating frequency inputs *kgenfreq1*, *kgenfreq2*. 0: input values are ratios with respect to the audio signal analyzed frequency. 1: input values are the actual requested frequencies in Hz.

iminfrq -- the lowest expected frequency (in Hz) of the audio input. This parameter determines how much of the input is saved for the running analysis, and sets a lower bound on the internal pitch tracker.

iprd -- period of analysis (in seconds). Since the internal pitch analysis can be time-consuming, the input is typically analyzed only each 20 to 50 milliseconds.

Performance

kestfrq -- estimated frequency of the input.

kmaxvar -- the maximum variance (expects a value betwee 0 and 1).

kgenfreq1 -- the first generated frequency.

kgenfreq2 -- the second generated frequency.

This unit is a harmonizer, able to provide up to two additional voices with the same amplitude and spectrum as the input. The input analysis is assisted by two things: an input estimated frequency *kestfrq* (in Hz), and a fractional maximum variance *kmaxvar* about that estimate which serves to limit the size of the search. Once the real input frequency is determined, the most recent pulse shape is used to generate the other voices at their requested frequencies.

The three frequency inputs can be derived in various ways from a score file or MIDI source. The first is the expected pitch, with a variance parameter allowing for inflections or inaccuracies; if the expected pitch is zero the harmonizer will be silent. The second and third pitches control the output frequencies; if either is zero the harmonizer will output only the non-zero request; if both are zero the harmonizer will be silent. When the requested frequency is higher than the input, the process requires additional computation due to overlapped output pulses. This is currently limited for efficiency reasons, with the result that only one voice can be higher than the input at any one time.

This unit is useful for supplying a background chorus effect on demand, or for correcting the pitch of a faulty input vocal. There is essentially no delay between input and output. Output includes only the generated parts, and does not include the input.

Examples

Here is an example of the harmon opcode. It uses the file *harmon.csd* [examples/harmon.csd].

Exemple 227. Example of the harmon opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o harmon.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; The frequency of the base note.
  inote = 440

  ; Generate the base note.
  avco vco 20000, inote, 1

  kestfrq = inote
  kmaxvar = 0.4

  ; Calculate frequencies 3 semitones above and
  ; below the base note.
  kgenfreq1 = inote * semitone(3)
  kgenfreq2 = inote * semitone(-3)

  imode = 1
  iminfrq = inote - 200
  iprd = 0.1

  ; Generate the harmony notes.
  a1 harmon avco, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, \
            imode, iminfrq, iprd

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Barry L. Vercoe
M.I.T., Cambridge, Mass
1997

New in version 3.47

Example written by Kevin Conder.

harmon2

harmon2 — Analyze an audio input and generate harmonizing voices in synchrony with formants preserved.

Description

Generate harmonizing voices with formants preserved.

Syntax

```
ares harmon2 asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]
```

```
ares harmon3 asig, koct, kfrq1, \
    kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]
```

```
ares harmon4 asig, koct, kfrq1, \
    kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]
```

Initialization

icpsmode -- interpreting mode for the generating frequency inputs *kfrq1*, *kfrq2*, *kfrq2* and *kfrq4:* 0: input values are ratios w.r.t. the cps equivalent of *koct.* 1: input values are the actual requested frequencies in cps.

ilowest -- owest value of the koct input for which harmonizing voices will be generated.

ipolarity -- polarity of asig input, 1 = positive glottal pulses, 0 = negative. Default is 1.

Performance

Harmon2, **harmon3** and **harmon4** are high-performance harmonizers, able to provide up to four pitch-shifted copies of the input asig with spectral formants preserved. The pitch-shifting algorithm requires an accurate running estimate (*koct*, in decimal oct units) of the pitched content of *asig*, normally gained from an independent pitch tracker such as *specptrk*. The algorithm then isolates the most recent full pulse within *asig*, and uses this to generate the other voices at their required pulse rates.

If the frequency (or ratio) presented to *kfrq1*, *kfrq2*, *kfrq3* or *kfrq4* is zero, then no signal is generated for that voice. If any of them is non-zero, but the *koct* input is below the value *ilowest*, then that voice will output a direct copy of the input *asig*. As a consequence, the data arriving at the k-rate inputs can variously cause the generated voices to be turned on or off, to pass a direct copy of a non-voiced fricative source, or to harmonize the source according to some constructed algorithm. The transition from one mode to another is cross-faded, giving seemless alternating between voiced (harmonized) and non-voiced fricatives during spoken or sung input.

harmon2, *harmon3*, *harmon4* are especially matched to the output of *specptrk*. The latter generates pitch data in decimal octave format; it also emits its base value if no pitch is identified (as in fricative noise) and emits zero if the energy falls below a threshold, so that *harmon2*, *harmon3*, *harmon4* can be set to pass the direct signal in both cases. Of course, any other form of pitch estimation could also be used. Since pitch trackers usually incur a slight delay for accurate estimation (for *specptrk* the delay is printed by the spectrum unit), it is normal to delay the audio signal by the same amount so that *harmon2*, *harmon3*, *harmon4* can work from a fully concurrent estimate.

Examples

Here is an example of the harmon opcode. It uses the file *harmon.csd* [examples/harmon.csd].

Exemple 228. Example of the harmon2 opcode.

```
a1,a2 ins                                          ; get mic input
w1 spectrum          a1, .02, 7, 24, 12, 1, 3 ; and examine it
koct,kamp specptrk          w1, 1, 6.5, 9.5, 7.5, 10, 7, .7, 0, 3, 1
a3 delay         a1, .065                 ; allow for ptrk delay
a4 harmon2         a3, koct, 1.25, 0.75, 0, 6.9 ; output a fixed 6-4 harmony
        outs         a3, a4                         ; as well as the original
```

Credits

Author: Barry L. Vercoe
M.I.T., Cambridge, Mass
2006

New in version 5.04

hilbert

hilbert — A Hilbert transformer.

Description

An IIR implementation of a Hilbert transformer.

Syntax

```
ar1, ar2 hilbert asig
```

Performance

asig -- input signal

ar1 -- sine output of *asig*

ar2 -- cosine output of *asig*

hilbert is an IIR filter based implementation of a broad-band 90 degree phase difference network. The input to *hilbert* is an audio signal, with a frequency range from 15 Hz to 15 kHz. The outputs of *hilbert* have an identical frequency response to the input (i.e. they sound the same), but the two outputs have a constant phase difference of 90 degrees, plus or minus some small amount of error, throughout the entire frequency range. The outputs are in quadrature.

hilbert is useful in the implementation of many digital signal processing techniques that require a signal in phase quadrature. *ar1* corresponds to the cosine output of *hilbert*, while *ar2* corresponds to the sine output. The two outputs have a constant phase difference throughout the audio range that corresponds to the phase relationship between cosine and sine waves.

Internally, *hilbert* is based on two parallel 6th-order allpass filters. Each allpass filter implements a phase lag that increases with frequency; the difference between the phase lags of the parallel allpass filters at any given point is approximately 90 degrees.

Unlike an FIR-based Hilbert transformer, the output of *hilbert* does not have a linear phase response. However, the IIR structure used in *hilbert* is far more efficient to compute, and the nonlinear phase response can be used in the creation of interesting audio effects, as in the second example below.

Examples

The first example implements frequency shifting, or single sideband amplitude modulation. Frequency shifting is similar to ring modulation, except the upper and lower sidebands are separated into individual outputs. By using only one of the outputs, the input signal can be "detuned," where the harmonic components of the signal are shifted out of harmonic alignment with each other, e.g. a signal with harmonics at 100, 200, 300, 400 and 500 Hz, shifted up by 50 Hz, will have harmonics at 150, 250, 350, 450, and 550 Hz.

Here is the first example of the hilbert opcode. It uses the file *hilbert.csd* [examples/hilbert.csd], and *mary.wav* [examples/mary.wav].

Exemple 229. Example of the hilbert opcode implementing frequency shifting.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hilbert.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  idur = p3
  ; Initial amount of frequency shift.
  ; It can be positive or negative.
  ibegshift = p4
  ; Final amount of frequency shift.
  ; It can be positive or negative.
  iendshift = p5

  ; A simple envelope for determining the
  ; amount of frequency shift.
  kfreq linseg ibegshift, idur, iendshift

  ; Use the sound of your choice.
  ain soundin "mary.wav"

  ; Phase quadrature output derived from input signal.
  areal, aimag hilbert ain

  ; Quadrature oscillator.
  asin oscili 1, kfreq, 1
  acos oscili 1, kfreq, 1, .25

  ; Use a trigonometric identity.
  ; See the references for further details.
  amod1 = areal * acos
  amod2 = aimag * asin

  ; Both sum and difference frequencies can be
  ; output at once (sum is made up by the positive freqs, diff by the negative)
  ; aupshift corresponds to the positive freq side of ain, shifted up by kfreq.
  aupshift = (amod1 - amod2) * 0.7
  ; adownshift corresponds to the negative freq side of ain, also shifted up by kfreq.
  adownshift = (amod1 + amod2) * 0.7

  ; Notice that the adding of the two together is
  ; identical to the output of ring modulation.

  out aupshift
endin


</CsInstruments>
<CsScore>

; Sine table for quadrature oscillator.
f 1 0 16384 10 1

; Starting with no shift, ending with all
; frequencies shifted up by 200 Hz.
i 1 0 2 0 200

; Starting with no shift, ending with all
; frequencies shifted down by 200 Hz.
i 1 2 2 0 -200
e


</CsScore>
</CsoundSynthesizer>
```

The second example is a variation of the first, but with the output being fed back into the input. With very small shift amounts (i.e. between 0 and +-6 Hz), the result is a sound that has been described as a « barberpole phaser » or « Shepard tone phase shifter. » Several notches appear in the spectrum, and are constantly swept in the direction opposite that of the shift, producing a filtering effect that is reminiscent of Risset's « endless glissando ».

Here is the second example of the hilbert opcode. It uses the file *hilbert_barberpole.csd* [examples/hilbert_barberpole.csd], and *mary.wav* [examples/mary.wav].

Exemple 230. Example of the hilbert opcode sounding like a « barberpole phaser ».

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hilbert_barberpole.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
; kr must equal sr for the barberpole effect to work.
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1
instr 1
  idur = p3
  ibegshift = p4
  iendshift = p5

  ; sawtooth wave, not bandlimited
  asaw   phasor 100
  ; add offset to center phasor amplitude between -.5 and .5
  asaw = asaw - .5
  ; sawtooth wave, with amplitude of 10000
  ain = asaw * 20000

  ; The envelope of the frequency shift.
  kfreq linseg ibegshift, idur, iendshift

  ; Phase quadrature output derived from input signal.
  areal, aimag hilbert ain

  ; The quadrature oscillator.
  asin oscili 1, kfreq, 1
  acos oscili 1, kfreq, 1, .25

  ; Based on trignometric identities.
  amod1 = areal * acos
  amod2 = aimag * asin

  ; Calculate the up-shift and down-shift.
  aupshift = (amod1 + amod2) * 0.7
  adownshift = (amod1 - amod2) * 0.7

  ; Mix in the original signal to achieve the barberpole effect.
  amix1 = aupshift + ain
  amix2 = aupshift + ain

  ; Make sure the output doesn't get louder than the original signal.
  aout1 balance amix1, ain
```

```
  aout2 balance amix2, ain

  outs aout1, aout2
endin


</CsInstruments>
<CsScore>

; Table 1: A sine wave for the quadrature oscillator.
f 1 0 16384 10 1

; The score.
; p4 = frequency shifter, starting frequency.
; p5 = frequency shifter, ending frequency.
i 1 0 6 -10 10
e


</CsScore>
</CsoundSynthesizer>
```

Technical History

The use of phase-difference networks in frequency shifters was pioneered by Harald Bode.[1] Bode and Bob Moog provide an excellent description of the implementation and use of a frequency shifter in the analog realm in;[2] this would be an excellent first source for those that wish to explore the possibilities of single sideband modulation. Bernie Hutchins provides more applications of the frequency shifter, as well as a detailed technical analysis.[3] A recent paper by Scott Wardle[4] describes a digital implementation of a frequency shifter, as well as some unique applications.

References

1. H. Bode, "Solid State Audio Frequency Spectrum Shifter." AES Preprint No. 395 (1965).

2. H. Bode and R.A. Moog, "A High-Accuracy Frequency Shfiter for Professional Audio Applications." *Journal of the Audio Engineering Society*, July/August 1972, vol. 20, no. 6, p. 453.

3. B. Hutchins. *Musical Engineer's Handbook* (Ithaca, NY: Electronotes, 1975), ch. 6a.

4. S. Wardle, "A Hilbert-Transformer Frequency Shifter for Audio." Available online at *http://www.iua.upf.es/dafx98/papers/*.

Credits

Author: Sean Costello
Seattle, Washington
1999

New in Csound version 3.55

The examples were updated April 2002. Thanks go to Sean Costello for fixing the barberpole example.

hrtfer

hrtfer — Creates 3D audio for two speakers.

Description

Output is binaural (headphone) 3D audio.

Syntax

```
aleft, aright hrtfer asig, kaz, kelev, « HRTFcompact »
```

Initialization

kAz -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

kElev -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions above horizontal.

At present, the only file which can be used with *hrtfer* is *HRTFcompact* [examples/HRTFcompact]. It must be passed to the opcode as the last argument within quotes as shown above.

HRTFcompact may also be obtained via anonymous ftp from: *ftp://ftp.cs.bath.ac.uk/pub/dream/utilities/Analysis/HRTFcompact*

Performance

These unit generators place a mono input signal in a virtual 3D space around the listener by convolving the input with the appropriate HRTF data specified by the opcode's azimuth and elevation values. *hrtfer* allows these values to be k-values, allowing for dynamic spatialization. *hrtfer* can only place the input at the requested position because the HRTF is loaded in at i-time (remember that currently, CSound has a limit of 20 files it can hold in memory, otherwise it causes a segmentation fault). The output will need to be scaled either by using balance or by multiplying the output by some scaling constant.

Note

The sampling rate of the orchestra must be 44.1kHz. This is because 44.1kHz is the sampling rate at which the HRTFs were measured. In order to be used at a different rate, the HRTFs would need to be re-sampled at the desired rate.

Examples

Here is an example of the hrtfer opcode. It uses the file *hrtfer.csd* [examples/hrtfer.csd], *HRTFcompact* [examples/HRTFcompact], and *beats.wav* [examples/beats.wav].

Exemple 231. Example of the hrtfer opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command

line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hrtfer.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
  kaz          linseg 0, p3, -360  ; move the sound in circle
  kel          linseg -40, p3, 45  ; around the listener, changing
                                    ; elevation as its turning
  asrc         soundin "beats.wav"
  aleft,aright hrtfer asrc, kaz, kel, "HRTFcompact"
  aleftscale   = aleft * 200
  arightscale  = aright * 200

  outs         aleftscale, arightscale
endin


</CsInstruments>
<CsScore>

i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

hrtfmove, *hrtfmove2*, *hrtfstat*.

Credits

Authors: Eli Breder and David MacIntyre
Montreal
1996

Fixed the example thanks to a message from Istvan Varga.

hrtfmove

hrtfmove — Generates dynamic 3d binaural audio for headphones using magnitude interpolation and phase truncation.

Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener by convolving the source with stored head related transfer function (HRTF) based filters.

Syntax

```
aleft, aright hrtfmove asrc, kAz, kElev, ifilel, ifiler [, imode, ifade, isr]
```

Initialization

Initialization

ifilel -- left HRTF spectral data file

ifiler -- right HRTF spectral data file

Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing *sr* should match datafile *sr*. Files should be in the current directory or the SADIR (see *Environment Variables*).

imode -- optional, default 0 for phase truncation, 1 for minimum phase

ifade -- optional, number of processing buffers for phase change crossfade (default 8). Legal range is 1-24. A low value is recommended for complex sources (4 or less: a higher value may make the cross-fade audible), a higher value (8 or more: a lower value may make the inconsistency when the filter changes phase values audible) for narrowband sources. Does not effect minimum phase processing.

Note

Ocassionally fades may overlap (when unnaturally fast/complex trajectories are requested). In this case, a warning will be printed. Use a smaller crossfade or slightly change trajectory to avoid any possible inconsistencies that may arise.

isr - optional, default 44.1kHz, legal values: 44100, 48000 and 96000.

kAz -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

kElev -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

Artifact-free user-defined trajectories are made possible using an interpolation algorithm based on spec-

tral magnitude interpolation and phase truncation. Crossfades are implemented to minimise/eliminate any inconsistencies caused by updating phase values. These crossfades are performed over a user definable number of convolution processing buffers. Complex sources may only need to crossfade over 1 buffer; narrow band sources may need several. The opcode also offers minimum phase based processing, a more traditional and complex method. In this mode, the hrtf filters used are reduced to minimum phase representations and the interpolation process then uses the relationship between minimum phase magnitude and phase spectra. Interaural time difference, which is inherent to the phase truncation process, is reintroduced in the minimum phase process using variable delay lines.

Examples

Here is an example of the hrtfmove opcode. It uses the file *hrtfmove.csd* [examples/hrtfmove.csd].

Exemple 232. Example of the htrfmove opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select flags here
; realtime audio out
; -o dac
; For Non-realtime ouput leave only the line below:
 -o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gasrc init 0

instr 1         ;a plucked string

  kamp = p4
  kcps = cpspch(p5)
  icps = cpspch(p5)

  a1 pluck kamp, kcps, icps, 0, 1

  gasrc = a1

endin

instr 10 ;uses output from instr1 as source

 kaz linseg 0, p3, 720         ;2 full rotations

 aleft,aright hrtfmove gasrc, kaz,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

 outs aleft, aright

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00
```

```
; Play Instrument 10 for 10 seconds.
i10 0 10

</CsScore>
</CsoundSynthesizer>
```

See Also

hrtfmove2, *hrtfstat*, *hrtfer*.

Credits

Author: Brian Carty
Maynooth
2008

hrtfmove2

hrtfmove2 — Generates dynamic 3d binaural audio for headphones using a Woodworth based spherical head model with improved low frequency phase accuracy.

Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener using head related transfer function (HRTF) based filters.

Syntax

```
aleft, aright hrtfmove2 asrc, kAz, kElev, ifilel, ifiler [,ioverlap, iradius, isr]
```

Initialization

ifilel -- left HRTF spectral data file

ifiler -- right HRTF spectral data file

Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing *sr* should match datafile *sr*. Files should be in the current directory or the SADIR (see *Environment Variables*).

ioverlap -- optional, number of overlaps for STFT processing (default 4). See STFT section of manual.

iradius -- optional, head radius used for phase spectra calculation in centimeters (default 9.0)

isr - optional, default 44.1kHz, legal values: 44100, 48000 and 96000.

Performance

asrc -- Input/source signal.

kAz -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

kElev -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

Artifact-free user-defined trajectories are made possible using an interpolation algorithm based on spectral magnitude interpolation and a derived phase spectrum based on the Woodworth spherical head model. Accuracy is increased for the data set provided by extracting and applying a frequency dependent scaling factor to the phase spectra, leading to a more precise low frequency interaural time difference. Users can control head radius for the phase derivation, allowing a crude level of individualisation. The dynamic source version of the opcode uses a Short Time Fourier Transform algorithm to avoid artefacts caused by derived phase spectra changes. STFT processing means this opcode is more computationally intensive than *hrtfmove* using phase truncation, but phase is constantly updated by *hrtfmove2*.

Examples

Here is an example of the hrtfmove2 opcode. It uses the file *hrtfmove2.csd* [examples/hrtfmove2.csd].

Exemple 233. Example of the htrfmove2 opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select flags here
; realtime audio out
; -o dac
; For Non-realtime ouput leave only the line below:
 -o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gasrc init 0

instr 1         ;a plucked string

  kamp = p4
  kcps = cpspch(p5)
  icps = cpspch(p5)

  a1 pluck kamp, kcps, icps, 0, 1

  gasrc = a1

endin

instr 10 ;uses output from instr1 as source

 kaz linseg 0, p3, 720         ;2 full rotations

 aleft,aright hrtfmove2 gasrc, kaz,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

 outs aleft, aright

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00

; Play Instrument 10 for 10 seconds.
i10 0 10

</CsScore>
</CsoundSynthesizer>
```

See Also

hrtfmove, *hrtfstat*, *hrtfer*.

Credits

Author: Brian Carty
Maynooth
2008

hrtfstat

hrtfstat — Generates static 3d binaural audio for headphones using a Woodworth based spherical head model with improved low frequency phase accuracy.

Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener using head related transfer function (HRTF) based filters. It produces a static output (azimuth and elevation parameters are i-rate), because a static source allows much more efficient processing than *hrtfmove* and *hrtfmove2*,.

Syntax

```
aleft, aright hrtfstat asrc, iAz, iElev, ifilel, ifiler [,iradius, isr]
```

Initialization

iAz -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

iElev -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

ifilel -- left HRTF spectral data file

ifiler -- right HRTF spectral data file

Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing sr should match datafile sr. Files should be in the current directory or the SADIR (see *Environment Variables*).

iradius -- optional, head radius used for phase spectra calculation in centimeters (default 9.0)

isr - optional (default 44.1kHz). Legal values are 44100, 48000 and 96000.

Performance

Artifact-free user-defined static spatialisation is made possible using an interpolation algorithm based on spectral magnitude interpolation and a derived phase based on the Woodworth spherical head model. Accuracy is increased for the data set provided by extracting and applying a frequency dependent scaling factor to the phase spectra, leading to a more precise low frequency interaural time difference. Users can control head radius for the phase derivation, allowing a crude level of individualisation. The static source version of the opcode uses overlap add convolution (it does not need STFT processing, see *hrtfmove2*), and is thus considerably more efficient than *hrtfmove2* or *hrtfmove*, but cannot generate moving sources.

Examples

It can be found in the file *htrfstat.csd* [examples/htrfstat.csd].

Exemple 234. Example of the htrfstat opcode.

```
<CsoundSynthesizer>
<CsOptions>
    ; Select flags here
    ; realtime audio out
    ; -o dac
    ; For Non-realtime ouput leave only the line below:
    -o hrtf.wav

</CsOptions>
<CsInstruments>

    sr = 44100
    kr = 4410
    ksmps = 10
    nchnls = 2

    gasrc init 0

    instr 1         ;a plucked string

    kamp = p4
    kcps = cpspch(p5)
    icps = cpspch(p5)

    a1 pluck kamp, kcps, icps, 0, 1

    gasrc = a1

    endin

    instr 10 ;uses output from instr1 as source

    aleft,aright hrtfstat gasrc, 90,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

    outs aleft, aright

    endin

</CsInstruments>
<CsScore>

    ; Play Instrument 1: a plucked string
    i1 0 2 20000 8.00

    ; Play Instrument 10 for 2 seconds.
    i10 0 2

</CsScore>
</CsoundSynthesizer>
```

See Also

hrtfmove, *hrtfmove2*, *hrtfer*.

Credits

Author: Brian Carty
Maynooth

2008

hsboscil

hsboscil — Un oscillateur qui prend en arguments l'intonation et la brillance.

Description

Un oscillateur qui prend en arguments l'intonation et la brillance, relativement à une fréquence de base.

Syntaxe

```
ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \
              [, ioctcnt] [, iphs]
```

Initialisation

ibasfreq -- fréquence de base par rapport à laquelle l'intonation et la brillance sont relatives.

iwfn -- table de fonction de la forme d'onde, habituellement une sinus.

ioctfn -- table de fonction utilisée pour pondérer les octaves, habituellement quelque chose comme

f1 0 1024 -19 1 0.5 270 0.5

ioctcnt (facultatif) -- nombre d'octaves utilisées pour le mélange de brillance. Doit valoir entre 2 et 10. Par défaut = 3.

iphs (facultatif, par défaut = 0) -- phase initiale de l'oscillateur. Si *iphs* = -1, l'initialisation est ignorée.

Exécution

kamp -- amplitude de la note

ktone -- paramètre cyclique d'intonation cyclique relatif à *ibasfreq* en octave logarithmique, entre 0 et 1, des valeurs > 1 peuvent être utilisées, et sont réduites en interne à *frac*(*ktone*).

kbrite -- paramètre de brillance relatif à *ibasfreq*, obtenue en pondérant *ioctcnt* octaves. Il est échelonné de telle manière qu'une valeur de 0 correspond à la valeur originale de *ibasfreq*, 1 correspond à une octave au-dessus de *ibasfreq*, -2 correspond à deux octaves sous *ibasfreq*, etc. *kbrite* peut être fractionnaire.

hsboscil prend en arguments l'intonation et la brillance, relativement à une fréquence de base (*ibasfreq*). L'intonation est un paramètre cyclique dans l'octave logarithmique, la brillance est réalisée en mélangeant plusieurs octaves pondérées. Il est utile lorsque l'espace d'intonation est appréhendé dans un concept de coordonnées polaires.

Si *ktone* est une droite et *kbrite* une constante, le résultat produit est le glissando de Risset.

La table de l'oscillateur *iwfn* est toujours lue avec interpolation. Le temps d'exécution est approximativement *ioctcnt* * *oscili*.

Exemples

Voici un exemple de l'opcode hsboscil. Il utilise le fichier *hsboscil.csd* [examples/hsboscil.csd].

Exemple 235. Exemple de l'opcode hsboscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hsboscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
; blending window
giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

; Instrument #1 - produces Risset's glissando.
instr 1
  kamp = 10000
  kbrite = 0.5
  ibasfreq = 200
  ioctcnt = 5

  ; Change ktone linearly from 0 to 1,
  ; over the period defined by p3.
  ktone line 0, p3, 1

  a1 hsboscil kamp, ktone, kbrite, ibasfreq, giwave, giblend, ioctcnt
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode hsboscil dans un instrument MIDI. Il utilise le fichier *hsboscil_midi.csd* [examples/hsboscil_midi.csd].

Exemple 236. Exemple de l'opcode hsboscil dans un instrument MIDI.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d           -M0    ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o hsboscil_midi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
; blending window
giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

; Instrument #1 - use hsboscil in a MIDI instrument.
instr 1
  ibase = cpsoct(6)
  ioctcnt = 5

  ; all octaves sound alike.
  itona octmidi
  ; velocity is mapped to brightness
  ibrite ampmidi 3

  ; Map an exponential envelope for the amplitude.
  kenv expon 20000, 1, 100

  asig hsboscil kenv, itona, ibrite, ibase, giwave, giblend, ioctcnt
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten minutes
i 1 0 600
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Peter Neubäcker
Munich, Allemagne
Août 1999

Nouveau dans la version 3.58 de Csound

hvs1

hvs1 — Allows one-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Description

hvs1 allows one-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Syntax

```
hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

Initialization

inumParms - number of parameters controlled by the HVS. Each HVS snapshot is made up of inumParms elements.

inumPointsX - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

iOutTab - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

iPositionsTab – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

iSnapTab – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be >= to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

iConfigTab – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particolar kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

Performance

kx - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of “mainstays” (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is 3 * 5 * 2 = 30. With this set of pivot points, the cube Is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is enstabilished in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

Tableau 10.

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the iSnapTab, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

Examples

Here is an example of the hvs1 opcode. It uses the file *hvs1.csd* [examples/hvs1.csd].

Exemple 237. Example of the hvs1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2

; Example by Gabriel Maldonado and Andres Cabrera

0dbfs = 1

ginumLinesX init 16
ginumParms  init 3


giOutTab ftgen 5,0,8, -2,      0
giPosTab ftgen 6,0,32, -2,     3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2,     1,1,1,   2,0,0,  3,2,0,  2,2,2,  5,2,1,  2,3,4,  6,1,7,    0,0,0, \
                              1,3,5,   3,4,4,  1,5,8,  1,1,5,  4,3,2,  3,4,5,  7,6,5,    7,8,9
tb0_init giOutTab

        FLpanel "hsv1",500,100,10,10,0
gk1,ih1 FLslider "X", 0,1,  0,5, -1, 400,30, 50,20
        FLpanel_end
        FLrun

        instr 1
;               kx,   inumParms,  inumPointsX,  iOutTab,  iPosTab,  iSnapTab  [, iConfigTab]
        hvs1    gk1,  ginumParms, ginumLinesX, giOutTab, giPosTab, giSnapTab  ;, iConfigTab

k0 init 0
k1 init 1
k2 init 2
```

```
printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

aosc1 oscil tb0(k0)/20, tb0(k1)*100 + 200, 1
aosc2 oscil tb0(k1)/20, tb0(k2)*100 + 200, 1
aosc3 oscil tb0(k2)/20, tb0(k0)*100 + 200, 1
aosc4 oscil tb0(k1)/20, tb0(k0)*100 + 200, 1
aosc5 oscil tb0(k2)/20, tb0(k1)*100 + 200, 1
aosc6 oscil tb0(k0)/20, tb0(k2)*100 + 200, 1

outs aosc1 + aosc2 + aosc3, aosc4 + aosc5 + aosc6
        endin


</CsInstruments>
<CsScore>

f1 0 1024 10 1
f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

hvs2, *hvs3*, *vphaseseg*

Credits

Author: Gabriel Maldonado

New in version 5.06

hvs2

hvs2 — Allows two-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Description

hvs2 allows two-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Syntax

```
hvs2 kx, ky, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

Initialization

inumParms - number of parameters controlled by the HVS. Each HVS snapshot is made up of inumParms elements.

inumPointsX - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

iOutTab - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

iPositionsTab – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

iSnapTab – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be >= to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

iConfigTab – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particolar kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

Performance

kx, ky - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of "mainstays" (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is 3 * 5 * 2 = 30. With this set of pivot points, the cube Is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is enstabilished in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

Tableau 11.

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the iSnapTab, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

Examples

Here is an example of the hvs2 opcode. It uses the file *hvs2.csd* [examples/hvs2.csd].

Exemple 238. Example of the hvs2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2

0dbfs = 1

ginumLinesX init 4
ginumLinesY init 4
ginumParms init 3

giOutTab ftgen 5,0,8, -2,      0
giPosTab ftgen 6,0,32, -2,     3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2,     1,1,1,   2,0,0,  3,2,0,  2,2,2,  5,2,1,  2,3,4,  6,1,7,    0,0,0, \
                              1,3,5,   3,4,4,  1,5,8,  1,1,5,  4,3,2,  3,4,5,  7,6,5,    7,8,9
tb0_init giOutTab

        FLpanel "Prova HVS2",600,400,10,100,0

gk1,    gk2,   ih1, ih2  FLjoy    "HVS controller XY", 0,    1,     1,     0,     0,     0,     -1,
; *ihandle,                       *numlinesX,   *numlinesY, *iwidth, *iheight, *ix, *iy,*image;
gihandle FLhvsBox ginumLinesX,    ginumLinesY,  300,    300,      300,  50, 1
        FLpanel_end
        FLrun


        instr 1

; Smooth control signals to avoid clicks
kx portk gk1, 0.02
ky portk gk2, 0.02
```

```
;               kx,  ky,  inumParms,  inumlinesX,  inumlinesY,  iOutTab,  iPosTab,  iSnapTab [, iConfigT
        hvs2  kx, ky, ginumParms, ginumLinesX, ginumLinesY, giOutTab, giPosTab, giSnapTab  ;, iConfigTa
;                        *kx, *ky, *ihandle;
        FLhvsBoxSetValue gk1, gk2, gihandle

k0 init 0
k1 init 1
k2 init 2

printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

  kris init 0.003
  kdur init 0.02
  kdec init 0.007

; Make parameters of synthesis depend on the table values produced by hvs
ares1 fof 0.2, tb0(k0)*100 + 50, tb0(k1)*100 + 200, 0, tb0(k2) * 10 + 50, 0.003, 0.02, 0.007, 20, \
      1, 2, p3
ares2 fof 0.2, tb0(k1)*100 + 50, tb0(k2)*100 + 200, 0, tb0(k0) * 10 + 50, 0.003, 0.02, 0.007, 20, \
      1, 2, p3

outs ares1, ares2
        endin


</CsInstruments>
<CsScore>

f 1 0 1024 10 1  ;Sine wave
f 2 0 1024 19 0.5 0.5 270 0.5  ;Grain envelope table

f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>
```

Here is second example of the hvs2 opcode. It uses the file *hvs2.csd* [examples/hvs2-2.csd].

Exemple 239. Example of the hvs2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hvs2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=48000
ksmps=100
nchnls=2

; Example by James Hearon 2008
; Edited by Andres Cabrera

ginumPointsX init       16
ginumPointsY init       16
ginumParms  init        3

;Generate 9 tables with arbitrary points
gitmp ftgen 100, 0, 16, -2, 70, 260, 390, 180, 200, 300, 980, 126, \
                330, 860, 580, 467, 220, 399, 1026, 1500
gitmp ftgen 200, 0, 16, -2, 100, 200, 300, 140, 600, 700, 880, 126, \
                330, 560, 780, 167, 220, 999, 1026, 1500
```

```
gitmp ftgen 300, 0, 16, -2, 400, 200, 300, 540, 600, 700, 880, 126, \
                330, 160, 780, 167, 820, 999, 1026, 1500
gitmp ftgen 400, 0, 16, -2, 100, 200, 800, 640, 600, 300, 880, 126, \
                330, 660, 780, 167, 220, 999, 1026, 1500
gitmp ftgen 500, 0, 16, -2, 200, 200, 360, 440, 600, 700, 880, 126, \
                330, 560, 380, 167, 220, 499, 1026, 1500
gitmp ftgen 600, 0, 16, -2, 100, 600, 300, 840, 600, 700, 880, 126, \
                330, 260, 980, 367, 120, 399, 1026, 1500
gitmp ftgen 700, 0, 16, -2, 100, 200, 300, 340, 200, 500, 380, 126, \
                330, 860, 780, 867, 120, 999, 1026, 1500
gitmp ftgen 800, 0, 16, -2, 100, 600, 300, 240, 200, 700, 880, 126, \
                130, 560, 980, 167, 220, 499, 1026, 1500
gitmp ftgen 900, 0, 16, -2, 100, 800, 200, 140, 600, 700, 680, 126, \
                330, 560, 780, 167, 120, 299, 1026, 1500

giOutTab ftgen   5,0,8, -2,      0
giPosTab ftgen   6,0,32, -2, 0,1,2,3,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen   8,0,64, -2,  1,1,1,   2,0,0,  3,2,0,  2,2,2,  \
        5,2,1,  2,3,4,  6,1,7,  0,0,0,  1,3,5,  3,4,4,  1,5,8,  1,1,5,  \
        4,3,2,  3,4,5,  7,6,5,  7,8,9

tb0_init        giOutTab

        FLpanel "hsv2",440,100,10,10,0
gk1,ih1 FLslider "X", 0,1, 0, 5, -1, 400,20, 20,10
gk2, ih2 FLslider "Y", 0, 1, 0, 5, -1, 400, 20, 20, 50
        FLpanel_end

        FLpanel "hvsBox",280,280,500,1000,0
;ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]
gih1   FLhvsBox  16, 16, 250, 250, 10, 1
        FLpanel_end
        FLrun


        instr   1
FLhvsBoxSetValue gk1, gk2, gih1

hvs2    gk1,gk2,  ginumParms, ginumPointsX, ginumPointsY, giOutTab, giPosTab, giSnapTab  ;, iConfigTab

k0      init    0
k1      init    1
k2      init    2
kspeed  init    0

kspeed = int((tb0(k2)) + 1)*.10

kenv  oscil   25000, kspeed*16, 10

k1    phasor kspeed ;slow phasor: 200 sec.
kpch  tableikt k1 * 16, int((tb0(k1)) +1)*100 ;scale phasor * length
a1    oscilikt kenv, kpch, int(tb0(k0)) +1000;scale pitch slightly
ahp butterlp a1, 2500
outs ahp, ahp

        endin


</CsInstruments>
<CsScore>

f 10  0  1024 20  5 ;use of windowing function
f1000 0 1024 10 .33 .25 .5
f1001 0 1024 10 1
f1002 0 1024 10 .5  .25  .05
f1003 0 1024 10 .05  .10  .3  .5  1
f1004 0 1024 10 1 .5  .25  .125  .625
f1005 0 1024 10 .33  .44  .55  .66
f1006 0 1024 10 1 1 1 1 1
f1007 0 1024 10 .05 .25 .05 .25 .05 1

f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

hvs1, *hvs3*, *vphaseseg*

Credits

Author: Gabriel Maldonado

New in version 5.06

hvs3

hvs3 — Allows three-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Description

hvs3 allows three-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

Syntax

```
hvs3 kx, ky, kz, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

Initialization

inumParms - number of parameters controlled by the HVS. Each HVS snapshot is made up of inumParms elements.

inumPointsX - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

iOutTab - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

iPositionsTab – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

iSnapTab – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be >= to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

iConfigTab – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particolar kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

Performance

kx, ky, kz - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of “mainstays” (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is 3 * 5 * 2 = 30. With this set of pivot points, the cube Is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is enstabilished in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

Tableau 12.

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the iSnapTab, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

See Also

hvs1, *hvs2*, *vphaseseg*

Credits

Author: Gabriel Maldonado

New in version 5.06

i

i — Retourne un équivalent de taux-i d'un argument de taux-k.

Description

Retourne un équivalent de taux-i d'un argument de taux-k.

Syntaxe

```
i(x) (arguments de taux-k seulement)
```

Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Note

L'utilisation de i() avec un argument expression de taux-k n'est pas recommandée et peut produire des résultats inattendus.

Voir Aussi

a, *k*, *abs*, *exp*, *frac*, *int*, *log*, *log10*, *sqrt*

ibetarand

ibetarand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

ibexprnd

ibexprnd — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

icauchy

icauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

ictrl14

ictrl14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl14*.

ictrl21

ictrl21 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl21*.

ictrl7

ictrl7 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl7*.

iexprand

iexprand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

if

if — Branchement conditionnel à l'initialisation ou durant l'exécution.

Description

if...igoto -- branchement conditionnel à l'initialisation, dépendant de la valeur de vérité de l'expression logique *ia R ib*. Le branchement n'a lieu que si le résultat est vrai.

if...kgoto -- branchement conditionnel durant l'exécution, dépendant de la valeur de vérité de l'expression logique *ka R kb*. Le branchement n'a lieu que si le résultat est vrai.

if...goto -- combinaison des deux versions ci-dessus. La condition est testée à chaque passage.

if...then -- donne la possibilité de spécifier des blocs conditionnels *if/else/endif*. Tous les blocs *if...then* doivent se terminer par une instruction *endif*. Les instructions *elseif* et *else* sont facultatives. On peut utiliser n'importe quel nombre d'instructions *elseif*. Il ne peut y avoir qu'une seule instruction *else* et elle doit être la dernière instruction conditionnelle avant l'instruction *endif*. Des blocs imbriqués de *if...then* sont permis.

Note

Notez que si la condition utilise une variable de taux-k (par exemple, « if kval > 0 »), l'instruction *if...goto* ou *if...then* sera ignorée lors de la phase d'initialisation. Cela permet une initialisation de l'opcode même si la variable de taux-k a déjà reçu une valeur appropriée par une instruction init antérieure.

Syntaxe

```
if ia R ib igoto label

if ka R kb kgoto label

if xa R xb goto label

if xa R xb then
```

où *label* est dans le même bloc d'instrument et n'est pas une expression, et où *R* est un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

Si l'on utilise *goto* ou *then* à la place de *kgoto* ou *igoto*, le comportement est déterminé par le type étant comparé. Si la comparaison utilise des variables de taux-k, kgoto est utilisé et vice-versa.

Note

Les instructions *If/then/goto* ne peuvent pas effectuer de comparaisons de type audio. On ne peut pas mettre de variables de type-a dans les expressions de comparaison pour ces opcodes. La raison en est que les variables audio sont des vecteurs qui ne peuvent pas être comparés de la même façon que des scalaires. Si l'on doit comparer des échantillons audio individuellement il faut utiliser *kr = 1* ou *Comparateurs et Accumulateurs*

Exemples

Voici une exemple de la combinaison if ... igoto. Il utilise le fichier *igoto.csd* [examples/igoto.csd].

Exemple 240. Exemple de la combinaison if ... igoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o igoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Get the value of the 4th p-field from the score.
  iparam = p4

  ; If iparam is 1 then play the high note.
  ; If not then play the low note.
  if (iparam == 1) igoto highnote
    igoto lownote

highnote:
  ifreq = 880
  goto playit

lownote:
  ifreq = 440
  goto playit

playit:
  ; Print the values of iparam and ifreq.
  print iparam
  print ifreq

  a1 oscil 10000, ifreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  iparam = 0.000
instr 1:  ifreq = 440.000
instr 1:  iparam = 1.000
instr 1:  ifreq = 880.000
```

Voici un exemple de la combinaison if ... kgoto. Il utilise le fichier *kgoto.csd* [examples/kgoto.csd].

Exemple 241. Exemple de la combinaison if ... kgoto.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o kgoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Change kval linearly from 0 to 2 over
  ; the period set by the third p-field.
  kval line 0, p3, 2

  ; If kval is greater than or equal to 1 then play the high note.
  ; If not then play the low note.
  if (kval >= 1) kgoto highnote
    kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit

playit:
  ; Print the values of kval and kfreq.
  printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

  a1 oscil 10000, kfreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
```

```
kval = 1.999639, kfreq = 880.000000
```

Exemples

Voici un exemple de la combinaison if ... then. Il utilise le fichier *if.csd* [examples/ifthen.csd].

Exemple 242. Exemple de la combinaison if ... then.

Voir Aussi

elseif, *else*, *endif*, *goto*, *igoto*, *kgoto*, *tigoto*, *timout*

Crédits

Exemples écrits par Kevin Conder.

Note ajoutée par Jim Aikin.

Février 2004. Note ajoutée par Matt Ingalls.

igauss

igauss — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

igoto

igoto — Transfer control during the i-time pass.

Description

During the i-time pass only, unconditionally transfer control to the statement labeled by *label*.

Syntax

```
igoto label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the igoto opcode. It uses the file *igoto.csd* [examples/igoto.csd].

Exemple 243. Example of the igoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o igoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Get the value of the 4th p-field from the score.
  iparam = p4

  ; If iparam is 1 then play the high note.
  ; If not then play the low note.
  if (iparam == 1) igoto highnote
    igoto lownote

highnote:
  ifreq = 880
  goto playit

lownote:
  ifreq = 440
  goto playit

playit:
  ; Print the values of iparam and ifreq.
  print iparam
```

```
  print ifreq

  a1 oscil 10000, ifreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
instr 1:  iparam = 0.000
instr 1:  ifreq = 440.000
instr 1:  iparam = 1.000
instr 1:  ifreq = 880.000
```

See Also

cggoto, *cigoto*, *ckgoto*, *goto*, *if*, *kgoto*, *rigoto*, *tigoto*, *timout*

Credits

Example written by Kevin Conder.

Added a note by Jim Aikin.

ihold

ihold — Creates a held note.

Description

Causes a finite-duration note to become a « held » note

Syntax

ihold

Performance

ihold -- this i-time statement causes a finite-duration note to become a « held » note. It thus has the same effect as a negative p3 (see score *i Statement*), except that p3 here remains positive and the instrument reclassifies itself to being held indefinitely. The note can be turned off explicitly with *turnoff*, or its space taken over by another note of the same instrument number (i.e. it is tied into that note). Effective at i-time only; no-op during a *reinit* pass.

Examples

Here is an example of the ihold opcode. It uses the file *ihold.csd* [examples/ihold.csd].

Exemple 244. Example of the ihold opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ihold.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; A simple oscillator with its note held indefinitely.
  a1 oscil 10000, 440, 1
  ihold

  ; If p4 equals 0, turn the note off.
  if (p4 == 0) kgoto offnow
    kgoto playit

offnow:
  ; Turn the note off now.
  turnoff
```

```
playit:
  ; Play the note.
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an ordinary sine wave.
f 1 0 32768 10 1

; p4 = turn the note off (if it is equal to 0).
; Start playing Instrument #1.
i 1 0 1 1
; Turn Instrument #1 off after 3 seconds.
i 1 3 1 0
e


</CsScore>
</CsoundSynthesizer>
```

See Also

i Statement, *turnoff*

Credits

Example written by Kevin Conder.

ilinrand

ilinrand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

imagecreate

imagecreate — Create an empty image of a given size.

Description

Create an empty image of a given size. Individual pixel values can then be set with. *imagegetpixel*.

Syntax

```
iimagenum imagecreate iwidth, iheight
```

Initialization

iimagenum -- number assigned to the created image.

iwidth -- desired image width.

iheight -- desired image height.

Examples

Here is an example of the imagecreate opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 245. Example of the imageload opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2
```

```
imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagesize*, *imagesave*, *imagegetpixel*, *imagesetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imagefree

imagecreate — Frees memory allocated for a previously loaded or created image.

Description

Frees memory allocated for a previously loaded or created image.

Syntax

```
imagefree iimagenum
```

Initialization

iimagenum -- reference of the image to free.

Examples

Here is an example of the imagefree opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 246. Example of the imagefree opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
```

```
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagecreate*, *imagesize*, *imagesave*, *imagegetpixel*, *imagesetpixel*

Credits

Author: Cesare Marilungo

New in version 5.08

imagegetpixel

imagegetpixel — Return the RGB pixel values of a previously opened or created image.

Description

Return the RGB pixel values of a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate*.

Syntax

```
ared agreen ablue imagegetpixel iimagenum, ax, ay
```

```
kred kgreen kblue imagegetpixel iimagenum, kx, ky
```

Initialization

iimagenum -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

Performance

ax (kx) -- horizontal pixel position (must be a float from 0 to 1).

ay (ky) -- vertical pixel position (must be a float from 0 to 1).

ared (kred) -- red value of the pixel (mapped to a float from 0 to 1).

agreen (kgreen) -- green value of the pixel (mapped to a float from 0 to 1).

ablue (kblue) -- blue value of the pixel (mapped to a float from 0 to 1).

Examples

Here is an example of the imagegetpixel opcode. It uses the files *imageopcodesdemo2.csd* [examples/imageopcodesdemo2.csd] *test1.png* [examples/test1.png] and *test2.png* [examples/test2.png].

Exemple 247. Example of the imagegetpixel opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o imageopcodesdemo2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr      =               48000
ksmps           =               100
nchnls = 2

;By Cesare Marilungo 2008
zakinit 10,1
```

```
;Load the image - should be 512x512 pixels
giimage imageload "test1.png"
;giimage imageload "test2.png" ;--try this too
giimagew, giimageh imagesize giimage

giwave ftgen 1, 0, 1024, 10, 1
gifrqs ftgen 2,0,512,-5, 1,512,10
giamps ftgen 3, 0, 512, 10, 1

        instr 100

kindex = 0
icnt = giimageh
kx_ linseg 0, p3, 1
kenv linseg 0, .2, 500, p3 - .4, 500, .2, 0

; Read a column of pixels and store the red values
; inside the table 'giamps'
loop:
    ky_ = kindex/giimageh

    ;Get the pixel color values at kx_, ky_
    kred, kgreen, kblue imagegetpixel giimage, kx_, ky_

    ;Write the red values inside the table 'giamps'
    tablew kred, kindex, giamps
    kindex = kindex+1

if (kindex < icnt) kgoto loop

; Use an oscillator bank (additive synthesis) to generate sound
; setting amplitudes for each partial according to the image
asig adsynt kenv, 220, giwave, gifrqs, giamps, icnt, 2
outs asig, asig

        endin

        instr 101
; Free memory used by the image
imagefree giimage
        endin

</CsInstruments>
<CsScore>

t 0 60

i100 1 20
i101 21 1

e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagecreate*, *imagesize*, *imagesave*, *imagesetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imageload

imageload — Load an image.

Description

Load an image and return a reference to it. Individual pixel values can then be accessed with *imagegetpixel*.

Syntax

```
iimagenum imageload filename
```

Initialization

iimagenum -- number assigned to the loaded image.

filename -- The filename of the image to load (should be a valid PNG image file).

Examples

Here is an example of the imageload opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 248. Example of the imageload opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
```

```
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imagecreate, *imagesize*, *imagesave*, *imagegetpixel*, *imagesetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imagesave

imagesave — Save a previously created image.

Description

Save a previously created image. An empty image can be created with *imagecreate* and its pixel RGB values can be set with *imagesetpixel*. The image will be saved in PNG format.

Syntax

```
imagesave iimagenum, filename
```

Initialization

iimagenum -- the reference of the image to be save. It should be a value returned by *imagecreate*.

filename -- The filename to use to save the image.

Examples

Here is an example of the imagesave opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 249. Example of the imagesave opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
```

```
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagecreate*, *imagesize*, *imagegetpixel*, *imagesetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imagesetpixel

imagesetpixel — Set the RGB value of a pixel inside a previously opened or created image.

Description

Set the RGB value of a pixel inside a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate* and saved with *imagesave*.

Syntax

```
imagegetpixel iimagenum, ax, ay, ared, agreen, ablue
```

```
imagegetpixel iimagenum, kx, ky, kred, kgreen, kblue
```

Initialization

iimagenum -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

Performance

ax (kx) -- horizontal pixel position (must be a float from 0 to 1).

ay (ky) -- vertical pixel position (must be a float from 0 to 1).

ared (kred) -- red value of the pixel (mapped to a float from 0 to 1).

agreen (kgreen) -- green value of the pixel (mapped to a float from 0 to 1).

ablue (kblue) -- blue value of the pixel (mapped to a float from 0 to 1).

Examples

Here is an example of the imagesetpixel opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 250. Example of the imagesetpixel opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
```

```
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagecreate*, *imagesize*, *imagesave*, *imagegetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imagesize

imagesize — Return the width and height of a previously opened or created image.

Description

Return the width and height of a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate*.

Syntax

```
iwidth iheight imagesize iimagenum
```

Initialization

iimagenum -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

iwidth -- image width.

iheight -- image height.

Examples

Here is an example of the imagesize opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

Exemple 251. Example of the imagesize opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin
```

```
    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

See Also

imageload, *imagecreate*, *imagesave*, *imagegetpixel*, *imagesetpixel*, *imagefree*

Credits

Author: Cesare Marilungo

New in version 5.08

imidic14

imidic14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic14*.

imidic21

imidic21 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic21*.

imidic7

imidic7 — Obsolète.

Description

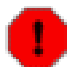
Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic7*.

in

in — Reads mono audio data from an external device or stream.

Description

Reads mono audio data from an external device or stream.



Warning

This opcode is designed to be used only with orchestras that have nchnls=1. Doing so with orchestras with nchnls > 1 will cause incorrect audio input.

Syntax

```
ar1 in
```

Performance

Reads mono audio data from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer. Any number of these opcodes can read freely from this buffer.

See Also

diskin, *inh*, *inh*, *ino*, *inq*, *ins*, *soundin*

Credits

Authors: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

Already in version 3.30

in32

in32 — Reads a 32-channel audio signal from an external device or stream.

Description

Reads a 32-channel audio signal from an external device or stream.

Warning

This opcode is designed to be used only with orchestras that have nchnls=32. Doing so with orchestras with nchnls > 32 will cause incorrect audio input.

Syntax

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
      ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \
      ar27, ar28, ar29, ar30, ar31, ar32 in32
```

Performance

in32 reads a 32-channel audio signal from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer.

Credits

inch, *inx*, *inz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

inch

inch — Reads from a numbered channel in an external audio signal or stream.

Description

Reads from a numbered channel in an external audio signal or stream.

Syntax

```
ar1 inch ksig1
```

Performance

inch reads from a numbered channel determined by *ksig1* into *a1*. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer.

Credits

in32, *inx*, *inz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

inh

inh — Reads six-channel audio data from an external device or stream.

Description

Reads six-channel audio data from an external device or stream.

Warning

This opcode is designed to be used only with orchestras that have nchnls=6. Doing so with orchestras with nchnls > 6 will cause incorrect audio input.

Syntax

```
ar1, ar2, ar3, ar4, ar5, ar6 inh
```

Performance

Reads six-channel audio data from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer. Any number of these opcodes can read freely from this buffer.

See Also

diskin, *in*, *ino*, *inq*, *ins*, *soundin*

Credits

Author: John ffitch

init

init — Met la valeur de l'expression de taux-i dans une variable de taux-k ou de taux-a.

Syntaxe

```
ares init iarg
```

```
ires init iarg
```

```
kres init iarg
```

Description

Met la valeur de l'expression de taux-i dans une variable de taux-k ou de taux-a.

Initialisation

Met la valeur de l'expression de taux-i *iarg* dans une variable de taux-k ou de taux-a, c-à-d., initialise le résultat. Noter que **init** présente le seul cas d'une instruction de la période d'initialisation autorisée à écrire dans un résultat de la période d'exécution (taux-k ou -a) ; cette instruction n'a aucun effet pendant l'exécution.

Voir Aussi

=, *divz*, *tival*

initc14

initc14 — Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

Description

Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

Syntaxe

```
initc14 ichan, ictlno1, ictlno2, ivalue
```

Initialisation

ichan -- canal MIDI (1-16)

ictlno1 -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids faible (0-127)

ivalue -- valeur décimale (doit être entre 0 et 1)

Exécution

initc14 peut être utilisé conjointement avec les opcodes *midic14* et *ctrl14* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic14* et de *ctrl14*:

ivalue = (valeur_initiale - min) / (max - min)

Voir Aussi

ctrl7, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc21*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

initc21

initc21 — Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

Description

Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

Syntaxe

```
initc21 ichan, ictlno1, ictlno2, ictlno3, ivalue
```

Initialisation

ichan -- canal MIDI (1-16)

ictlno1 -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids moyen (0-127)

ictlno3 -- numéro de contrôleur pour l'octet de poids faible (0-127)

ivalue -- valeur décimale (doit être entre 0 et 1)

Exécution

initc21 peut être utilisé conjointement avec les deux opcodes *midic21* et *ctrl21* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic21* et de *ctrl21*:

ivalue = (valeur_initiale - min) / (max - min)

Voir aussi

ctrl7, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc14*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

initc7

initc7 — Initialise le contrôleur utilisé pour créer une valeur MIDI sur 7 bit.

Description

Initialise le contrôleur MIDI *ictlno* avec *ivalue*

Syntaxe

```
initc7 ichan, ictlno, ivalue
```

Initialisation

ichan -- canal MIDI (1-16)

ictlno -- numéro du contrôleur (0-127)

ivalue -- valeur décimale (doit être entre 0 et 1)

Exécution

initc7 peut être utilisé conjointement avec les opcodes *midic7* et *ctrl7* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic7* et de *ctrl7*:

ivalue = (valeur_initiale - min) / (max - min)

Voir aussi

ctrl7, *ctrl14*, *ctrl21*, *ctrlinit*, *initc14*, *initc21*, *midic7*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

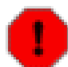
Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

ino

ino — Reads eight-channel audio data from an external device or stream.

Description

Reads eight-channel audio data from an external device or stream.



Warning

This opcode is designed to be used only with orchestras that have nchnls=8. Doing so with orchestras with nchnls > 8 will cause incorrect audio input.

Syntax

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino
```

Performance

Reads eight-channel audio data from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer. Any number of these opcodes can read freely from this buffer.

See Also

diskin, *in*, *inh*, *inh*, *inq*, *ins*, *soundin*

Credits

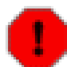
Author: John ffitch

inq

inq — Reads quad audio data from an external device or stream.

Description

Reads quad audio data from an external device or stream.



Warning

This opcode is designed to be used only with orchestras that have nchnls=4. Doing so with orchestras with nchnls > 4 will cause incorrect audio input.

Syntax

```
ar1, ar2,  ar3, a4 inq
```

Performance

Reads quad audio data from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer. Any number of these opcodes can read freely from this buffer.

See Also

diskin, *in*, *inh*, *inh*, *ino*, *ins*, *soundin*

Credits

Authors: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

inrg

inrg — Allow input from a range of adjacent audio channels from the audio input device

Description

inrg reads audio from a range of adjacent audio channels from the audio input device.

Syntax

```
inrg kstart, ain1 [,ain2, ain3, ..., ainN]
```

Performance

kstart - the number of the first channel of the input device to be accessed (channel numbers starts with 1, which is the first channel)

ain1, ain2, ... ainN - the output arguments filled with the incoming audio coming from corresponding channels.

inrg allows input from a range of adjacent channels from the input device. *kstart* indicates the first channel to be accessed (channel 1 is the first channel). The user must be sure that the number obtained by summing *kstart* plus the number of accessed channels -1 is <= *nchnls*.

Note

Note that this opcode is exceptional in that it produces its « output » on the parameters to the right.

Credits

Author: Gabriel Maldonado

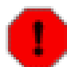
New in version 5.06

ins

ins — Reads stereo audio data from an external device or stream.

Description

Reads stereo audio data from an external device or stream.



Warning

This opcode is designed to be used only with orchestras that have nchnls=2. Doing so with orchestras with nchnls > 2 will cause incorrect audio input.

Syntax

```
ar1, ar2 ins
```

Performance

Reads stereo audio data from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer. Any number of these opcodes can read freely from this buffer.

See Also

diskin, *in*, *inh*, *inh*, *ino*, *inq*, *soundin*

Credits

Authors: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

insremot

insremot — An opcode which can be used to implement a remote orchestra. This opcode will send note events from a source machine to one destination.

Description

With the insremot and insglobal opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the score. During the performance the master machine sends the note events to the clients. The insremot opcode will send events from a source machine to one destination if you want to send events to many destinations (broadcast) use the insglobal opcode instead. These two opcodes can be used in combination.

Syntax

```
insremotidestination, isource, instrnum [,instrnum...]
```

Initialization

idestination -- a string that is the intended host computer (e.g. 192.168.0.100). This is the destination host which receives the events from the given instrument.

isource -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument and sends it to the address given by idestination.

instrnum -- a list of instrument numbers which will be played on the destination machine

Examples

Here is an example of the insremot opcode. It uses the files *insremot.csd* [examples/insremot.csd] and *insremotM.csd* [examples/insremotM.csd].

Exemple 252. Example of the insremot opcode.

The simple example below shows the bilbar example played on a remote machine. The master machine is named "192.168.1.100" and the client "192.168.1.101". Start the client on the machine (it will wait to receive the events from the master machine) and then start the master. Here is the command on linux to start a client (csound -+rtaudio=alsa -odac -dm0 insremot.csd), and the command on the master machine will look like this (csound -+rtaudio=alsa -odac -dm0 insremotM.csd).

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o insremot.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1

instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
     out      aq
endin

</CsInstruments>
<CsScore>
f0 360

e
</CsScore>
</CsoundSynthesizer>
```

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o insremotM.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1

instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
     out      aq
endin

</CsInstruments>
<CsScore>
i1 0 0.5 3 0.2 500 0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800 0.05
e
</CsScore>
</CsoundSynthesizer>
```

See also

insglobal, *midglobal*, *midremot*, *remoteport*

Credits

Author: Simon Schampijer
2006

New in version 5.03

insglobal

insglobal — An opcode which can be used to implement a remote orchestra. This opcode will send note events from a source machine to many destinations.

Description

With the *insremot* and *insglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the score. During the performance the master machine sends the note events to the clients. The *insglobal* opcode sends the events to all the machines involved in the remote concert. These machines are determined by the *insremot* definitions made above the *insglobal* command. To send events to only one machine use *insremot*.

Syntax

```
insglobalisource, instrnum [,instrnum...]
```

Initialization

isource -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument(s) and sends it to all the machines involved in the remote concert.

instrnum -- a list of instrument numbers which will be played on the destination machines

Examples

See the entry for *insremot* for an example of usage.

See also

insremot, *midglobal*, *midremot*, *remoteport*

Credits

Author: Simon Schampijer
2006

New in version 5.03

instimek

instimek — Obsolète.

Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinstk*.

Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

instimes

instimes — Obsolète.

Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinsts*.

Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

instr

instr — Commence un bloc d'instrument.

Description

Commence un bloc d'instrument.

Syntaxe

```
instr i, j, ...
```

Initialisation

Commence un bloc d'instrument, définissant les instruments *i, j*, ...

i, j, ... doivent être des nombres, pas des expressions. Tout entier positif convient, dans n'importe quel ordre, mais on préfère éviter les nombres excessivement grands.

Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

On peut définir les instruments dans n'importe quel ordre (mais ils seront toujours initialisés et exécutés par ordre de numéro d'instrument ascendant, à l'exception des notes provoquées par des évènements en temps réel, qui sont initialisées dans l'ordre où elles sont reçues mais toujours exécutées par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).

Exécution

Appeler un Instrument depuis un Instrument

On peut appeler un instrument depuis un instrument comme si c'était un opcode soit au moyen de l'opcode *subinstr* soit en spécifiant un instrument avec un nom textuel :

```
instr MonOscil
...
endin
```

Si un instrument est défini avec un nom, on peut l'appeler directement comme un opcode :

```
asig MonOscil iamp, ihaut, iftable
```

Par défaut, tous les paramètres de sortie correspondent aux sorties de l'instrument exprimées par des opcodes de *sortie de signal*. Tous les paramètres d'entrée sont affectés aux p-champs de l'instrument appelé

en commençant par le quatrième, p4. Les valeurs des deuxième et troisième p-champs de l'instrument appelé, p2 et p3, sont automatiquement fixés à la même valeur que ceux de l'instrument appelant.

Un intrument nommé doit être défini avant les instrument qui l'appellent.

Conseils

Si vous utiliser l'opcode *outc*, vous pouvez créer un instrument qui pourra être compilé et fontionner dans des orchestres avec n'importe quel nombre de canaux plus grand au égal ou nombre de canaux de sortie de cet instrument.

Il est intéressant d'utiliser la fonctionnalité *#include* avec les instruments nommés. Vous pouvez définir vos instruments nommés dans des fichiers séparés, et utiliser #include lorsque vous en avez besoin.

Exemples

Voici un exemple de l'opcode *instr*. Il utilise le fichier *instr.csd* [examples/instr.csd].

Exemple 253. Exemple de l'opcode *instr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o instr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

endin, *in*, *out*, *opcode*, *endop*, *setksmps*, *xin*, *xout*, *subinstr*, *subinstrinit*

Crédits

Exemple écrit par Kevin Conder.

int

int — Extrait la partie entière d'un nombre décimal.

Description

Retourne la partie entière de *x*.

Syntaxe

```
int(x)  (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode int. Il utilise le fichier *int.csd* [examples/int.csd].

Exemple 254. Exemple de l'opcode int.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o int.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 16 / 5
  i2 = int(i1)

  print i2
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i2 = 3.000
```

Voir Aussi

abs, *exp*, *frac*, *log*, *log10*, *i*, *sqrt*

Crédits

Exemple écrit par Kevin Conder.

integ

integ — Modify a signal by integration.

Description

Modify a signal by integration.

Syntax

```
ares integ asig [, iskip]
```

```
kres integ ksig [, iskip]
```

Initialization

iskip (optional) -- initial disposition of internal save space (see *reson*). The default value is 0.

Performance

integ and *diff* perform integration and differentiation on an input control signal or audio signal. Each is the converse of the other, and applying both will reconstruct the original signal. Since these units are special cases of low-pass and high-pass filters, they produce a scaled (and phase shifted) output that is frequency-dependent. Thus *diff* of a sine produces a cosine, with amplitude *2 * sin(pi * Hz /* sr*)* that of the original (for each component partial); *integ* will inversely affect the magnitudes of its component inputs. With this understanding, these units can provide useful signal modification.

Examples

Here is an example of the integ opcode. It uses the file *integ.csd* [examples/integ.csd].

Exemple 255. Example of the integ opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o integ.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- a differentiated signal.
instr 1
  ; Generate a band-limited pulse train.
```

```
  asrc buzz 20000, 440, 20,  1

  ; Differentiate the signal.
  adiff diff asrc

  out adiff
endin

; Instrument #2 -- a re-integrated signal.
instr 2
  ; Generate a band-limited pulse train.
  asrc buzz 20000, 440, 20,  1

  ; Differentiate the signal.
  adiff diff asrc

  ; Re-integrate the previously differentiated signal.
  a1 integ adiff

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e


</CsScore>
</CsoundSynthesizer>
```

See Also

diff, *downsamp*, *interp*, *samphold*, *upsamp*

Credits

Example written by Kevin Conder.

interp

interp — Converts a control signal to an audio signal using linear interpolation.

Description

Converts a control signal to an audio signal using linear interpolation.

Syntax

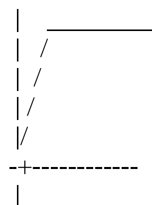
```
ares interp ksig [, iskip] [, imode]
```

Initialization

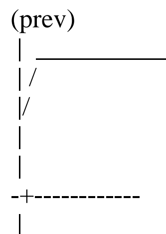
iskip (optional, default=0) -- initial disposition of internal save space (see *reson*). The default value is 0.

imode (optional, default=0) -- sets the initial output value to the first k-rate input instead of zero. The following graphs show the output of interp with a constant input value, in the original, when skipping init, and in the new mode:

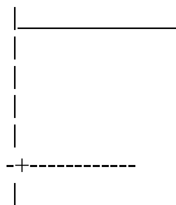
Exemple 256. iskip=0, imode=0



Exemple 257. iskip=1, imode=0



Exemple 258. iskip=0, imode=1



Performance

ksig -- input k-rate signal.

interp converts a control signal to an audio signal. It uses linear interpolation between successive kvals.

Examples

Here is an example of the interp opcode. It uses the file *interp.csd* [examples/interp.csd].

Exemple 259. Example of the interp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o interp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 8000
kr = 8
ksmps = 1000
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
  ; Create an amplitude envelope.
  kamp linseg 0, p3/2, 20000, p3/2, 0

  ; The amplitude envelope will sound rough because it
  ; jumps every ksmps period, 1000.
  a1 oscil kamp, 440, 1
  out a1
endin

; Instrument #2 - a smoother sounding instrument.
instr 2
  ; Create an amplitude envelope.
  kamp linseg 0, p3/2, 25000, p3/2, 0
  aamp interp kamp
```

```
  ; The amplitude envelope will sound smoother due to
  ; linear interpolation at the higher a-rate, 8000.
  a1 oscil aamp, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 256 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

diff, *downsamp*, *integ*, *samphold*, *upsamp*

Credits

Example written by Kevin Conder.

Updated November 2002, thanks to a note from both Rasmus Ekman and Istvan Varga.

invalue

invalue — Reads a k-rate signal from a user-defined channel.

Description

Reads a k-rate signal or string from a user-defined channel.

Syntax

```
kvalue invalue "channel name"
```

```
Sname invalue "channel name"
```

Performance

kvalue -- The k-rate value that is read from the channel.

Sname -- The string variable that is read from the channel.

"channel name" -- An integer, string (in double-quotes), or string variable identifying the channel.

See Also

outvalue

Credits

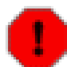
Author: Matt Ingalls

inx

inx — Reads a 16-channel audio signal from an external device or stream.

Description

Reads a 16-channel audio signal from an external device or stream.



Warning

This opcode is designed to be used only with orchestras that have nchnls=16. Doing so with orchestras with nchnls > 16 will cause incorrect audio input.

Syntax

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \
      ar13, ar14, ar15, ar16 inx
```

Performance

inx reads a 16-channel audio signal from an external device or stream. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer.

Credits

in32, *inch*, *inz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

inz

inz — Reads multi-channel audio samples into a ZAK array from an external device or stream.

Description

Reads multi-channel audio samples into a ZAK array from an external device or stream.

Syntax

```
inz ksig1
```

Performance

inz reads audio samples in *nchnls* into a ZAK array starting at *ksig1*. If the command-line *-i* flag is set, sound is read continuously from the audio input stream (e.g. *stdin* or a soundfile) into an internal buffer.

Credits

in32, *inch*, *inx*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

ioff

ioff — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteoff*.

ion

ion — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteon*.

iondur

iondur — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur*.

iondur2

iondur2 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur2*.

ioutat

ioutat — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outiat*.

ioutc

ioutc — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic*.

ioutc14

ioutc14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic14*.

ioutpat

ioutpat — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipat*.

ioutpb

ioutpb — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipb*.

ioutpc

ioutpc — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipc*.

ipcauchy

ipcauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

ipoisson

ipoisson — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

ipow

ipow — Obsolète.

Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

is16b14

is16b14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s16b14*.

is32b14

is32b14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s32b14*.

islider16

islider16 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider16*.

islider32

islider32 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider32*.

islider64

islider64 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider64*.

islider8

islider8 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider8*.

itablecopy

itablecopy — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableicopy*.

itablegpw

itablegpw — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableigpw*.

itablemix

itablemix — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableimix*.

itablew

itablew — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableiw*.

itrirand

itrirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

iunirand

iunirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

iweibull

iweibull — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

jacktransport

jacktransport — Start/stop jack_transport and can optionally relocate the playback head.

Description

Start/stop jack_transport and can optionally relocate the playback head.

Syntax

```
jacktransport icommand [, ilocation]
```

Initialization

icommand -- 1 to start playing, 0 to stop.

ilocation -- optional location in seconds to specify where the playback head should be moved. If omitted, the transport is started from current location.

Note

Since *jacktransport* depends on jack audio connection kit, it will work only on Linux or Mac OS X systems which have the jack server running.

Examples

Here is a simple example of the jacktransport opcode. It uses the file *jacktransport.csd* [examples/jacktransport.csd].

Exemple 260. Simple example of the jacktransport opcode.

```
<CsoundSynthesizer>
<CsOptions>

-+rtaudio=JACK -b 64 --sched  -o dac:system:playback_
</CsOptions>
<CsInstruments>

sr      =               44100
ksmps           =               16
nchnls = 2


        instr 1
jacktransport p4, p5
        endin

        instr 2
jacktransport p4
        endin


</CsInstruments>
<CsScore>

i2 0 5 1; play
i2 5 1 0; stop
```

```
i1 6 5 1 2 ; move at 2 seconds and start playing back
i1 11 1 0 0 ; stop and rewind

e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Cesare Marilungo

New in version 5.08

jitter

jitter — Génère aléatoirement une suite de segments de droite.

Description

Génère aléatoirement une suite de segments de droite.

Syntaxe

```
kout jitter kamp, kcpsMin, kcpsMax
```

Exécution

kamp -- Amplitude de la déviation de jitter

kcpsMin -- Vitesse minimale des variations aléatoires de fréquence (exprimée en cps)

kcpsMax -- Vitesse maximale des variations aléatoires de fréquence (exprimée en cps)

jitter génère aléatoirement une suite de segments de droite entre -*kamp* et +*kamp*. La durée de chaque segment est une valeur aléatoire générée en fonction des valeurs *kcpsmin* et *kcpsmax*.

On peut utiliser *jitter* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

Exemples

Voici un exemple de l'opcode jitter. Il utilise le fichier *jitter.csd* [examples/jitter.csd].

Exemple 261. Exemple de l'opcode jitter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o jitter.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- plain instrument.
instr 1
  aplain vco 20000, 220, 2, 0.83

  outs aplain, aplain
endin
```

```
; Instrument #2 -- instrument with jitter.
instr 2
  ; Create a signal modulated the jitter opcode.
  kamp init 2
  kcpsmin init 4
  kcpsmax init 6
  kj jitter kamp, kcpsmin, kcpsmax

  aplain vco 20000, 220, 2, 0.83
  ajitter vco 20000, 220+kj, 2, 0.83

  outs aplain, ajitter
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
; Play Instrument #2 for 3 seconds.
i 2 3 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

jitter2, *vibr*, *vibrato*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

jitter2

jitter2 — Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

Description

Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

Syntaxe

```
kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3
```

Exécution

ktotamp -- Amplitude résultante de jitter2

kamp1 -- Amplitude du premier composant de jitter

kcps1 -- Vitesse de la variation aléatoire du premier composant de jitter (exprimée en cps)

kamp2 -- Amplitude du second composant de jitter

kcps2 -- Vitesse de la variation aléatoire du second composant de jitter (exprimée en cps)

kamp3 -- Amplitude du troisième composant de jitter

kcps3 -- Vitesse de la variation aléatoire du troisième composant de jitter (exprimée en cps)

jitter2 génère une ligne segmentée comme *jitter*, mais ici le résultat est semblable à la somme de trois opcodes *randi*, chacun avec ses propres valeurs d'amplitude et de fréquence (voir *randi* pour plus de détails), qui sont modifiables au taux-k. On peut obtenir différents effets en variant les arguments en entrée.

On peut utiliser *jitter2* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

Exemples

Voici un exemple de l'opcode jitter2. Il utilise le fichier *jitter2.csd* [examples/jitter2.csd].

Exemple 262. Exemple de l'opcode jitter2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o jitter2.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- plain instrument.
instr 1
  aplain vco 20000, 220, 2, 0.83

  outs aplain, aplain
endin

; Instrument #2 -- instrument with jitter.
instr 2
  ; Create a signal modulated with the jitter2 opcode.
  ktotamp init 2
  kamp1 init 0.66
  kcps1 init 3
  kamp2 init 0.66
  kcps2 init 3
  kamp3 init 0.66
  kcps3 init 3
  kj jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, \
             kamp3, kcps3

  aplain vco 20000, 220, 2, 0.83
  ajitter vco 20000, 220+kj, 2, 0.83

  outs aplain, ajitter
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
; Play Instrument #2 for 3 seconds.
i 2 3 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

jitter, *vibr*, *vibrato*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

jspline

jspline — Un générateur de spline avec gigue.

Description

Un générateur de spline avec gigue.

Syntaxe

```
ares jspline xamp, kcpsMin, kcpsMax
```

```
kres jspline kamp, kcpsMin, kcpsMax
```

Exécution

kres, ares -- Signal de sortie.

xamp -- Facteur d'amplitude.

kcpsMin, kcpsMax -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

jspline (générateur de spline avec gigue) génère une courbe lisse basée sur des points aléatoires engendrés au taux [*cpsMin*, *cpsMax*]. Cet opcode est semblable à *randomi* ou à *randi* ou à *jitter*, toutefois les segments ne sont pas des lignes droites, mais des courbes splines cubiques. Les valeurs de sortie sont approximativement comprises entre -*xamp* et *xamp*. Dans la réalité, l'intervalle peut être un peu plus grand, à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin*-*cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

k

k — Convertit un paramètre de taux-i en une valeur de taux-k.

Description

Convertit une valeur de taux-i en une valeur de taux-k, par exemple pour une utilisation avec *rnd()* et *birnd()* pour générer des nombres aléatoires au taux-k.

Syntaxe

```
k(x) (arguments de taux-i seulement)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Voir Aussi

i a

Crédits

Auteur : Istvan Varga

Nouveau dans la version 5.00 de Csound

kbetarand

kbetarand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

kbexprnd

kbexprnd — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

kcauchy

kcauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

kdump

kdump — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk*.

kdump2

kdump2 — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk2*.

kdump3

kdump3 — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk3*.

kdump4

kdump4 — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk4*.

kexprand

kexprand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

kfilter2

kfilter2 — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *filter2*.

Crédits

Auteur : Michael A. Casey
M.I.T.
Cambridge, Mass.
1997

Nouveau dans la version 3.47

kgauss

kgauss — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

kgoto

kgoto — Transfer control during the p-time passes.

Description

During the p-time passes only, unconditionally transfer control to the statement labeled by *label*.

Syntax

```
kgoto label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

Examples

Here is an example of the kgoto opcode. It uses the file *kgoto.csd* [examples/kgoto.csd].

Exemple 263. Example of the kgoto opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o kgoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Change kval linearly from 0 to 2 over
  ; the period set by the third p-field.
  kval line 0, p3, 2

  ; If kval is greater than or equal to 1 then play the high note.
  ; If not then play the low note.
  if (kval >= 1) kgoto highnote
    kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit

playit:
  ; Print the values of kval and kfreq.
```

```
  printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

  a1 oscil 10000, kfreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

See Also

cggoto, *cigoto*, *ckgoto*, *goto*, *if*, *igoto*, *tigoto*, *timout*

Credits

Example written by Kevin Conder.

Added a note by Jim Aikin.

klinrand

klinrand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

kon

kon — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *midion*.

koutat

koutat — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkat*.

koutc

koutc — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc*.

koutc14

koutc14 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc14*.

koutpat

koutpat — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpat*.

koutpb

koutpb — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpb*.

koutpc

koutpc — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpc*.

kpcauchy

kpcauchy — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

kpoisson

kpoisson — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

kpow

kpow — Obsolète.

Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

kr

kr — Fixe le taux de contrôle.

Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

Syntaxe

```
kr = iarg
```

Initialisation

kr = (facultatif) -- fixe le taux de contrôle à *iarg* échantillons par seconde. La valeur par défaut est 1000.

De plus, toute *variable globale* [54] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *kr*. Csound utilisera les valeurs par défaut, ou calculera *kr* à partir des valeurs définies de *ksmps* et *sr*. Habituellement, il est mieux de ne spécifier que *ksmps* et *sr* et de laisser csound calculer *kr*.

Exemples

```
sr = 10000
kr = 500
ksmps = 20
gi1 = sr/2.
ga init 0
itranspose = octpch(.01)
```

Voir Aussi

ksmps, *nchnls*, *sr*

kread

kread — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk*.

kread2

kread2 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk2*.

kread3

kread3 — Obsolète.

Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk3*.

kread4

kread4 — Obsolète.

Description

Obsolète depuis la version 3.52. Use the *readk4* opcode instead.

ksmps

ksmps — Fixe le nombre d'échantillons dans une période de contrôle.

Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

Syntaxe

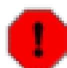
```
ksmps = iarg
```

Initialisation

ksmps = (facultatif) -- fixe le nombre d'échantillons dans une période de contrôle. Cette valeur doit être égale à *sr/kr*. La valeur par défaut est 10.

De plus, toute *variable globale* [54] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *ksmps*. Csound essaiera de calculer la valeur omise à partir des valeurs spécifiées pour *sr* et *kr*, mais le résultat devra être un nombre entier.



Avertissement

ksmps doit avoir une valeur entière.

Exemples

```
sr = 10000
kr = 500
ksmps = 20
gi1 = sr/2.
ga init 0
itranspose = octpch(.01)
```

Voir Aussi

kr, *nchnls*, *sr*

Crédits

Grâce à une note de Gabriel Maldonado, un avertissement sur les valeurs entières a été ajouté.

ktableseg

ktableseg — Deprecated.

Description

Deprecated. Use the *tableseg* opcode instead.

Syntax

```
ktableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

ktrirand

ktrirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

kunirand

kunirand — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

kweibull

kweibull — Obsolète.

Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

lfo

lfo — Un oscillateur basse fréquence avec différentes formes d'onde.

Description

Un oscillateur basse fréquence avec différentes formes d'onde.

Syntaxe

```
kres lfo kamp, kcps [, itype]
```

```
ares lfo kamp, kcps [, itype]
```

Initialisation

itype (facultatif, par défaut 0) -- détermine la forme d'onde de l'oscillateur. La valeur par défaut est 0.

- *itype* = 0 - sinus
- *itype* = 1 - triangle
- *itype* = 2 - carrée (bipolaire)
- *itype* = 3 - carrée (unipolaire)
- *itype* = 4 - dent de scie
- *itype* = 5 - dent de scie (vers le bas)

L'onde sinus est implémentée comme une table de 4096 éléments avec interpolation linéaire. Les autres sont calculées.

Exécution

kamp -- amplitude de la sortie

kcps -- fréquence de l'oscillateur

Exemples

Voici un exemple de l'opcode lfo. Il utilise le fichier *lfo.csd* [examples/lfo.csd].

Exemple 264. Exemple de l'opcode lfo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lfo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10
  kcps = 5
  itype = 4

  k1 lfo kamp, kcps, itype
  ar oscil p4, p5+k1, 1
  out ar
endin


</CsInstruments>
<CsScore>

; Table #1: an ordinary sine wave.
f 1 0 32768 10 1

; p4 = amplitude of the output signal.
; p5 = frequency (in cycles per second) of the output signal.
; Play Instrument #1 for two seconds.
i 1 0 2 10000 220
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Novembre 1998

Nouveau dans la version 3.491 de Csound

limit

limit — Sets the lower and upper limits of the value it processes.

Description

Sets the lower and upper limits of the value it processes.

Syntax

```
ares limit asig, klow, khigh
```

```
ires limit isig, ilow, ihigh
```

```
kres limit ksig, klow, khigh
```

Initialization

isig -- input signal

ilow -- low threshold

ihigh -- high threshold

Performance

xsig -- input signal

klow -- low threshold

khigh -- high threshold

limit sets the lower and upper limits on the *xsig* value it processes. If *xhigh* is lower than *xlow*, then the output will be the average of the two - it will not be affected by *xsig*.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals.

See Also

mirror, *wrap*

Credits

Author: Robin Whittle
Australia

New in Csound version 3.46

line

line — Trace un segment de droite entre les points spécifiés.

Description

Trace un segment de droite entre les points spécifiés.

Syntaxe

```
ares line ia, idur, ib
```

```
kres line ia, idur, ib
```

Initialisation

ia -- valeur initiale.

ib -- valeur après *idur* secondes.

idur -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

Exécution

line génère des signaux de contrôle ou des signaux audio dont les valeurs évoluent linéairement depuis une valeur initiale jusqu'à une valeur finale.

Note

Une erreur habituelle avec cet opcode est de croire que la valeur de *ib* est tenue après la durée *idur*. *line* ne s'arrête pas automatiquement à la fin de la durée donnée. Si la longueur de votre note dépasse *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas sur *ib*, mais au contraire il continuera à monter ou à descendre à la même vitesse. Si l'on a besoin d'une pente ascendante (ou descendante) suivie d'une tenue il faut utiliser l'opcode *linseg*.

Exemples

Voici un exemple de l'opcode line. Il utilise le fichier *line.csd* [examples/line.csd].

Exemple 265. Exemple de l'opcode line.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o line.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Define kcps as a frequency value that linearly declines
  ; from 880 to 220. It declines over the period set by p3.
  kcps line 880, p3, 220

  a1 oscil 20000, kcps, 1
  out a1
endin

instr 2
  kcps line 880, 1, 660   ; kcps won't stop at 660 if p3 > 1
  a1 oscil 20000, kcps, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2

; Play Instrument #2 for two seconds.
i 2 3 2

e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

expon, *expseg*, *expsegr*, *linseg*, *linsegr*

Crédits

Exemple écrit par Kevin Conder.

linen

linen — Applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

Description

linen -- applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

Syntaxe

```
ares linen xamp, irise, idur, idec
```

```
kres linen kamp, irise, idur, idec
```

Initialisation

irise -- durée de l'attaque en secondes. Un valeur nulle ou négative signifie pas d'attaque.

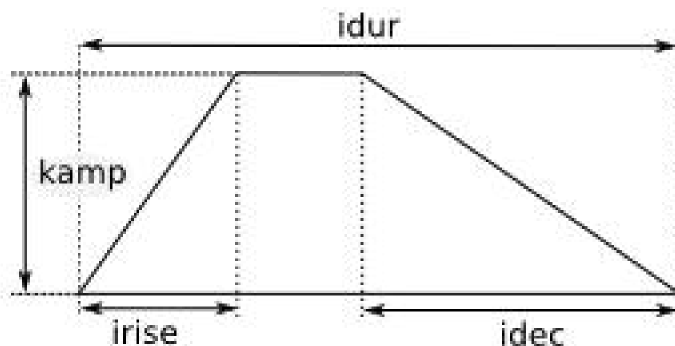
idur -- durée totale en secondes. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

idec -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

Exécution

kamp, xamp -- signal d'amplitude en entrée.

L'attaque est appliquée pendant les *irise* premières secondes, et la chute à partir de *idur - idec*. Si ces périodes sont séparées dans le temps il y aura un entretien durant lequel *amp* ne sera pas modifié. Si l'attaque et la chute de *linen* se chevauchent, les deux modifications agiront en même temps pendant cette période. Si la durée totale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, devenant négative.



Enveloppe générée par l'opcode *linen*

Note

Il est faux de croire que la valeur 0 sera tenue après la fin de l'enveloppe à *idur* secondes. *linen* ne se termine pas automatiquement à la fin de la durée donnée. Si la longueur de la note est supérieure à *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas à 0, mais continuera au contraire à chuter à la même vitesse. Si l'on a besoin d'une chute suivie d'une valeur stable il vaut mieux utiliser l'opcode *linseg*.

Voir Aussi

envlpx, *envlpxr*, *linenr*

linenr

linenr — L'opcode *linen* rallongé avec un segment de relâchement.

Description

linenr -- comme *linen* sauf que le dernier segment n'est entamé qu'après la détection d'un relâchement de note MIDI. La note est alors rallongée de la durée de la chute.

Syntaxe

```
ares linenr xamp, irise, idec, iatdec
```

```
kres linenr kamp, irise, idec, iatdec
```

Initialisation

irise -- durée de l'attaque en secondes. Un valeur nulle ou négative signifie pas d'attaque.

idec -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

iatdec -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulle ou négatives sont interdites.

Exécution

kamp, xamp -- signal d'amplitude en entrée.

Ce qui rend unique *linenr* dans Csound c'est qu'il contient un *détecteur de note-off* et un *allongement de la durée de relâchement*. Lorqu'il détecte la fin d'un évènement de partition ou un note-off MIDI, il allonge immédiatement la durée d'exécution de l'instrument courant de *idec* secondes, puis il exécute une chute exponentielle vers le facteur *iatdec*. S'il y a plusieurs unités dans un instrument, l'allongement est défini par le plus grand *idec*.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linenr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des évènements MIDI note-off provoqués par une vélocité nulle.

Voir Aussi

linsegr, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *envlpx*, *linen*, *xtratim*

lineto

lineto — Génère un glissando à partir d'un signal de contrôle.

Description

Génère un glissando à partir d'un signal de contrôle.

Syntaxe

```
kres lineto ksig, ktime
```

Exécution

kres -- Signal de sortie.

ksig -- Signal d'entrée.

ktime -- Durée du glissando en secondes.

lineto ajoute un glissando (c-à-d des segments de droite) à un signal d'entrée en escalier (produit par exemple par *randh* ou par *lpshold*). Il génère un segment de droite allant d'un degré à l'autre en *ktime* secondes. Lorsque le degré suivant est atteint, cette valeur est maintenue jusqu'à ce qu'un nouveau degré apparaisse. Il faut s'assurer que la valeur de l'argument *ktime* est inférieure à l'intervalle de temps entre deux degrés consécutifs du signal original, sinon des discontinuités apparaitront dans le signal de sortie.

Lorsqu'on l'utilise avec la sortie de *lpshold*, on obtient une simulation de l'effet de glissando des vieux synthétiseurs analogiques.

Note

Une nouvelle valeur de *ksig* ou de *ktime* n'aura d'effet qu'après que la valeur précédente de *ktime* se soit écoulée.

Voir Aussi

tlineto

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

linrand

linrand — Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares linrand krange
```

```
ires linrand krange
```

```
kres linrand krange
```

Exécution

krange -- l'intervalle des nombres aléatoires (0 - *krange*). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode linrand. Il utilise le fichier *linrand.csd* [examples/linrand.csd].

Exemple 266. Exemple de l'opcode linrand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o linrand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; krange = 1

  i1 linrand 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.394
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *pcauchy*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

linseg

linseg — Trace une suite de segments de droite entre les points spécifiés.

Description

Trace une suite de segments de droite entre les points spécifiés.

Syntaxe

```
ares linseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
kres linseg ia, idur1, ib [, idur2] [, ic] [...]
```

Initialisation

ia -- valeur initiale.

ib, ic, etc. -- valeur après *dur1* secondes, etc.

idur1 -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

idur2, idur3, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

Note

Une erreur habituelle avec cet opcode est de croire que la dernière valeur est tenue après la durée totale. *linseg* ne s'arrête pas automatiquement à la fin de la durée totale. Si la longueur de votre note dépasse la somme de tous les *idur*, *kres* (ou *ares*) ne s'arrêtera pas sur la dernière valeur donnée, mais au contraire il continuera à monter ou à descendre à la même vitesse. On peut ajouter un segment final avec la même valeur que la précédente pour créer une valeur finale tenue.

Exemples

Voici un exemple de l'opcode linseg. Il utilise le fichier *linseg.csd* [examples/linseg.csd].

Exemple 267. Exemple de l'opcode linseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o linseg.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Create an amplitude envelope.
  kenv linseg 0, p3*0.25, 1, p3*0.75, 0
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin


instr 2
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Create an amplitude envelope.
  kenv linseg 0, 0.25, 1, 0.75, 0  ; kenv will go into negative if p3 > 1
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin

instr 3
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Create an amplitude envelope.
  kenv linseg 0, 0.25, 1, 0.75, 0, 1, 0  ; kenv will stay at 0 indefinetely at the end
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin
</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03

i 2 4  1.5 8.00 ; Notice the problem with linseg

i 3 6  1.5 8.00  ; this is the solution (instr 3)
e


</CsScore>
```

```
</CsoundSynthesizer>
```

Voir Aussi

expon, *expseg*, *expsegr*, *line*, *linsegr* *transeg*

Crédits

Exemple écrit par Kevin Conder.

linsegr

linsegr — Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

Description

Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

Syntaxe

```
ares linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kres linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

Initialisation

ia -- valeur initiale.

ib, ic, etc. -- valeur après *dur1* secondes, etc.

idur1 -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

idur2, idur3, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

irel, iz -- durée en secondes et valeur finale du segment de relâchement de la note.

Pour les versions de Csound antérieures à la 5.00, le temps de relâchement ne peut pas dépasser 32767/*kr* secondes. Cette limite a été étendue à $(2^{31}-1)$/kr.

Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

linsegr fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un évènement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vélocités nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linenr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linsegr*, car la durée est allongée automatiquement.

Exemples

Voici un exemple de l'opcode linsegr. Il utilise le fichier *linsegr.csd* [examples/linsegr.csd].

Exemple 268. Exemple de l'opcode linsegr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o linsegr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Use an amplitude envelope with second-long release.
  kenv linsegr 1, p3, 0.25, 1, 0
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Make sure the score lasts for four seconds.
f 0 4

; p4 = frequency (in pitch-class notation).
; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

linsegr, *expsegr*, *envlpxr*, *mxadsr*, *madsr expon*, *expseg*, *expsega line*, *linseg*, *xtratim*

Crédits

Auteur : Barry L. Vercoe

Exemple écrit par Kevin Conder.

Décembre 2002, Décembre 2006. Merci à Istvan Varga pour l'ajout de la documentation sur le temps de relâchement maximum.

Nouveau dans Csound 3.47

locsend

locsend — Distributes the audio signals of a previous *locsig* opcode.

Description

locsend depends upon the existence of a previously defined *locsig*. The number of output signals must match the number in the previous *locsig*. The output signals from *locsend* are derived from the values given for distance and reverb in the *locsig* and are ready to be sent to local or global reverb units (see example below). The reverb amount and the balance between the 2 or 4 channels are calculated in the same way as described in the Dodge book (an essential text!).

Syntax

```
a1, a2 locsend
```

```
a1, a2,  a3, a4 locsend
```

Examples

```
  asig some audio signal
  kdegree           line    0, p3, 360
  kdistance         line    1, p3, 10
  a1, a2, a3, a4    locsig  asig, kdegree, kdistance, .1
  ar1, ar2, ar3, ar4 locsend
  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4
                    outq    a1, a2, a3, a4
endin

instr 99 ; reverb instrument
  a1                reverb2 ga1, 2.5, .5
  a2                reverb2 ga2, 2.5, .5
  a3                reverb2 ga3, 2.5, .5
  a4                reverb2 ga4, 2.5, .5
                    outq    a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
```

In the above example, the signal, *asig*, is sent around a complete circle once during the duration of a note while at the same time it becomes more and more « distant » from the listeners' location. *locsig* sends the appropriate amount of the signal internally to *locsend*. The outputs of the *locsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

locsig is useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field.

```
instr 1
  a1, a2            locsig  asig, p4, p5, .1
  ar1, ar2          locsend
  ga1=ga1+ar1
  ga2=ga2+ar2
                    outs a1, a
endin
instr 99
  ; reverb....
endin
```

A few notes:

```
  ;place the sound in the left speaker and near:
  i1 0 1 0 1

  ;place the sound in the right speaker and far:
  i1 1 1 90 25

  ;place the sound equally between left and right and in the middle ground distance:
  i1 2 1 45 12
  e
```

The next example shows a simple intuitive use of the distance value to simulate Doppler shift. The same value is used to scale the frequency as is used as the distance input to *locsig*.

```
  kdistance          line    1, p3, 10
  kfreq = (ifreq * 340) / (340 + kdistance)
  asig               oscili  iamp, kfreq, 1
  kdegree            line    0, p3, 360
  a1, a2, a3, a4     locsig  asig, kdegree, kdistance, .1
  ar1, ar2, ar3, ar4 locsend
```

See Also

locsig

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48

locsig

locsig — Takes and input signal and distributes between 2 or 4 channels.

Description

locsig takes an input signal and distributes it among 2 or 4 channels using values in degrees to calculate the balance between adjacent channels. It also takes arguments for distance (used to attenuate signals that are to sound as if they are some distance further than the loudspeaker itself), and for the amount the signal that will be sent to reverberators. This unit is based upon the example in the Charles Dodge/Thomas Jerse book, *Computer Music*, page 320.

Syntax

```
a1, a2 locsig asig, kdegree, kdistance, kreverbsend
```

```
a1, a2,  a3, a4 locsig asig, kdegree, kdistance, kreverbsend
```

Performance

kdegree -- value between 0 and 360 for placement of the signal in a 2 or 4 channel space configured as: a1=0, a2=90, a3=180, a4=270 (kdegree=45 would balanced the signal equally between a1 and a2). *locsig* maps *kdegree* to sin and cos functions to derive the signal balances (ie.: asig=1, kdegree=45, a1=a2=.707).

kdistance -- value >= 1 used to attenuate the signal and to calculate reverb level to simulate distance cues. As *kdistance* gets larger the sound should get softer and somewhat more reverberant (assuming the use of *locsend* in this case).

kreverbsend -- the percentage of the direct signal that will be factored along with the distance and degree values to derive signal amounts that can be sent to a reverb unit such as reverb, or reverb2.

Examples

```
  asig some audio signal
  kdegree            line    0, p3, 360
  kdistance          line    1, p3, 10
  a1, a2, a3, a4     locsig  asig, kdegree, kdistance, .1
  ar1, ar2, ar3, ar4 locsend
  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4
                     outq    a1, a2, a3, a4
endin

instr 99 ; reverb instrument
  a1                 reverb2 ga1, 2.5, .5
  a2                 reverb2 ga2, 2.5, .5
  a3                 reverb2 ga3, 2.5, .5
  a4                 reverb2 ga4, 2.5, .5
                     outq    a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
```

In the above example, the signal, *asig*, is sent around a complete circle once during the duration of a note while at the same time it becomes more and more "distant" from the listeners' location. *locsig* sends the appropriate amount of the signal internally to *locsend*. The outputs of the *locsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

locsig is useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field.

```
instr 1
  a1, a2            locsig  asig, p4, p5, .1
  ar1, ar2          locsend
  ga1=ga1+ar1
  ga2=ga2+ar2
                    outs a1, a
endin
instr 99
  ; reverb....
endin
```

A few notes:

```
  ;place the sound in the left speaker and near:
  i1 0 1 0 1

  ;place the sound in the right speaker and far:
  i1 1 1 90 25

  ;place the sound equally between left and right and in the middle ground distance:
  i1 2 1 45 12
  e
```

The next example shows a simple intuitive use of the distance value to simulate Doppler shift. The same value is used to scale the frequency as is used as the distance input to *locsig*.

```
  kdistance          line    1, p3, 10
  kfreq = (ifreq * 340) / (340 + kdistance)
  asig               oscili  iamp, kfreq, 1
  kdegree            line    0, p3, 360
  a1, a2, a3, a4     locsig  asig, kdegree, kdistance, .1
  ar1, ar2, ar3, ar4 locsend
```

See Also

locsend

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48

log

log — Retourne un logarithme naturel.

Description

Retourne le logarithme naturel de *x* (*x* strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

Syntaxe

```
log(x) (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode log. Il utilise le fichier *log.csd* [examples/log.csd].

Exemple 269. Exemple de l'opcode log.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log(8)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 2.079
```

Voir Aussi

abs, *exp*, *frac*, *int*, *log10*, *i*, *sqrt*

Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

log10

log10 — Retourne un logarithme en base 10.

Description

Retourne le logarithme en base 10 de *x* (*x* strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

Syntaxe

```
log10(x) (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode log10. Il utilise le fichier *log10.csd* [examples/log10.csd].

Exemple 270. Exemple de l'opcode log10.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log10.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log10(8)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 0.903
```

Voir Aussi

abs, *exp*, *frac*, *int*, *log*, *i*, *sqrt*

Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

logbtwo

logbtwo — Calcule le logarithme en base deux.

Description

Calcule le logarithme en base deux.

Syntaxe

```
logbtwo(x)  (argument au taux d'initialisation ou de contrôle seulement)
```

Exécution

logbtwo() retourne le logarithme en base deux de *x*. L'intervalle des valeurs permises en argument va de 0,25 à 4 (c-à-d une réponse comprise entre -2 et +2 octaves). Cette fonction est l'inverse de *powoftwo*().

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

Exemples

Voici un exemple de l'opcode logbtwo. Il utilise le fichier *logbtwo.csd* [examples/logbtwo.csd].

Exemple 271. Exemple de l'opcode logbtwo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o logbtwo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = logbtwo(3)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.585
```

Voir Aussi

powoftwo

Crédits

Auteur : Gabriel Maldonado
Italie
Juin 1998

Auteur : John ffitch
Université de Bath, Codemist, Ltd.
Bath, UK
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

logcurve

logcurve — Cet opcode implémente une formule qui génère une courbe logarithmique normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

Description

Génère une courbe logarithmique dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lévera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

```
log(x * (y-1)+1) / (log(y)
```

où x est égal à *kindex* et y est égal à *ksteepness*.

Syntaxe

```
kout logcurve kindex, ksteepness
```

Exécution

kindex -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

ksteepness -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

kout -- Sortie pondérée.

Exemples

Voici un exemple de l'opcode logcurve. Il utilise le fichier *logcurve.csd* [examples/logcurve.csd].

Exemple 272. Exemple de l'opcode logcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent
-odac           -idac     -d    ;;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

/*--- ---*/

                instr 1 ; logcurve test

kmod phasor 1/200
```

```
kout logcurve kmod, 2

        printk2 kmod
        printk2 kout

                endin

/*--- ---*/
</CsInstruments>
<CsScore>

i1 0 8888

e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

scale, *gainslider*, *expcurve*

Crédits

Auteur : David Akbari
Octobre
2006

loop_ge

loop_ge — Looping constructions.

Description

Construction of looping operations.

Syntax

```
loop_ge   indx, idecr, imin, label
```

```
loop_ge   kndx, kdecr, kmin, label
```

Initialization

indx -- i-rate variable to count loop.

idecr -- value to decrement the loop.

imin -- minimum value of loop index.

Performance

kndx -- k-rate variable to count loop.

kdecr -- value to decrement the loop.

kmin -- minimum value of loop index.

The actions of **loop_ge** are equivalent to the code

```
indx  =  indx - idecr
if (indx >= imin) igoto label
```

or

```
kndx  =  kndx - kdecr
if (kndx >= kmin) kgoto label
```

See Also

loop_gt, *loop_le* and *loop_lt*.

Credits

Istvan Varga. 2006

New in Csound version 5.01

loop_gt

loop_gt — Looping constructions.

Description

Construction of looping operations.

Syntax

```
loop_gt   indx, idecr, imin, label

loop_gt   kndx, kdecr, kmin, label
```

Initialization

indx -- i-rate variable to count loop.

idecr -- value to decrement the loop.

imin -- minimum value of loop index.

Performance

kndx -- k-rate variable to count loop.

kdecr -- value to decrement the loop.

kmin -- minimum value of loop index.

The actions of **loop_gt** are equivalent to the code

```
indx  =  indx - idecr
if (indx > imin) igoto label
```

or

```
kndx  =  kndx - kdecr
if (kndx > kmin) kgoto label
```

See Also

loop_ge, *loop_le* and *loop_lt*.

Credits

Istvan Varga.

New in Csound version 5.01

loop_le

loop_le — Looping constructions.

Description

Construction of looping operations.

Syntax

```
loop_le   indx, incr, imax, label
```

```
loop_le   kndx, kncr, kmax, label
```

Initialization

indx -- i-rate variable to count loop.

incr -- value to increment the loop.

imax -- maximum value of loop index.

Performance

kndx -- k-rate variable to count loop.

kncr -- value to increment the loop.

kmax -- maximum value of loop index.

The actions of **loop_le** are equivalent to the code

```
indx  =  indx + incr
if (indx <= imax) igoto label
```

or

```
kndx  =  kndx + kncr
if (kndx <= kmax) kgoto label
```

See Also

loop_ge, *loop_gt* and *loop_lt*.

Credits

Istvan Varga.

New in Csound version 5.01

loop_lt

loop_lt — Looping constructions.

Description

Construction of looping operations.

Syntax

```
loop_lt   indx, incr, imax, label

loop_lt   kndx, kncr, kmax, label
```

Initialization

indx -- i-rate variable to count loop.

incr -- value to increment the loop.

imax -- maximum value of loop index.

Performance

kndx -- k-rate variable to count loop.

kncr -- value to increment the loop.

kmax -- maximum value of loop index.

The actions of **loop_lt** are equivalent to the code

```
indx  =  indx + incr
if (indx < imax) igoto label
```

or

```
kndx  =  kndx + kncr
if (kndx < kmax) kgoto label
```

See Also

loop_ge, *loop_gt* and *loop_le*.

Credits

Istvan Varga.

New in Csound version 5.01

loopseg

loopseg — Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés.

Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

Syntaxe

```
ksig loopseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

Exécution

ksig -- Signal de sortie.

kfreq -- Taux de répétition en Hz ou en fraction de Hz.

ktrig -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

ktime0...ktimeN -- Dates des points ; exprimées en fraction de cycle.

kvalue0...kvalueN -- Valeurs des points.

L'opcode *loopseg* est semblable à *linseg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions du cycle. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont negatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

Exemples

Voici un exemple de l'opcode loopseg. Il utilise le fichier *loopseg.csd* [examples/loopseg.csd].

Exemple 273. Exemple de l'opcode loopseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loopseg.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp loopseg kfreq, 0, 0, 0, 0.5, 30000, 1, 0

  a1 oscil klp, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

lpshold

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

loopsegp

loopsegp — Signaux de contrôle basés sur des segments de droite.

Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

Syntaxe

```
ksig loopsegp  kphase, kvalue0, kdur0, kvalue1 \
      [, kdur1, ... , kdurN-1, kvalueN]
```

Exécution

ksig - signal de sortie.

kphase - point de la séquence lu, exprimé en fraction d'un cycle (de 0 à 1)

kvalue0 ...kvalueN - valeurs des points.

kdur0 ...kdurN-1 - durée des points exprimée en fractions d'un cycle.

L'opcode *loopsegp* est semblable à *loopseg* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un *phaseur* pour obtenir la valeur de la phase, on aura un comportement identique à celui de *loopseg*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *loopsegp* plus puissant et plus général que *loopseg*.

Exemples

Voici un exemple de l'opcode loopsegp. Il utilise le fichier *loopsegp.csd* [examples/loopsegp.csd].

Exemple 274. Exemple de l'opcode loopsegp.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loopsegp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=2

; By Mark Van Peteghem 2008

   instr 1
iphase  = p4

kenv      linen 1, 0.1, p3, 0.1
kph_amp   phasor 2, 0
```

```
kamp      loopsegp kph_amp, 60, 1, 30, 1, 60
kamp    = ampdb(kamp)*kenv

kph_freq  phasor 2, iphase
klow_freq line 200, p3, 100
kfreq     loopsegp kph_freq, 400, 1, klow_freq, 1, 400

asig      vco2 kamp, kfreq, 2, 0.5

         outs asig, asig

   endin

</CsInstruments>
<CsScore>
i1 0 3 0
i1 + . 0.50
</CsScore>
</CsoundSynthesizer>
```

Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

lorenz

lorenz — Implémente le système d'équations de Lorenz.

Description

Implémente le système d'équations de Lorenz. Le système de Lorenz est un système dynamique chaotique qui fut utilisé à l'origine pour simuler le mouvement d'une particule dans des courants de convection et des systèmes météorologiques simplifiés. De petites différences dans les conditions initiales conduisent rapidement à des valeurs divergentes. C'est ce qu'on appelle parfois l'effet papillon. Si un papillon bat des ailes en Australie, cela aura des conséquences sur le temps en Alaska. Ce système est l'un des éléments fondateurs du développement de la théorie du chaos. Il est utile comme source audio chaotique ou comme source de modulation basse fréquence.

Syntaxe

```
ax, ay, az lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip [, iskipinit]
```

Initialisation

ix, *iy*, *iz* -- les coordonnées initiales de la particule.

iskip -- utilisé pour sauter des valeurs générées. Si *iskip* vaut 5, seulement une valeur sur cinq sera retournée. Utile pour générer des sons de hauteur plus élévée.

iskipinit (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre sera ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

Exécution

ksv -- le nombre de Prandtl ou sigma

krv -- le nombre de Rayleigh

kbv -- le rapport entre la longueur et la largeur de la boîte dans laquelle les courants de convection sont générés

kh -- le pas de progression utilisé pour le calcul approché de l'équation différentielle. On peut l'utiliser pour contrôler la hauteur du système. Des valeurs comprises entre 0,1 et 0,001 sont typiques.

Le calcul approché des équations se fait comme suit :

```
x = x + h*(s*(y - x))
y = y + h*(-x*z + r*x - y)
z = z + h*(x*y - b*z)
```

Les valeurs historiques des paramètres sont :

```
ks = 10
kr = 28
```

kb = 8/3

Exemples

Voici un exemple de l'opcode lorenz. Il utilise le fichier *lorenz.csd* [examples/lorenz.csd].

Exemple 275. Exemple de l'opcode lorenz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lorenz.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a lorenz system in 3D space.
instr 1
  ; Create a basic tone.
  kamp init 25000
  kcps init 220
  ifn = 1
  asnd oscil kamp, kcps, ifn

  ; Figure out its X, Y, Z coordinates.
  ksv init 10
  krv init 28
  kbv init 2.667
  kh init 0.0003
  ix = 0.6
  iy = 0.6
  iz = 0.6
  iskip = 1
  ax1, ay1, az1 lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip

  ; Place the basic tone within 3D space.
  kx downsamp ax1
  ky downsamp ay1
  kz downsamp az1
  idist = 1
  ift = 0
  imode = 1
  imdel = 1.018853416
  iovr = 2
  aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
                          ift, imode, imdel, iovr

  ; Convert the 3D sound to stereo.
  aleft = aw2 + ay2
  aright = aw2 - ay2

  outs aleft, aright
endin


</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1
```

```
; Play Instrument #1 for 5 seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Hans Mikelson
Février 1999

Nouveau dans la version 3.53 de Csound

lorisread

lorisread — Imports a set of bandwidth-enhanced partials from a SDIF-format data file, applying control-rate frequency, amplitude, and bandwidth scaling envelopes, and stores the modified partials in memory.

Syntax

```
lorisread ktimpnt, ifilcod, istoreidx, kfreqenv, kampenv, kbwenv[, ifadetime]
```

Description

lorisread imports a set of bandwidth-enhanced partials from a SDIF-format data file, applying control-rate frequency, amplitude, and bandwidth scaling envelopes, and stores the modified partials in memory.

Initialization

ifilcod - integer or character-string denoting a control-file derived from reassigned bandwidth-enhanced analysis of an audio signal. An integer denotes the suffix of a file loris.sdif (e.g. loris.sdif.1); a character-string (in double quotes) gives a filename, optionally a full pathname. If not a full pathname, the file is sought first in the current directory, then in the one given by the environment variable SADIR (if defined). The reassigned bandwidth-enhanced data file contains breakpoint frequency, amplitude, noisiness, and phase envelope values organized for bandwidth-enhanced additive resynthesis. The control data must conform to one of the SDIF formats that can be

Loris stores partials in SDIF RBEP frames. Each RBEP frame contains one RBEP matrix, and each row in a RBEP matrix describes one breakpoint in a Loris partial. A RBEL frame containing one RBEL matrix describing the labeling of the partials may precede the first RBEP frame in the SDIF file. The RBEP and RBEL frame and matrix definitions are included in the SDIF file's header. In addition to RBEP frames, Loris can also read and write SDIF 1TRC frames. Since 1TRC frames do not represent bandwidth-enhancement or the exact timing of Loris breakpoints, their use is not recommended. 1TRC capabilities are provided to allow interchange with programs that are unable to handle RBEP frames.

istoreidx, ireadidx, isrcidx, itgtidx are labels that identify a stored set of bandwidth-enhanced partials. lorisread imports partials from a SDIF file and stores them with the integer label istoreidx. lorismorph morphs sets of partials labeled isrcidx and itgtidx, and stores the resulting partials with the integer label istoreidx. lorisplay renders the partials stored with the label ireadidx. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

ifadetime (optional) - In general, partials exported from Loris begin and end at non-zero amplitude. In order to prevent artifacts, it is very often necessary to fade the partials in and out, instead of turning them abruptly on and off. Specification of a non-zero ifadetime causes partials to fade in at their onsets and to fade out at their terminations. This is achieved by adding two more breakpoints to each partial: one ifadetime seconds before the start time and another ifadetime seconds after the end time. (However, no breakpoint will be introduced at a time less than zero. If necessary, the onset fade time will be shortened.) The additional breakpoints at the partial onset and termination will have the same frequency and bandwidth as the first and last breakpoints in the partial, respectively, but their amplitudes will be zero. The phase of the new breakpoints will be extrapolated to preserve phase correctness. If no value is specified, ifadetime defaults to zero. Note that the fadetime may not be exact, since the partial parameter envelopes are sampled at the control rate (krate) and indexed by ktimpnt (see below), and not by real time.

Performance

lorisread reads pre-computed Reassigned Bandwidth-Enhanced analysis data from a file stored in SDIF format, as described above. The passage of time through this file is specified by ktimpnt, which represents the time in seconds. ktimpnt must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file. kfreqenv is a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave. kampenv is a control-rate scale factor that is applied to all partial amplitude envelopes. kbwenv is a control-rate scale factor that is applied to all partial bandwidth or noisiness envelopes. The bandwidth-enhanced partial data is stored in memory with a specified label for future access by another generator.

Credits

This implementation of the Loris unit generators was written by Kelly Fitz (loris@cerlsoundgroup.org [mailto:loris@cerlsoundgroup.org]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael Gogins.

lorismorph

lorismorph — Morphs two stored sets of bandwidth-enhanced partials and stores a new set of partials representing the morphed sound. The morph is performed by linearly interpolating the parameter envelopes (frequency, amplitude, and bandwidth, or noisiness) of the bandwidth-enhanced partials according to control-rate frequency, amplitude, and bandwidth morphing functions.

Syntax

```
lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv
```

Description

lorismorph morphs two stored sets of bandwidth-enhanced partials and stores a new set of partials representing the morphed sound. The morph is performed by linearly interpolating the parameter envelopes (frequency, amplitude, and bandwidth, or noisiness) of the bandwidth-enhanced partials according to control-rate frequency, amplitude, and bandwidth morphing functions.

Initialization

istoreidx, ireadidx, isrcidx, itgtidx are labels that identify a stored set of bandwidth-enhanced partials. *lorisread* imports partials from a SDIF file and stores them with the integer label istoreidx. *lorismorph* morphs sets of partials labeled *isrcidx* and *itgtidx*, and stores the resulting partials with the integer label *istoreidx*. *lorisplay* renders the partials stored with the label *ireadidx*. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

Performance

lorismorph generates a set of bandwidth-enhanced partials by morphing two stored sets of partials, the source and target partials, which may have been imported using *lorisread*, or generated by another unit generator, including another instance of *lorismorph*. The morph is performed by interpolating the parameters of corresponding (labeled) partials in the two source sounds. The sound morph is described by three control-rate morphing envelopes. *kfreqmorphenv* describes the interpolation of partial frequency values in the two source sounds. When *kfreqmorphenv* is 0, partial frequencies are obtained from the partials stored at *isrcidx*. When *kfreqmorphenv* is 1, partial frequencies are obtained from the partials at *itgtidx*. When *kfreqmorphenv* is between 0 and 1, the partial frequencies are interpolated between corresponding source and target partials. Interpolation of partial amplitudes and bandwidth (noisiness) coefficients are similarly described by *kampmorphenv* and *kbwmorphenv*.

Credits

This implementation of the Loris unit generators was written by Kelly Fitz (loris@cerlsoundgroup.org [mailto:loris@cerlsoundgroup.org]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael gogins.

lorisplay

lorisplay — renders a stored set of bandwidth-enhanced partials using the method of Bandwidth-Enhanced Additive Synthesis implemented in the Loris software, applying control-rate frequency, amplitude, and bandwidth scaling envelopes.

Syntax

```
ar lorisplay ireadidx, kfreqenv, kampenv, kbwenv
```

Description

lorisplay renders a stored set of bandwidth-enhanced partials using the method of Bandwidth-Enhanced Additive Synthesis implemented in the Loris software, applying control-rate frequency, amplitude, and bandwidth scaling envelopes.

Initialization

istoreidx, ireadidx, isrcidx, itgtidx are labels that identify a stored set of bandwidth-enhanced partials. *lorisread* imports partials from a SDIF file and stores them with the integer label istoreidx. *lorismorph* morphs sets of partials labeled *isrcidx* and *itgtidx*, and stores the resulting partials with the integer label *istoreidx*. *lorisplay* renders the partials stored with the label *ireadidx*. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

Performance

lorisplay implements signal reconstruction using Bandwidth-Enhanced Additive Synthesis. The control data is obtained from a stored set of bandwidth-enhanced partials imported from an SDIF file using *lorisread* or constructed by another unit generator such as *lorismorph*. *kfreqenv* is a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave. *kampenv* is a control-rate scale factor that is applied to all partial amplitude envelopes. kbwenv is a control-rate scale factor that is applied to all partial bandwidth or noisiness envelopes. The bandwidth-enhanced partial data is stored in memory with a specified label for future access by another generator.

Credits

This implementation of the Loris unit generators was written by Kelly Fitz (loris@cerlsoundgroup.org [mailto:loris@cerlsoundgroup.org]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael Gogins.

loscil

loscil — Read sampled sound from a table.

Description

Read sampled sound (mono or stereo) from a table, with optional sustain and release looping.

Syntax

```
ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
      [, imod2] [, ibeg2] [, iend2]
```

Initialization

ifn -- function table number, typically denoting an sampled sound segment with prescribed looping points loaded using *GEN01*. The source file may be mono or stereo.

ibas (optional) -- base frequency in *Hz* of the recorded sound. This optionally overrides the frequency given in the audio file, but is required if the file did not contain one. The default value is 261.626 Hz, i.e. middle C. (New in Csound 4.03). If this value is not known or not present, use 1 here and in kcps.

imod1, imod2 (optional, default=-1) -- play modes for the sustain and release loops. A value of 1 denotes normal looping, 2 denotes forward & backward looping, 0 denotes no looping. The default value (-1) will defer to the mode and the looping points given in the source file. Make sure you select an appropriate mode if the file does not contain this information.

ibeg1, iend1, ibeg2, iend2 (optional, dependent on *mod1, mod2*) -- begin and end points of the sustain and release loops. These are measured in *sample frames* from the beginning of the file, so will look the same whether the sound segment is monaural or stereo. If no loop points are specified, and a looping mode (imod1, imod2) is given, the file will be looped for the whole length.

Performance

ar1, ar2 -- the output at audio-rate. There is just *ar1* for mono output. However, there is both *ar1* and *ar2* for stereo output.

xamp -- the amplitude of the output signal.

kcps -- the frequency of the output signal in cycles per second.

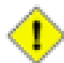
loscil samples the ftable audio at a-rate determined by *kcps*, then multiplies the result by *xamp*. The sampling increment for *kcps* is dependent on the table's base-note frequency *ibas*, and is automatically adjusted if the orchestra *sr* value differs from that at which the source was recorded. In this unit, ftable is always sampled with interpolation.

If sampling reaches the *sustain loop* endpoint and looping is in effect, the point of sampling will be modified and loscil will continue reading from within that loop segment. Once the instrument has received a *turnoff* signal (from the score or from a MIDI noteoff event), the next sustain endpoint encountered will be ignored and sampling will continue towards the *release loop* end-point, or towards the last sample (henceforth to zeros).

loscil is the basic unit for building a sampling synthesizer. Given a sufficient set of recorded piano tones,

for example, this unit can resample them to simulate the missing tones. Locating the sound source nearest a desired pitch can be done via table lookup. Once a sampling instrument has begun, its *turnoff* point may be unpredictable and require an external *release* envelope; this is often done by gating the sampled audio with *linenr*, which will extend the duration of a turned-off instrument by a specific period while it implements a decay.

If you want to loop the whole file, specify a looping mode in imod1 and do not enter any values for ibeg and iend.



Note to Windows users

Windows users typically use back-slashes, « \ », when specifying the paths of their files. As an example, a Windows user might use the path « c:\music\samples\loop001.wav ». This is problematic because back-slashes are normally used to specify special characters.

To correctly specify this path in Csound, one may alternately:

- *Use forward slashes*: c:/music/samples/loop001.wav
- *Use back-slash special characters, « \\ »*: c:\\music\\samples\\loop001.wav

Note

This is mono loscil:

```
a1 loscil 10000, 1, 1, 1 ,1
```

...and this is stereo loscil:

```
a1, a2 loscil 10000, 1, 1, 1 ,1
```

Examples

Here is an example of the loscil opcode. It uses the file *loscil.csd* [examples/loscil.csd], and *beats.wav* [examples/beats.wav].

Exemple 276. Example of the loscil opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loscil.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
   kamp = 30000
   ; If you don't know the frequency of your audio file,
   ; set both the kcps and ibas parameters equal to 1.
   kcps = 1
   ifn = 1
   ibas = 1

   a1 loscil kamp, kcps, ifn, ibas
   out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an audio file.
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the audio file several times.
i 1 0 6
e


</CsScore>
</CsoundSynthesizer>
```

See Also

loscil3 and *GEN01*

Credits

Note about the mono/stereo difference was contributed by Rasmus Ekman.

Example written by Kevin Conder.

loscil3

loscil3 — Read sampled sound from a table using cubic interpolation.

Description

Read sampled sound (mono or stereo) from a table, with optional sustain and release looping, using cubic interpolation.

Syntax

```
ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
      [, imod2] [, ibeg2] [, iend2]
```

Initialization

ifn -- function table number, typically denoting an sampled sound segment with prescribed looping points loaded using *GEN01*. The source file may be mono or stereo.

ibas (optional) -- base frequency in *Hz* of the recorded sound. This optionally overrides the frequency given in the audio file, but is required if the file did not contain one. The default value is 261.626 Hz, i.e. middle C. (New in Csound 4.03). If this value is not known or not present, use 1 here and in kcps.

imod1, imod2 (optional, default=-1) -- play modes for the sustain and release loops. A value of 1 denotes normal looping, 2 denotes forward & backward looping, 0 denotes no looping. The default value (-1) will defer to the mode and the looping points given in the source file. Make sure you select an appropriate mode if the file does not contain this information.

ibeg1, iend1, ibeg2, iend2 (optional, dependent on *mod1, mod2*) -- begin and end points of the sustain and release loops. These are measured in *sample frames* from the beginning of the file, so will look the same whether the sound segment is monaural or stereo. If no loop points are specified, and a looping mode (imod1, imod2) is given, the file will be looped for the whole length.

Performance

ar1, ar2 -- the output at audio-rate. There is just *ar1* for mono output. However, there is both *ar1* and *ar2* for stereo output.

xamp -- the amplitude of the output signal.

kcps -- the frequency of the output signal in cycles per second.

loscil3 is identical to *loscil* except that it uses cubic interpolation. New in Csound version 3.50.

Note to Windows users

Windows users typically use back-slashes, « \ », when specifying the paths of their files. As an example, a Windows user might use the path « c:\music\samples\loop001.wav ». This is problematic because back-slashes are normally used to specify special characters.

To correctly specify this path in Csound, one may alternately:

- *Use forward slashes*: c:/music/samples/loop001.wav
- *Use back-slash special characters, « \\ »*: c:\\music\\samples\\loop001.wav

Note

This is mono loscil3:

```
a1 loscil3 10000, 1, 1, 1, 1
```

...and this is stereo loscil3:

```
a1, a2 loscil3 10000, 1, 1, 1, 1
```

Examples

Here is an example of the loscil3 opcode. It uses the file *loscil3.csd* [examples/loscil3.csd], and *beats.wav* [examples/beats.wav].

Exemple 277. Example of the loscil3 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loscil3.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
   kamp = 30000
   ; If you don't know the frequency of your audio file,
   ; set both the kcps and ibas parameters equal to 1.
   kcps = 1
   ifn = 1
   ibas = 1

   a1 loscil3 kamp, kcps, ifn, ibas
   out a1
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: an audio file.
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 6
e


</CsScore>
</CsoundSynthesizer>
```

See Also

loscil and *GEN01*

Credits

Note about the mono/stereo difference was contributed by Rasmus Ekman.

Example written by Kevin Conder.

loscilx

loscilx — Loop oscillator.

Description

This file is currently a stub, but the syntax should be correct.

Syntax

```
ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
      ar15, ar16] loscilx xamp, kcps, ifn \
      [, iwsize, ibas, istrt, imod1, ibeg1, iend1]
```

See Also

sndload

loscil

Credits

Written by Istvan Varga.

2006

New in Csound 5.03

lowpass2

lowpass2 — A resonant lowpass filter.

Description

Implementation of a resonant second-order lowpass filter.

Syntax

```
ares lowpass2 asig, kcf, kq [, iskip]
```

Initialization

iskip -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal to be filtered

kcf -- cutoff or resonant frequency of the filter, measured in Hz

kq -- Q of the filter, defined, for bandpass filters, as bandwidth/cutoff. *kq* should be between 1 and 500

lowpass2 is a second order IIR lowpass filter, with k-rate controls for cutoff frequency (*kcf*) and Q (*kq*). As *kq* is increased, a resonant peak forms around the cutoff frequency, transforming the lowpass filter response into a response that is similar to a bandpass filter, but with more low frequency energy. This corresponds to an increase in the magnitude and "sharpness" of the resonant peak. For high values of *kq*, a scaling function such as *balance* may be required. In practice, this allows for the simulation of the voltage-controlled filters of analog synthesizers, or for the creation of a pitch of constant amplitude while filtering white noise.

Examples

Here is an example of the lowpass2 opcode. It uses the file *lowpass2.csd* [examples/lowpass2.csd].

Exemple 278. Example of the lowpass2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowpass2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
```

```
; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
  sr = 44100
  kr = 2205
  ksmps = 20
  nchnls = 1

          instr 1

  idur    =          p3
  ifreq   =          p4
  iamp    =          p5 * .5
  iharms  =          (sr*.4) / ifreq

; Sawtooth-like waveform
  asig    gbuzz 1, ifreq, iharms, 1, .9, 1

; Envelope to control filter cutoff
  kfreq   linseg 1, idur * 0.5, 5000, idur * 0.5, 1

  afilt   lowpass2 asig, kfreq, 30

; Simple amplitude envelope
  kenv    linseg 0, .1, iamp, idur -.2, iamp, .1, 0
          out asig * kenv

          endin


</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25

i1 0 5 100 1000
i1 5 5 200 1000
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Sean Costello
Seattle, Washington
August 1999

New in Csound version 4.0

lowres

lowres — Another resonant lowpass filter.

Description

lowres is a resonant lowpass filter.

Syntax

```
ares lowres asig, kcutoff, kresonance [, iskip]
```

Initialization

iskip -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal

kcutoff -- filter cutoff frequency point

kresonance -- resonance amount

lowres is a resonant lowpass filter derived from a Hans Mikelson orchestra. This implementation is much faster than implementing it in Csound language, and it allows *kr* lower than *sr*. *kcutoff* is not in Hz and *kresonance* is not in dB, so experiment for the finding best results.

Examples

Here is an example of the lowres opcode. It uses the file *lowres.csd* [examples/lowres.csd] and *beats.wav* [examples/beats.wav].

Exemple 279. Example of the lowres opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowres.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 5000, 440, 1

  ; Vary the cutoff frequency from 30 to 300 Hz.
  kcutoff line 30, p3, 300
  kresonance = 10

  ; Apply the filter.
  a1 lowres asig, kcutoff, kresonance

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

lowresx

Credits

Author: Gabriel Maldonado (adapted by John ffitch)
Italy

Example written by Kevin Conder.

New in Csound version 3.49

lowresx

lowresx — Simulates layers of serially connected resonant lowpass filters.

Description

lowresx is equivalent to more layers of *lowres* with the same arguments serially connected.

Syntax

```
ares lowresx asig, kcutoff, kresonance [, inumlayer] [, iskip]
```

Initialization

inumlayer -- number of elements in a *lowresx* stack. Default value is 4. There is no maximum.

iskip -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal

kcutoff -- filter cutoff frequency point

kresonance -- resonance amount

lowresx is equivalent to more layer of *lowres* with the same arguments serially connected. Using a stack of a larger number of filters allows a sharper cutoff. This is faster than using a larger number of instances of *lowres* in a Csound orchestra because only one initialization and k cycle are needed at time and the audio loop falls entirely inside the cache memory of processor. Based on an orchestra by Hans Mikelson

Examples

Here is an example of the lowresx opcode. It uses the file *lowresx.csd* [examples/lowresx.csd], and *beats.wav* [examples/beats.wav].

Exemple 280. Example of the lowresx opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowresx.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play the sawtooth waveform through a
; stack of filters.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 5000, 440, 1

  ; Vary the cutoff frequency from 30 to 300 Hz.
  kcutoff line 30, p3, 300
  kresonance = 3
  inumlayer = 2

  alr lowresx asig, kcutoff, kresonance, inumlayer

  ; It gets loud, so clip the output amplitude to 30,000.
  a1 clip alr, 1, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

lowres

Credits

Author: Gabriel Maldonado (adapted by John ffitch)
Italy

New in Csound version 3.49

lpf18

lpf18 — A 3-pole sweepable resonant lowpass filter.

Description

Implementation of a 3 pole sweepable resonant lowpass filter.

Syntax

```
ares lpf18 asig, kfco, kres, kdist
```

Performance

kfco -- the filter cutoff frequency in Hz. Should be in the range 0 to sr/2.

kres -- the amount of resonance. Self-oscillation occurs when *kres* is approximately 1. Shoujld usually be in the range 0 to 1, however, values slightly greater than 1 are possible for more sustained oscillation and an « overdrive » effect.

kdist -- amount of distortion. *kdis*t = 0 gives a clean output. *kdist* > 0 adds *tanh*() distortion controlled by the filter parameters, in such a way that both low cutoff and high resonance increase the distortion amount. Some experimentation is encouraged.

lpf18 is a digital emulation of a 3 pole (18 dB/oct.) lowpass filter capable of self-oscillation with a built-in distortion unit. It is really a 3-pole version of *moogvcf*, retuned, recalibrated and with some performance improvements. The tuning and feedback tables use no more than 6 adds and 6 multiplies per control rate. The distortion unit, itself, is based on a modified *tanh* function driven by the filter controls.

Note

This filter requires that the input signal be normalized to one.

Examples

Here is an example of the lpf18 opcode. It uses the file *lpf18.csd* [examples/lpf18.csd].

Exemple 281. Example of the lpf18 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lpf18.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a sine waveform.
  ; Note that its amplitude (kamp) ranges from 0 to 1.
  kamp init 1
  kcps init 440
  knh init 3
  ifn = 1
  asine buzz kamp, kcps, knh, ifn

  ; Filter the sine waveform.
  ; Vary the cutoff frequency (kfco) from 300 to 3,000 Hz.
  kfco line 300, p3, 3000
  kres init 0.8
  kdist init 0.3
  aout lpf18 asine, kfco, kres, kdist

  out aout * 30000
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for four seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Josep M Comajuncosas
Spain
December 2000

Example written by Kevin Conder with help from Iain Duncan. Thanks goes to Iain for helping with the example.

New in Csound version 4.10

lpfreson

lpfreson — Resynthesises a signal from the data passed internally by a previous lpread, applying formant shifting.

Description

Resynthesises a signal from the data passed internally by a previous lpread, applying formant shifting.

Syntax

```
ares lpfreson asig, kfrqratio
```

Performance

asig -- an audio driving function for resynthesis.

kfrqratio -- frequency ratio. Must be greater than 0.

lpfreson receives values internally produced by a leading *lpread.lpread* gets its values from the control file according to the input value *ktimpnt* (in seconds). If *ktimpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each k-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

lpfreson is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of lpread/lpreson (or lpfreson) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

See Also

lpread, *lpreson*

lphasor

lphasor — Generates a table index for sample playback

Description

This opcode can be used to generate table index for sample playback (e.g. tablexkt).

Syntax

```
ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istrt] [, istor]
```

Initialization

ilps -- loop start.

ilpe -- loop end (must be greater than *ilps* to enable looping). The default value of *ilps* and *ilpe* is zero.

imode (optional: default = 0) -- loop mode. Allowed values are:

- *0:* no loop
- *1:* forward loop
- *2:* backward loop
- *3:* forward-backward loop

istrt (optional: default = 0) -- The initial output value (phase). It must be less than *ilpe* if looping is enabled, but is allowed to be greater than *ilps* (i.e. you can start playback in the middle of the loop).

istor (optional: default = 0) -- skip initialization if set to any non-zero value.

Performance

ares -- a raw table index in samples (same unit for loop points). Can be used as index with the table opcodes.

xtrns -- transpose factor, expressed as a playback ratio. *ares* is incremented by this value, and wraps around loop points. For example, 1.5 means a fifth above, 0.75 means fourth below. It is not allowed to be negative.

Examples

Here is an example of the lphasor opcode. It uses the file *lphasor.csd* [examples/lphasor.csd].

Exemple 282. Example of the lphasor opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command

line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lphashor.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; Example by Jonathan Murphy Dec 2006

  sr      =  44100
  ksmps      =  10
  nchnls     =  1

    instr 1

  ifn      =  1   ; table number
  ilen      =  nsamp(ifn)    ; return actual number of samples in table
  itrns      =  1   ; no transposition
  ilps      =  0   ; loop starts at index 0
  ilpe      =  ilen ; ends at value returned by nsamp above
  imode      =  3    ; loop forwards & backwards
  istrt      =  10000  ; commence playback at index 10000 samples
  ; lphasor provides index into f1
  alphs      lphasor   itrns, ilps, ilpe, imode, istrt
  atab      tablei    alphs, ifn
             ; amplify signal
  atab      =  atab * 10000

             out       atab

    endin

</CsInstruments>
<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
i1 0 60
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Istvan Varga
January 2002
Example by: Jonathan Murphy

New in version 4.18

Updated April 2002 and November 2002 by Istvan Varga

lpinterp

lpslot, lpinterp — Computes a new set of poles from the interpolation between two analysis.

Description

Computes a new set of poles from the interpolation between two analysis.

Syntax

```
lpinterp islot1, islot2, kmix
```

Initialization

islot1 -- slot where the first analysis was stored

islot2 -- slot where the second analysis was stored

kmix -- mix value between the two analysis. Should be between 0 and 1. 0 means analysis 1 only. 1 means analysis 2 only. Any value in between will produce interpolation between the filters.

lpinterp computes a new set of poles from the interpolation between two analysis. The poles will be stored in the current *lpslot* and used by the next *lpreson* opcode.

Examples

Here is a typical orc using the opcodes:

```
ipower init 50000  ; Define sound generator
ifreq  init 440
asrc   buzz ipower,ifreq,10,1

ktime  line 0,p3,p3           ; Define time lin
       lpslot 0               ; Read square data poles
krmsr,krmso,kerr,kcps lpread    ktime,"square.pol"
       lpslot 1               ; Read triangle data poles
krmsr,krmso,kerr,kcps lpread    ktime,"triangle.pol"
kmix   line 0,p3,1            ; Compute result of mixing
       lpinterp 0,1,kmix      ; and balance power
ares   lpreson asrc
aout   balance ares,asrc
       out aout
```

See Also

lpslot

Credits

Author: Gabriel Maldonado

lposcil

lposcil, lposcil3 — Read sampled sound from a table with optional looping and high precision.

Description

Read sampled sound (mono or stereo) from a table, with optional sustain and release looping, and high precision.

Syntax

```
ares lposcil kamp, kfreqratio, kloop, kend, ifn [, iphs]
```

Initialization

ifn -- function table number

Performance

kamp -- amplitude

kfreqratio -- multiply factor of table frequency (for example: 1 = original frequency, 1.5 = a fifth up , .5 = an octave down)

kloop -- loop point (in samples)

kend -- end loop point (in samples)

lposcil (looping precise oscillator) allows varying at k-rate, the starting and ending point of a sample contained in a table (*GEN01*). This can be useful when reading a sampled loop of a wavetable, where repeat speed can be varied during the performance.

See Also

lposcil3, *lposcila*, *lposcilsa*, *lposcilsa2*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.52

lposcil3

lposcil3 — Read sampled sound from a table with high precision and cubic interpolation.

Description

Read sampled sound (mono or stereo) from a table, with optional sustain and release looping, and high precision. *lposcil3* uses cubic interpolation.

Syntax

```
ares lposcil3 kamp, kfreqratio, kloop, kend, ifn [, iphs]
```

Initialization

ifn -- function table number

Performance

kamp -- amplitude

kfreqratio -- multiply factor of table frequency (for example: 1 = original frequency, 1.5 = a fifth up , .5 = an octave down)

kloop -- loop point (in samples)

kend -- end loop point (in samples)

lposcil (looping precise oscillator) allows varying at k-rate, the starting and ending point of a sample contained in a table (*GEN01*). This can be useful when reading a sampled loop of a wavetable, where repeat speed can be varied during the performance.

See Also

lposcil

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.52

lposcila

lposcila — Read sampled sound from a table with optional looping and high precision.

Description

lposcila reads sampled sound from a table with optional looping and high precision.

Syntax

```
ar lposcila aamp, kfreqratio, kloop, kend, ift [,iphs]
```

Initialization

ift - function table number

iphs - initial phase (in samples)

Performance

ar - output signal

aamp - amplitude

kfreqratio - multiply factor of table frequency (for example: 1 = original frequency, 1.5 = a fifth up , .5 = an octave down)

kloop - loop point (in samples)

kend - end loop point (in samples)

lposcila is the same as *lposcil*, but has an audio-rate amplitude argument (instead of k-rate) to allow fast envelope transients.

See Also

lposcil, *lposcilsa*, *lposcilsa2*

Credits

Author: Gabriel Maldonado

New in version 5.06

lposcilsa

lposcilsa — Read stereo sampled sound from a table with optional looping and high precision.

Description

lposcilsa reads stereo sampled sound from a table with optional looping and high precision.

Syntax

```
ar1, ar2 lposcilsa aamp, kfreqratio, kloop, kend, ift [,iphs]
```

Initialization

ift - function table number

iphs - initial phase (in samples)

Performance

ar1, ar2 - output signal

aamp - amplitude

kfreqratio - multiply factor of table frequency (for example: 1 = original frequency, 1.5 = a fifth up , .5 = an octave down)

kloop - loop point (in samples)

kend - end loop point (in samples)

lposcilsa is the same as *lposcila*, but works with stereo files loaded with *GEN01*.

See Also

lposcil, *lposcila*, *lposcilsa2*

Credits

Author: Gabriel Maldonado

New in version 5.06

lposcilsa2

lposcilsa2 — Read stereo sampled sound from a table with optional looping and high precision.

Description

lposcilsa2 reads stereo sampled sound from a table with optional looping and high precision.

Syntax

```
ar1, ar2 lposcilsa2 aamp, kfreqratio, kloop, kend, ift [,iphs]
```

Initialization

ift - function table number

iphs - initial phase (in samples)

Performance

ar1, ar2 - output signal

aamp - amplitude

kfreqratio - multiply factor of table frequency (for example: 1 = original frequency, 2 = an octave up). Only integers are allowed

kloop - loop point (in samples)

kend - end loop point (in samples)

lposcilsa2 is the same as *lposcilsa*, but no interpolation is implemented and only works with integer *kfreqratio* values. Much faster than *lposcilsa*, it is mainly intended to be used with *kfreqratio* = 1, being in this case a fast substitute of *soundin*, since the soundfile must be entirely loaded in memory.

See Also

lposcil, *lposcila*, *lposcilsa*

Credits

Author: Gabriel Maldonado

New in version 5.06

lpread

lpread — Reads a control file of time-ordered information frames.

Description

Reads a control file of time-ordered information frames.

Syntax

```
krmsr, krmso, kerr, kcps lpread ktimpnt, ifilcod [, inpoles] [, ifrmrate]
```

Initialization

ifilcod -- integer or character-string denoting a control-file (reflection coefficients and four parameter values) derived from n-pole linear predictive spectral analysis of a source audio signal. An integer denotes the suffix of a file *lp.m*; a character-string (in double quotes) gives a filename, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in that of the environment variable SADIR (if defined). Memory usage depends on the size of the file, which is held entirely in memory during computation but shared by multiple calls (see also *adsyn*, *pvoc*).

inpoles (optional, default=0) -- number of poles in the lpc analysis. It is required only when the control file does not have a header; it is ignored when a header is detected.

ifrmrate (optional, default=0) -- frame rate per second in the lpc analysis. It is required only when the control file does not have a header; it is ignored when a header is detected.

Performance

lpread accesses a control file of time-ordered information frames, each containing n-pole filter coefficients derived from linear predictive analysis of a source signal at fixed time intervals (e.g. 1/100 of a second), plus four parameter values:

krmsr -- root-mean-square (rms) of the residual of analysis

krmso -- rms of the original signal

kerr -- the normalized error signal

kcps -- pitch in Hz

ktimpnt -- The passage of time, in seconds, through the analysis file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

lpread gets its values from the control file according to the input value *ktimpnt* (in seconds). If *ktimpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each k-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for

noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

lpfreson is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of lpread/lpreson (or lpfreson) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

See Also

lpfreson, *lpreson*, *LPANAL*

lpreson

lpreson — Resynthesises a signal from the data passed internally by a previous lpread.

Description

Resynthesises a signal from the data passed internally by a previous lpread.

Syntax

```
ares lpreson asig
```

Performance

asig -- an audio driving function for resynthesis.

lpreson receives values internally produced by a leading *lpread.lpread* gets its values from the control file according to the input value *ktimpnt* (in seconds). If *ktimpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each k-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

lpfreson is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of lpread/lpreson (or lpfreson) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

See Also

lpfreson, *lpread*

lpshold

lpshold — Génère un signal de contrôle constitué de segments tenus.

Description

Génère un signal de contrôle constitué de segments tenus délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

Syntaxe

```
ksig lpshold kfreq, ktrig, ktime0, kvalue0  [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

Exécution

ksig -- Signal de sortie.

kfreq -- Taux de répétition en Hz ou en fraction de Hz.

ktrig -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

ktime0...ktimeN -- Dates des points ; exprimées en fraction de cycle.

kvalue0...kvalueN -- Valeurs des points.

lpshold est semblable à *loopseg*, mais il ne peut générer que des segments horizontaux, car il maintient une valeur constante pendant chaque intervalle de temps placé entre *ktimeN* et *ktimeN+1*. Il est utile, entre autres, pour un contrôle mélodique comme celui des vieux séquenceurs analogiques.

Exemples

Voici un exemple de l'opcode lpshold. Il utilise le fichier *lpshold.csd* [examples/lpshold.csd].

Exemple 283. Exemple de l'opcode lpshold.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lpshold.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp lpshold kfreq, 0, 0, 0, p3*0.25, 20000, p3*0.75, 0

  a1 oscil klp, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

loopseg

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

lpsholdp

lpsholdp — Signaux de contrôle basés sur des segments tenus.

Description

Génère un signal de contrôle constitué de segments de droite tenus délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

Syntaxe

```
ksig lpsholdp  kphase, ktrig, ktime0, kvalue0  [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

Exécution

ksig - signal de sortie.

kphase -

kvalue0 ...kvalueN - valeurs des points.

ktime0 ...ktimeN - dates des points exprimées en fractions d'un cycle.

L'opcode *lpsholdp* est semblable à *lpshold* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un phaseur pour obtenir la valeur de la phase, on aura un comportement identique à *lpshold*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *lpsholdp* plus puissant et plus général que *lpshold*.

Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

lpslot

lpslot — Selects the slot to be use by further lp opcodes.

Description

Selects the slot to be use by further lp opcodes.

Syntax

```
lpslot islot
```

Initialization

islot -- number of slot to be selected.

Performance

lpslot selects the slot to be use by further lp opcodes. This is the way to load and reference several analyses at the same time.

Examples

Here is a typical orc using the opcodes:

```
ipower init 50000  ; Define sound generator
ifreq  init 440
asrc   buzz ipower,ifreq,10,1

ktime  line 0,p3,p3          ; Define time lin
       lpslot 0              ; Read square data poles
krmsr,krmso,kerr,kcps lpread    ktime,"square.pol"
       lpslot 1              ; Read triangle data poles
krmsr,krmso,kerr,kcps lpread    ktime,"triangle.pol"
kmix   line 0,p3,1           ; Compute result of mixing
       lpinterp 0,1,kmix     ; and balance power
ares   lpreson asrc
aout   balance ares,asrc
       out aout
```

See Also

lpinterp

Credits

Author: Mark Resibois
Brussels
1996

New in version 3.44

mac

mac — Multiplies and accumulates a- and k-rate signals.

Description

Multiplies and accumulates a- and k-rate signals.

Syntax

```
ares mac asig1, ksig1 [, asig2] [, ksig2] [, asig3] [, ksig3] [...]
```

Performance

ksig1, etc. -- k-rate input signals

asig1, etc. -- a-rate input signals

mac multiplies and accumulates a- and k-rate signals. It is equivalent to:

ares = asig1*ksig1 + asig2*ksig2 + asig3*ksig3 + ...

See Also

maca

Credits

Author: John ffitch
University of Bath, Codemist, Ltd.
Bath, UK
May 1999

New in Csound version 3.54

maca

maca — Multiply and accumulate a-rate signals only.

Description

Multiply and accumulate a-rate signals only.

Syntax

```
ares maca asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]
```

Performance

asig1, asig2, ... -- a-rate input signals

maca multiplies and accumulates a-rate signals only. It is equivalent to:

ares = asig1*asig2 + asig3*asig4 + asig5*asig6 + ...

See Also

mac

Credits

Author: John ffitch
University of Bath, Codemist, Ltd.
Bath, UK
May 1999

New in Csound version 3.54

madsr

madsr — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

Syntaxe

```
ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

Initialisation

iatt -- durée de l'attaque (attack)

idec -- durée de la première chute (decay)

islev -- niveau d'entretien (sustain)

irel -- durée de la chute (release)

idel -- délai de niveau zéro avant le démarrage de l'enveloppe

ireltim (facultatif, par défault = -1) -- Contrôle la durée du relâchement après la réception d'un évènement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

Noter que la durée du relâchement ne peut pas dépasser 32767/*kr* secondes.

Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

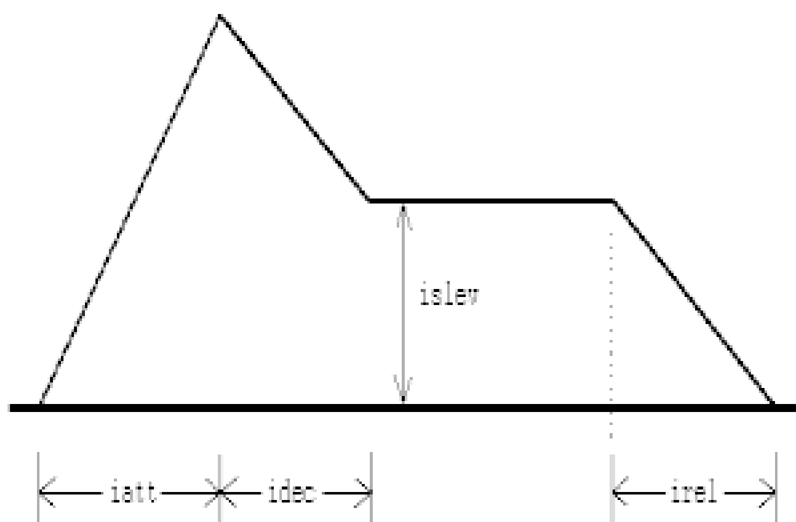


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des évènements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *madsr*, car la durée est allongée automatiquement.

Exemples

Voici un exemple de l'opcode madsr. Il utilise le fichier *madsr.csd* [examples/madsr.csd].

Exemple 284. Exemple de l'opcode madsr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o madsr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Iain McCurdy */
; Initialize the global variables.
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

; Instrument #1.
instr 1
  ; Attack time.
  iattack = 0.5
  ; Decay time.
  idecay = 0
  ; Sustain level.
```

```
  isustain = 1
  ; Release time.
  irelease = 0.5
  aenv madsr iattack, idecay, isustain, irelease

  a1 oscili 10000, 440, 1
  out a1*aenv
endin


</CsInstruments>
<CsScore>

/* Written by Iain McCurdy */
; Table #1, a sine wave.
f 1 0 1024 10 1

; Leave the score running for 6 seconds.
f 0 6

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

linsegr, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *xadsr expon*, *expseg*, *expsega line*, *linseg*, *xtratim*

Crédits

Auteur : John ffitch

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Décembre 2002. Merci à Iain McCurdy pour avoir ajouté un exemple.

Décembre 2002. Merci à Istvan Varga pour avoir indiqué la durée maximale de relâchement.

Nouveau dans la version 3.49 de Csound.

mandel

mandel — Ensemble de Mandelbrot.

Description

Retourne le nombre d'itérations correspondant à un point donné du plan complexe auquel on applique les formules de l'ensemble de Mandelbrot.

Syntaxe

```
kiter, koutrig mandel  ktrig, kx, ky, kmaxIter
```

Exécution

kiter - nombre d'itérations

koutrig - signal de déclenchement en sortie

ktrig - signal de déclenchement en entrée

kx, ky - coordonnées d'un point appartenant au plan complexe

kmaxIter - nombre maximum d'itérations autorisé

mandel est un opcode qui utilise les formules de l'ensemble de Mandelbrot pour générer une sortie que l'on peut appliquer à n'importe quel paramètre musical (ou non musical). Il comprend deux paramètres de sortie : *kiter*, qui contient le nombre d'itérations d'un point donné, et *koutrig*, qui génère un 'bang' de déclenchement chaque fois que *kiter* change. L'évaluation d'un nouveau nombre d'itérations n'a lieu que lorque *ktrig* prend une valeur non nulle. Les coordonnées du plan complexe sont fixées dans *kx* et *ky*, tandis que *kmaxIter* contient le nombre maximum d'itérations. Les valeurs de sorties, qui sont des nombres entiers, peuvent être interprétées de n'importe quelle manière par le compositeur.

Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Seulement disponible auparavant dans CsoundAV)

mandol

mandol — Une simulation de mandoline.

Description

Une simulation de mandoline.

Syntaxe

```
ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [, iminfreq]
```

Initialisation

ifn -- numéro d'une table contenant la forme d'onde du pincement de corde. Le fichier *mandpluk.aiff* [examples/mandpluk.aiff] convient pour cela. On peut aussi l'obtenir à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

iminfreq (facultatif, 0 par défaut) -- Fréquence la plus basse pour une note. Si ce paramètre est omis, il prend la valeur initiale de *kfreq*.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note.

kpluck -- Position du pincement sur la corde, compris entre 0 et 1. Valeur suggérée : 0,4.

kdetune -- Proportion de désaccord entre les deux cordes. La valeur suggérée va de 0,9 à 1.

kgain -- le gain de la boucle du modèle, compris entre 0,97 et 1.

ksize -- La taille du corps de la mandoline. Compris entre 0 et 2.

Exemples

Voici un exemple de l'opcode mandol. Il utilise les fichiers *mandol.csd* [examples/mandol.csd], et *mandpluk.aiff* [examples/mandpluk.aiff].

Exemple 285. Exemple de l'opcode mandol.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mandol.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; kamp = 30000
  ; kfreq = 880
  ; kpluck = 0.4
  ; kdetune = 0.99
  ; kgain = 0.99
  ; ksize = 2
  ; ifn = 1
  ; ifreq = 220

  a1 mandol 30000, 880, 0.4, 0.99, 0.99, 2, 1, 220

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: the "mandpluk.aiff" audio file
f 1 0 8192 1 "mandpluk.aiff" 0 0 0

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

marimba

marimba — Modèle physique de la frappe d'un bloc de bois.

Description

La sortie audio est un son tel que celui produit lorque l'on frappe un bloc de bois comme dans un marimba. Il s'agit d'un modèle physique développé d'après Perry Cook mais recodé pour Csound.

Syntaxe

```
ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \
      [, idoubles] [, itriples]
```

Initialisation

ihrd -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

ipos -- le point d'impact sur le bloc, compris entre 0 et 1.

imp -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

ivfn -- forme du vibrato, habituellement une table sinus, créée par une fonction

idec -- durée avant la fin de la note lorqu'il y a une atténuation

idoubles (facultatif) -- pourcentage de frappes doubles. La valeur par défaut est de 40%.

itriples (facultatif) -- pourcentage de frappes triples. La valeur par défaut est de 20%.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note.

kvibf -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de de 0 à 12.

kvamp -- Amplitude du vibrato.

Exemples

Voici un exemple de l'opcode marimba. Il utilise les fichiers *marimba.csd* [examples/marimba.csd] et *marmstk1.wav* [examples/marmstk1.wav].

Exemple 286. Exemple de l'opcode marimba.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o marimba.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 2

; Instrument #1.
instr 1
  ifreq = cpspch(p4)
  ihrd = 0.1
  ipos = 0.561
  imp = 1
  kvibf = 6.0
  kvamp = 0.05
  ivibfn = 2
  idec = 0.6

  a1 marimba 20000, ifreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec, 20, 10

  outs a1, a1
endin


</CsInstruments>
<CsScore>

; Table #1, the "marmstk1.wav" audio file.
f 1 0 256 1 "marmstk1.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1 8.09
i 1 + 0.5 8.00
i 1 + 0.5 7.00
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.3334 8.01
i 1 + 0.25 8.00
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.25 8.01
i 1 + 0.3333 7.00
i 1 + 0.3334 6.00

e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

vibes

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 3.47 de Csound

massign

massign — Affecte un numéro de canal MIDI à un instrument de Csound.

Description

Affecte un numéro de canal MIDI à un instrument de Csound.

Syntaxe

```
massign ichnl, insnum[, ireset]
```

```
massign ichnl, "insname"[, ireset]
```

Initialisation

ichnl -- numéro de canal MIDI (1-16).

insnum -- numéro de l'instrument d'orchestre de Csound. S'il est inférieur ou égal à zéro, le canal est désactivé (c-à-d. qu'il ne déclenche aucun instrument de csound, bien que l'information soit toujours reçue par des opcodes tels que *midiin*).

« insname » -- une chaîne de caractères entre guillemets représentant un nom d'instrument.

ireset -- sil est non nul, les contrôleurs sont réinitialisés ; c'est le comportement par défaut.

Exécution

Affecte un numéro de canal MIDI à un instrument de Csound. Egalement utile pour s'assurer qu'un instrument particulier (si son numéro est compris entre 1 et 16) ne sera pas déclenché par des messages MIDI noteon (si l'on utilise quelque chose comme *midiin* pour interpréter l'information MIDI). Dans ce cas, fixer *insnum* à un nombre inférieur ou égal à 0.

Si *ichan* est fixé à 0, la valeur de *insnum* est utilisée pour tous les canaux. On peut envoyer de cette manière tous les canaux MIDI vers un seul instrument de Csound. On peut aussi empêcher le déclenchement des instruments à partir d'évènements de note MIDI en provenance de tous les canaux avec la ligne suivante :

```
massign 0, 0
```

Ceci peut être utile si l'on effectue toutes les évaluations MIDI dans Csound avec un instrument actif en permanence (par exemple en utilisant *midiin* et *turnon*) pour éviter une doublure de l'instrument quand une note est jouée.

Voir Aussi

ctrlinit

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Le paramètre *ireset* est nouveau dans Csound5

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de canal MIDI.

max

max — Produces a signal that is the maximum of any number of input signals.

Description

The *max* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the maximum of all of the inputs. For a-rate signals, the inputs are compared one sample at a time (i.e. *max* does not scan an entire ksmps period of a signal for its local maximum as the *max_k* opcode does).

Syntax

```
amax max ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmax max kin1 [, kin2] [, kin3] [, kin4] [...]
```

Performance

ain1, ain2, ... -- a-rate signals to be compared.

kin1, kin2, ... -- k-rate signals to be compared.

See Also

min, *maxabs*, *minabs*, *maxaccum*, *minaccum*, *maxabsaccum*, *minabsaccum*, *max_k*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

maxabs

maxabs — Produces a signal that is the maximum of the absolute values of any number of input signals.

Description

The *maxabs* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the maximum of all of the inputs. It is identical to the *max* opcode except that it takes the absolute value of each input before comparing them. Therefore, the output is always non-negative. For a-rate signals, the inputs are compared one sample at a time (i.e. *maxabs* does not scan an entire ksmps period of a signal for its local maximum as the *max_k* opcode does).

Syntax

```
amax maxabs ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmax maxabs kin1 [, kin2] [, kin3] [, kin4] [...]
```

Performance

ain1, ain2, ... -- a-rate signals to be compared.

kin1, kin2, ... -- k-rate signals to be compared.

See Also

minabs, *max*, *min*, *maxaccum*, *minaccum*, *maxabsaccum*, *minabsaccum*, *max_k*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

maxabsaccum

maxabsaccum — Accumulates the maximum of the absolute values of audio signals.

Description

maxabsaccum compares two audio-rate variables and stores the maximum of their absolute values into the first.

Syntax

```
maxabsaccum aAccumulator, aInput
```

Performance

aAccumulator -- audio variable to store the maximum value

aInput -- signal that aAccumulator is compared to

The *maxabsaccum* opcode is designed to accumulate the maximum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *maxabs* opcode. *maxabsaccum* is identical to *maxaccum* except that it takes the absolute value of aInput before the comparison. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *maxabsaccum* keeps the maximum absolute value instead of adding the signals together. *maxabsaccum* performs the following operation on each pair of samples:

```
if  (abs(aInput) > aAccumulator)  aAccumulator = abs(aInput)
```

aAccumulator will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to zero (perhaps by using the *clear* opcode). Clearing to zero is sufficient for *maxabsaccum*, unlike the *maxaccum* opcode.

See Also

minabsaccum, *maxaccum*, *minaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*, *clear*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

maxaccum

maxaccum — Accumulates the maximum value of audio signals.

Description

maxaccum compares two audio-rate variables and stores the maximum value between them into the first.

Syntax

```
maxaccum aAccumulator, aInput
```

Performance

aAccumulator -- audio variable to store the maximum value

aInput -- signal that aAccumulator is compared to

The *maxaccum* opcode is designed to accumulate the maximum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *max* opcode. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *maxaccum* keeps the maximum value instead of adding the signals together. *maxaccum* performs the following operation on each pair of samples:

```
if  (aInput > aAccumulator)  aAccumulator = aInput
```

aAccumulator will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to zero (perhaps by using the *clear* opcode). Care must be taken however if aInput is negative at any point, in which case the accumulator should be initialized and reset to some large enough negative value that will always be less than the input signals to which it is compared.

See Also

minaccum, *maxabsaccum*, *minabsaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*, *clear*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

maxalloc

maxalloc — Limits the number of allocations of an instrument.

Description

Limits the number of allocations of an instrument.

Syntax

```
maxalloc insnum, icount
```

Initialization

insnum -- instrument number

icount -- number of instrument allocations

Performance

All instances of *maxalloc* must be defined in the header section, not in the instrument body.

Examples

Here is an example of the maxalloc opcode. It uses the file *maxalloc.csd* [examples/maxalloc.csd].

Exemple 287. Example of the maxalloc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o maxalloc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Limit Instrument #1 to three instances.
maxalloc 1, 3

; Instrument #1
instr 1
  ; Generate a waveform, get the cycles per second from the 4th p-field.
  a1 oscil 6500, p4, 1
  out a1
endin
```

```
</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play five instances of Instrument #1 for one second.
; Note that 4th p-field contains cycles per second.
i 1 0 1 220
i 1 0 1 440
i 1 0 1 880
i 1 0 1 1320
i 1 0 1 1760
e


</CsScore>
</CsoundSynthesizer>
```

Its output should contain a message like this:

```
WARNING: cannot allocate last note because it exceeds instr maxalloc
```

See Also

cpuprc, *prealloc*

Credits

Author: Gabriel Maldonado
Italy
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

max_k

max_k — Local maximum (or minimum) value of an incoming asig signal

Description

max_k outputs the local maximum (or minimum) value of the incoming asig signal, checked in the time interval between ktrig has become true twice.

Syntax

```
knumkout max_k asig, ktrig, itype
```

Initialization

itype - itype determinates the behaviour of max_k (see below)

Performance

asig - incoming (input) signal

ktrig - trigger signal

max_k outputs the local maximum (or minimum) value of the incoming *asig* signal, checked in the time interval between *ktrig* has become true twice. *itype* determinates the behaviour of max_k:

1 - absolute maximum (sign of negative values is changed to positive before evaluation)

2 - actual maximum

3 - actual minimum

4 - calculate average value of *asig* in the time interval

This opcode can be useful in several situations, for example to implement a vu-meter.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

mclock

mclock — Sends a MIDI CLOCK message.

Description

Sends a MIDI CLOCK message.

Syntax

```
mclock ifreq
```

Initialization

ifreq -- clock message frequency rate in Hz

Performance

Sends a MIDI CLOCK message (0xF8) every 1/*ifreq* seconds. So *ifreq* is the frequency rate of CLOCK message in Hz.

See Also

mrtmsg

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

mdelay

mdelay — A MIDI delay opcode.

Description

A MIDI delay opcode.

Syntax

```
mdelay kstatus, kchan, kd1, kd2, kdelay
```

Performance

kstatus -- status byte of MIDI message to be delayed

kchan -- MIDI channel (1-16)

kd1 -- first MIDI data byte

kd2 -- second MIDI data byte

kdelay -- delay time in seconds

Each time that *kstatus* is other than zero, *mdelay* outputs a MIDI message to the MIDI out port after *kdelay* seconds. This opcode is useful in implementing MIDI delays. Several instances of *mdelay* can be present in the same instrument with different argument values, so complex and colorful MIDI echoes can be implemented. Further, the delay time can be changed at k-rate.

Examples

Here is an example of the mdelay opcode. It uses the file *mdelay.csd* [examples/mdelay.csd].

Exemple 288. Example of the mdelay opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1  ;Triggered by MIDI notes on channel 1

  kstatus init 0
```

```
  ifund   notnum
  ivel         veloc

  noteondur  1, ifund, ivel, 1

  kstatus = kstatus + 1

  idel1 = .2
  idel2 = .4
  idel3 = .6
  idel4 = .8

  ;make four delay lines

  mdelay         kstatus,1,ifund+2, ivel,idel1
  mdelay         kstatus,1,ifund+4, ivel,idel2
  mdelay         kstatus,1,ifund+6, ivel,idel3
  mdelay         kstatus,1,ifund+8, ivel,idel4

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy
November 1998

New in Csound version 3.492

metro

metro — Trigger Metronome

Description

Generate a metronomic signal to be used in any circumstance an isochronous trigger is needed.

Syntax

```
ktrig  metro  kfreq [, initphase]
```

Initialization

initphase - initial phase value (in the 0 to 1 range)

Performance

ktrig - output trigger signal

kfreq - frequency of trigger bangs in cps

metro is a simple opcode that outputs a sequence of isochronous bangs (that is 1 values) each 1/kfreq seconds. Trigger signals can be used in any circumstance, mainly to temporize realtime algorithmic compositional structures.

Examples

Here is an example of the metro opcode. It uses the file *metro.csd* [examples/metro.csd]

Exemple 289. Example of the metro opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =        44100
kr      =        100
ksmps   =        441
nchnls  =        2

       instr     1
ktrig metro 0.2
printk2 ktrig
        endin

</CsInstruments>
<CsScore>
i 1 0 20


</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example written by Andrés Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

midic14

midic14 — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest midic14 ictlno1, ictlno2, imin, imax [, ifn]
```

```
kdest midic14 ictlno1, ictlno2, kmin, kmax [, ifn]
```

Initialisation

idest -- signal de sortie

ictln1o -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids faible (0-127)

imin -- valeur décimale minimale de sortie définie par l'utilisateur

imax -- valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

Exécution

kdest -- signal de sortie

kmin -- valeur décimale minimale de sortie définie par l'utilisateur

kmax -- valeur décimale maximale de sortie définie par l'utilisateur

midic14 (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite deux contrôleurs MIDI en entrée.

Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des évènements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un évènement de partition.

Voir aussi

ctrl7, *ctrl14*, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic7*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

midic21

midic21 — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest midic21 ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
```

```
kdest midic21 ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

Initialisation

idest -- signal de sortie

ictln1o -- numéro de contrôleur pour l'octet de poids fort (0-127)

ictlno2 -- numéro de contrôleur pour l'octet de poids moyen (0-127)

ictlno3 -- numéro de contrôleur pour l'octet de poids faible (0-127)

imin -- valeur décimale minimale de sortie définie par l'utilisateur

imax -- valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

Exécution

kdest -- signal de sortie

kmin -- valeur décimale minimale de sortie définie par l'utilisateur

kmax -- valeur décimale maximale de sortie définie par l'utilisateur

midic21 (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite trois contrôleurs MIDI en entrée.

Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des évènements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un évènement de partition.

Voir aussi

ctrl7, *ctrl14*, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic7*, *midic14*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

midic7

midic7 — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

Syntaxe

```
idest midic7 ictlno, imin, imax [, ifn]
```

```
kdest midic7 ictlno, kmin, kmax [, ifn]
```

Initialisation

idest -- signal de sortie

ictlno -- numéro de contrôleur MIDI (0-127)

imin -- valeur décimale minimale de sortie définie par l'utilisateur

imax -- valeur décimale maximale de sortie définie par l'utilisateur

ifn (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imin* et *imax*.

Exécution

kdest -- signal de sortie

kmin -- valeur décimale minimale de sortie définie par l'utilisateur

kmax -- valeur décimale maximale de sortie définie par l'utilisateur

midic7 (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Dans *midic7* les valeurs minimale et maximale peuvent varier au taux-k.

Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des évènements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un évènement de partition.

Voir aussi

ctrl7, *ctrl14*, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic14*, *midic21*

Crédits

Auteur : Gabriel Maldonado
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

midichannelaftertouch

midichannelaftertouch — Gets a MIDI channel's aftertouch value.

Description

midichannelaftertouch is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midichannelaftertouch xchannelaftertouch [, ilow] [, ihigh]
```

Initialization

ilow (optional) -- optional low value after rescaling, defaults to 0.

ihigh (optional) -- optional high value after rescaling, defaults to 127.

Performance

xchannelaftertouch -- returns the MIDI channel aftertouch during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xchannelaftertouch* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xchannelaftertouch* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midichannelaftertouch opcode. It uses the file *midichannelaftertouch.csd* [examples/midichannelaftertouch.csd].

Exemple 290. Example of the midichannelaftertouch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac          -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichannelaftertouch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kaft init 0
  midichannelaftertouch kaft

  ; Display the aftertouch value when it changes.
  printk2 kaft
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1   127.00000
 i1    20.00000
 i1    44.00000
```

See Also

midicontrolchange, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midichn

midichn — Returns the MIDI channel number from which the note was activated.

Description

midichn returns the MIDI channel number (1 - 16) from which the note was activated. In the case of score notes, it returns 0.

Syntax

```
ichn midichn
```

Initialization

ichn -- channel number. If the current note was activated from score, it is set to zero.

Examples

Here is a simple example of the midichn opcode. It uses the file *midichn.csd* [examples/midichn.csd].

Exemple 291. Example of the midichn opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichn.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 midichn

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

Here is an advanced example of the midichn opcode. It uses the file *midichn_advanced.csd* [examples/midichn_advanced.csd].

Don't forget that you must include the *-F flag* when using an external MIDI file like « midichn_advanced.mid ».

Exemple 292. An advanced example of the midichn opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichn_advanced.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr =  44100
ksmps =  10
nchnls =  1

        massign  1, 1         ; all channels use instr 1
        massign  2, 1
        massign  3, 1
        massign  4, 1
        massign  5, 1
        massign  6, 1
        massign  7, 1
        massign  8, 1
        massign  9, 1
        massign 10, 1
        massign 11, 1
        massign 12, 1
        massign 13, 1
        massign 14, 1
        massign 15, 1
        massign 16, 1

gicnt =  0                  ; note counter

        instr 1

gicnt =  gicnt + 1 ; update note counter
kcnt init gicnt ; copy to local variable
ichn midichn          ; get channel number
istime times          ; note-on time

        if (ichn > 0.5) goto l2         ; MIDI note
        printks "note %.0f (time = %.2f) was activated from the score\\n", \
                3600, kcnt, istime
        goto l1
l2:
        printks "note %.0f (time = %.2f) was activated from channel %.0f\\n", \
                3600, kcnt, istime, ichn
l1:
        endin


</CsInstruments>
<CsScore>

t 0 60
f 0 6 2 -2 0
i 1 1 0.5
i 1 4 0.5
e


</CsScore>
```

```
</CsoundSynthesizer>
```

Its output should include lines like:

```
note 7 (time = 0.00) was activated from channel 4
note 8 (time = 0.00) was activated from channel 2
```

See Also

pgmassign

Credits

Author: Istvan Varga
May 2002

The simple example was written by Kevin Conder.

New in version 4.20

midicontrolchange

midicontrolchange — Gets a MIDI control change value.

Description

midicontrolchange is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midicontrolchange xcontroller, xcontrollervalue [, ilow] [, ihigh]
```

Initialization

ilow (optional) -- optional low value after rescaling, defaults to 0.

ihigh (optional) -- optional high value after rescaling, defaults to 127.

Performance

xcontroller -- specifies a MIDI controller number (0-127).

xcontrollervalue -- returns the value of the MIDI controller during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of the *xcontroller* and *xcontrollervalue* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xcontroller* and *xcontrollervalue* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

See Also

midichannelaftertouch, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

New in version 4.20

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

midictrl

midictrl — Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

Description

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

Syntaxe

```
ival midictrl inum [, imin] [, imax]
```

```
kval midictrl inum [, imin] [, imax]
```

Initialisation

inum -- numéro de contrôleur MIDI (0-127)

imin, imax -- Ajuste les limites minimale et maximales pour les valeurs obtenues.

Exécution

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

Avertissement

midictrl doit être utilisé seulement pour les notes déclenchées par MIDI, permettant la disponibilité d'un numéro de canal associé. Pour les notes activée depuis la partition, les évènements de ligne, ou l'orchestre, il faut utiliser l'opcode *ctrl7* qui prend un numéro de canal explicite.

Voir aussi

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

mididefault

mididefault — Changes values, depending on MIDI activation.

Description

mididefault is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
mididefault xdefault, xvalue
```

Performance

xdefault -- specifies a default value that will be used during MIDI activation.

xvalue -- overwritten by *xdefault* during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode will overwrite the value of *xvalue* with the value of *xdefault*. If the instrument was *NOT* activated by MIDI input, *xvalue* will remain unchanged.

This enables score pfields to receive a default value during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

See Also

midichannelaftertouch, *midicontrolchange*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

New in version 4.20

midiin

midiin — Returns a generic MIDI message received by the MIDI IN port.

Description

Returns a generic MIDI message received by the MIDI IN port

Syntax

```
kstatus, kchan, kdata1, kdata2 midiin
```

Performance

kstatus -- the type of MIDI message. Can be:

- 128 (note off)
- 144 (note on)
- 160 (polyphonic aftertouch)
- 176 (control change)
- 192 (program change)
- 208 (channel aftertouch)
- 224 (pitch bend
- 0 if no MIDI message are pending in the MIDI IN buffer

kchan -- MIDI channel (1-16)

kdata1, kdata2 -- message-dependent data values

midiin has no input arguments, because it reads at the MIDI in port implicitly. It works at k-rate. Normally (i.e., when no messages are pending) *kstatus* is zero, only when MIDI data are present in the MIDI IN buffer, is *kstatus* set to the type of the relevant messages.

Note

Be careful when using *midiin* in low numbered instruments, since a MIDI note will launch additional instances of the instrument, resulting in duplicate events and weird behaviour. Use *massign* to direct MIDI note on messages to a different instrument or to disable triggering of instruments from MIDI.

Examples

Here is an example of the midiin opcode. It uses the file *midiin.csd* [examples/midiin.csd].

Exemple 293. Example of the midiin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -+rtmidi=virtual ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midiin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr        =  44100
ksmps     =  10
nchnls    =  1

; Example by schwaahed 2006

      massign        0, 130  ; make sure that all channels
      pgmassign      0, 130  ; and programs are assigned to test instr


  instr   130

knotelength    init    0
knoteontime    init    0

kstatus, kchan, kdata1, kdata2                  midiin

if (kstatus == 128) then
knoteofftime    times
knotelength    =    knoteofftime - knoteontime
printks "kstatus= %d, kchan = %d, \\tnote#  = %d, velocity = %d \\tNote OFF\\t%f %f\\n", 0, kstatus, kc

elseif (kstatus == 144) then
knoteontime    times
printks "kstatus= %d, kchan = %d, \\tnote#  = %d, velocity = %d \\tNote ON\\t%f\\n", 0, kstatus, kchan,


elseif (kstatus == 160) then
printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tPolyphonic Aftertouch\\n", 0, kstatus,

elseif (kstatus == 176) then
printks "kstatus= %d, kchan = %d, \\t CC = %d, value = %d \\tControl Change\\n", 0, kstatus, kchan, kda

elseif (kstatus == 192) then
printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tProgram Change\\n", 0, kstatus, kchan,

elseif (kstatus == 208) then
printks  "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tChannel Aftertouch\\n", 0, kstatus, k

elseif (kstatus == 224) then
printks "kstatus= %d, kchan = %d, \\t ( data1 , kdata2 ) = ( %d, %d )\\tPitch Bend\\n", 0, kstatus, kch

endif

  endin

</CsInstruments>
<CsScore>
i130 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy
1998

New in Csound version 3.492

midinoteoff

midinoteoff — Gets a MIDI noteoff value.

Description

midinoteoff is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midinoteoff xkey, xvelocity
```

Performance

xkey -- returns MIDI key during MIDI activation, remains unchanged otherwise.

xvelocity -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of the *xkey* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xkey* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midinoteoff opcode. It uses the file *midinoteoff.csd* [examples/midinoteoff.csd].

Exemple 294. Example of the midinoteoff opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteoff.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kkey init 0
  kvelocity init 0

  midinoteoff kkey, kvelocity

  ; Display the key value when it changes.
  printk2 kkey
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1    60.00000
 i1    76.00000
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midinoteoncps

midinoteoncps — Gets a MIDI note number as a cycles-per-second frequency.

Description

midinoteoncps is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midinoteoncps xcps, xvelocity
```

Performance

xcps -- returns MIDI key translated to cycles per second during MIDI activation, remains unchanged otherwise.

xvelocity -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xcps* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xcps* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midinoteoncps opcode. It uses the file *midinoteoncps.csd* [examples/midinoteoncps.csd].

Exemple 295. Example of the midinoteoncps opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteoncps.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kcps init 0
  kvelocity init 0

  midinoteoncps kcps, kvelocity

  ; Display the cycles-per-second value when it changes.
  printk2 kcps
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1   261.62561
 i1   440.00006
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midinoteonkey

midinoteonkey — Gets a MIDI note number value.

Description

midinoteonkey is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midinoteonkey xkey, xvelocity
```

Performance

xkey -- returns MIDI key during MIDI activation, remains unchanged otherwise.

xvelocity -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xkey* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xkey* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midinoteonkey opcode. It uses the file *midinoteonkey.csd* [examples/midinoteonkey.csd].

Exemple 296. Example of the midinoteonkey opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteonkey.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kkey init 0
  kvelocity init 0

  midinoteonkey kkey, kvelocity

  ; Display the key value when it changes.
  printk2 kkey
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1    60.00000
 i1    69.00000
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midinoteonoct

midinoteonoct — Gets a MIDI note number value as octave-point-decimal value.

Description

midinoteonoct is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midinoteonoct xoct, xvelocity
```

Performance

xoct -- returns MIDI key translated to linear octaves during MIDI activation, remains unchanged otherwise.

xvelocity -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xoct* and *xvelocity* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xoct* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midinoteonoct opcode. It uses the file *midinoteonoct.csd* [examples/midinoteonoct.csd].

Exemple 297. Example of the midinoteonoct opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteonoct.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  koct init 0
  kvelocity init 0

  midinoteonoct koct, kvelocity

  ; Display the octave-point-decimal value when it changes.
  printk2 koct
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1     8.00000
 i1     9.33333
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midinoteonpch

midinoteonpch — Gets a MIDI note number as a pitch-class value.

Description

midinoteonpch is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midinoteonpch xpch, xvelocity
```

Performance

xpch -- returns MIDI key translated to octave.pch during MIDI activation, remains unchanged otherwise.

xvelocity -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xpch* and *xvelocity* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xpch* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midinoteonpch opcode. It uses the file *midinoteonpch.csd* [examples/midinoteonpch.csd].

Exemple 298. Example of the midinoteonpch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteonpch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kpch init 0
  kvelocity init 0

  midinoteonpch kpch, kvelocity

  ; Display the pitch-class value when it changes.
  printk2 kpch
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1     8.09000
 i1     9.05000
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midion

midion — Generates MIDI note messages at k-rate.

Description

Generates MIDI note messages at k-rate.

Syntax

```
midion kchn, knum, kvel
```

Performance

kchn -- MIDI channel number (1-16)

knum -- note number (0-127)

kvel -- velocity (0-127)

midion (k-rate note on) plays MIDI notes with current *kchn*, *knum* and *kvel*. These arguments can be varied at k-rate. Each time the MIDI converted value of any of these arguments changes, last MIDI note played by current instance of *midion* is immediately turned off and a new note with the new argument values is activated. This opcode, as well as *moscil*, can generate very complex melodic textures if controlled by complex k-rate signals.

Any number of *midion* opcodes can appear in the same Csound instrument, allowing a counterpoint-style polyphony within a single instrument.

Examples

Here is a simple example of the midion opcode. It uses the file *midion_simple.csd* [examples/midion_simple.csd].

Exemple 299. Simple Example of the midion opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a minor chord over every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 2

; Example by Giorgio Zucco 2007


instr 1 ;Triggered by MIDI notes on channel 1

  ifund notnum
  ivel  veloc

  knote1 init ifund
  knote2 init ifund + 3
  knote3 init ifund + 5

  ;minor chord on MIDI out channel 1
  ;Needs something plugged to csound's MIDI output
  midion 1, knote1,ivel
  midion 1, knote2,ivel
  midion 1, knote3,ivel

endin


</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the midion opcode. It uses the file *midion_scale.csd* [examples/midion_scale.csd].

Exemple 300. Example of the midion opcode to generate random notes from a scale.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates random notes from a given scale for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ; Triggered by MIDI notes on channel 1

  ivel         veloc

  krate = 8
  iscale = 100 ;f

  ; Random sequence from table f100
  krnd  randh int(14),krate,-1
  knote table abs(krnd),iscale
  ; Generates random notes from the scale on ftable 100
  ; on channel 1 of csound's MIDI output
  midion 1,knote,ivel
```

```
endin

</CsInstruments>
<CsScore>
f100 0 32 -2  40 50 60 70 80 44 54 65 74 84 39 49 69 69

; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>
```

See Also

moscil, *midion2*, *noteon*, *noteoff*, *noteondur*, *noteondur2*

Credits

Author: Gabriel Maldonado
Italy
May 1997

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

midion2

midion2 — Sends noteon and noteoff messages to the MIDI OUT port.

Description

Sends noteon and noteoff messages to the MIDI OUT port when triggered by a value different than zero.

Syntax

```
midion2 kchn, knum, kvel, ktrig
```

Performance

kchn -- MIDI channel (1-16)

knum -- MIDI note number (0-127)

kvel -- note velocity (0-127)

ktrig -- trigger input signal (normally 0)

Similar to *midion*, this opcode sends noteon and noteoff messages to the MIDI out port, but only when *ktrig* is non-zero. This opcode is can work together with the output of the *trigger* opcode.

See Also

moscil, *midion*, *noteon*, *noteoff*, *noteondur*, *noteondur2*

Credits

Author: Gabriel Maldonado
Italy
1998

New in Csound version 3.492

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

midiout

midiout — Sends a generic MIDI message to the MIDI OUT port.

Description

Sends a generic MIDI message to the MIDI OUT port.

Syntax

```
midiout kstatus, kchan, kdata1, kdata2
```

Performance

kstatus -- the type of MIDI message. Can be:

- 128 (note off)
- 144 (note on)
- 160 (polyphonic aftertouch)
- 176 (control change)
- 192 (program change)
- 208 (channel aftertouch)
- 224 (pitch bend)
- 0 when no MIDI messages must be sent to the MIDI OUT port

kchan -- MIDI channel (1-16)

kdata1, kdata2 -- message-dependent data values

midiout has no output arguments, because it sends a message to the MIDI OUT port implicitly. It works at k-rate. It sends a MIDI message only when *kstatus* is non-zero.

Avertissement

Warning: Normally *kstatus* should be set to 0. Only when the user intends to send a MIDI message, can it be set to the corresponding message type number.

Credits

Author: Gabriel Maldonado
Italy
1998

New in Csound version 3.492

midipitchbend

midipitchbend — Gets a MIDI pitchbend value.

Description

midipitchbend is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midipitchbend xpitchbend [, ilow] [, ihigh]
```

Initialization

ilow (optional) -- optional low value after rescaling, defaults to 0.

ihigh (optional) -- optional high value after rescaling, defaults to 127.

Performance

xpitchbend -- returns the MIDI pitch bend during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xpitchbend* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xpitchbend* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

Examples

Here is an example of the midipitchbend opcode. It uses the file *midipitchbend.csd* [examples/midipitchbend.csd].

Exemple 301. Example of the midipitchbend opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midipitchbend.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kpb init 0

  midipitchbend kpb

  ; Display the pitch-bend value when it changes.
  printk2 kpb
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
 i1     0.12695
 i1     0.00000
 i1    -0.01562
```

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipolyaftertouch*, *midiprogramchange*

Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

midipolyaftertouch

midipolyaftertouch — Gets a MIDI polyphonic aftertouch value.

Description

midipolyaftertouch is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midipolyaftertouch xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]
```

Initialization

ilow (optional) -- optional low value after rescaling, defaults to 0.

ihigh (optional) -- optional high value after rescaling, defaults to 127.

Performance

xpolyaftertouch -- returns MIDI polyphonic aftertouch during MIDI activation, remains unchanged otherwise.

xcontrollervalue -- returns the value of the MIDI controller during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xpolyaftertouch* and *xcontrollervalue* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xpolyaftertouch* and *xcontrollervalue* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *mi-*

dinoteonoct, *midinoteonpch*, *midipitchbend*, *midiprogramchange*

Credits

Author: Michael Gogins

New in version 4.20

midiprogramchange

midiprogramchange — Gets a MIDI program change value.

Description

midiprogramchange is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

Syntax

```
midiprogramchange xprogram
```

Performance

xprogram -- returns the MIDI program change value during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xprogram* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xprogram* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.

Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

See Also

midichannelaftertouch, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*

Credits

Author: Michael Gogins

New in version 4.20

miditempo

miditempo — Returns the current tempo at k-rate, of either the MIDI file (if available) or the score

Description

Returns the current tempo at k-rate, of either the MIDI file (if available) or the score

Syntax

```
ksig  miditempo
```

Credits

Author: Istvan Varga
March 2005
New in Csound5

midremot

midremot — An opcode which can be used to implement a remote midi orchestra. This opcode will send midi events from a source machine to one destination.

Description

With the *midremot* and *midglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the midi score. During the performance the master machine sends the midi events to the clients. The *midremot* opcode will send events from a source machine to one destination if you want to send events to many destinations (broadcast) use the *midglobal* opcode instead. These two opcodes can be used in combination.

Syntax

```
midremotidestination, isource, instrnum [,instrnum...]
```

Initialization

idestination -- a string that is the intended host computer (e.g. 192.168.0.100). This is the destination host which receives the events from the given instrument.

isource -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument and sends it to the address given by idestination.

instrnum -- a list of instrument numbers which will be played on the destination machine

Example

Examples

Here is an example of the midremot opcode. It uses the files *insremot.csd* [examples/midremot.csd].

Exemple 302. Example of the insremot opcode.

The example shows a Bach fugue played on 4 remote computers. The master machine is named "192.168.1.100", client1 "192.168.1.101" and so on. Start the clients on each machine (they will be waiting to receive the events from the master machine) and then start the master. Here is the command on linux to start a client (csound -dm0 -odac -+rtaudio=alsa midremot.csd -+rtmidi=Null), and the command on the master machine will look like this (csound -dm0 -odac -+rtaudio=alsa midremot.csd -F midremot.mid).

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o midremot.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 441
ksmps = 100
nchnls = 2

massign 1,1
massign 2,2
massign 3,3
massign 4,4
massign 5,5

ga1 init 0
ga2 init 0

gi1 sfload    "19Trumpet.sf2"

gi2 sfload    "01hpschd.sf2"

gi3 sfload    "07AcousticGuitar.sf2"

gi4 sfload    "22Bassoon.sf2"

gitab ftgen 1,0,1024,10,1

midremot "192.168.1.100", "192.168.1.101", 1
midremot "192.168.1.100", "192.168.1.102", 2
midremot "192.168.1.100", "192.168.1.103", 3

midglobal "192.168.1.100", 5


        instr 1
        sfpassign    0, gi1
ifreq cpsmidi
iamp ampmidi 10
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
        outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
        endin

        instr 2
        sfpassign 0,    gi2
ifreq cpsmidi
iamp ampmidi 15
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
        outs a1,a2
vincr ga1, a1*.4
vincr ga2, a2*.4
        endin

        instr 3
        sfpassign 0,    gi3
ifreq cpsmidi
iamp ampmidi 10
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
        outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
        endin

        instr 4
        sfpassign 0,    gi4
```

```
ifreq cpsmidi
iamp ampmidi 15
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
        outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
        endin

instr 5
      kamp midic7 1,0,1
      denorm ga1
      denorm ga2
aL, aR  reverbsc ga1, ga2, .9, 16000, sr, 0.5
        outs aL, aR
        ga1 = 0
     ga2   =     0
endin

</CsInstruments>
<CsScore>
; Score
f0  160
</CsScore>
</CsoundSynthesizer>
```

See also

insglobal, *insremot*, *midglobal*, *remoteport*

Credits

Author: Simon Schampijer
2006

New in version 5.03

midglobal

midglobal — An opcode which can be used to implement a remote midi orchestra. This opcode will broadcast the midi events to all the machines involved in the remote concert.

Description

With the *midremot* and *midglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the midi score. During the performance the master machine sends the midi events to the clients. The midglobal opcode sends the events to all the machines involved in the remote concert. These machines are determined by the *midremot* definitions made above the *midglobal* command. To send events to only one machine use *midremot*.

Syntax

```
midglobalisource, instrnum [,instrnum...]
```

Initialization

isource -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument(s) and sends it to all the machines involved in the remote concert.

instrnum -- a list of instrument numbers which will be played on the destination machines

Examples

See the entry for *midremot* for an example of usage.

See also

insglobal, *insremot*, *midremot*, *remoteport*

Credits

Author: Simon Schampijer
2006

New in version 5.03

min

min — Produces a signal that is the minimum of any number of input signals.

Description

The *min* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the minimum of all of the inputs. For a-rate signals, the inputs are compared one sample at a time (i.e. *min* does not scan an entire ksmps period of a signal for its local minimum as the *max_k* opcode does).

Syntax

```
amin min ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmin min kin1 [, kin2] [, kin3] [, kin4] [...]
```

Performance

ain1, ain2, ... -- a-rate signals to be compared.

kin1, kin2, ... -- k-rate signals to be compared.

See Also

max, *maxabs*, *minabs*, *maxaccum*, *minaccum*, *maxabsaccum*, *minabsaccum*, *max_k*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

minabs

minabs — Produces a signal that is the minimum of the absolute values of any number of input signals.

Description

The *minabs* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the minimum of all of the inputs. It is identical to the *min* opcode except that it takes the absolute value of each input before comparing them. Therefore, the output is always non-negative. For a-rate signals, the inputs are compared one sample at a time (i.e. *minabs* does not scan an entire ksmps period of a signal for its local minimum as the *max_k* opcode does).

Syntax

```
amin minabs ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmin minabs kin1 [, kin2] [, kin3] [, kin4] [...]
```

Performance

ain1, ain2, ... -- a-rate signals to be compared.

kin1, kin2, ... -- k-rate signals to be compared.

See Also

maxabs, *max*, *min*, *maxaccum*, *minaccum*, *maxabsaccum*, *minabsaccum*, *max_k*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

minabsaccum

minabsaccum — Accumulates the minimum of the absolute values of audio signals.

Description

minabsaccum compares two audio-rate variables and stores the minimum of their absolute values into the first.

Syntax

```
minabsaccum aAccumulator, aInput
```

Performance

aAccumulator -- audio variable to store the minimum value

aInput -- signal that aAccumulator is compared to

The *minabsaccum* opcode is designed to accumulate the minimum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *minabs* opcode. *minabsaccum* is identical to *minaccum* except that it takes the absolute value of aInput before the comparison. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *minabsaccum* keeps the minimum absolute value instead of adding the signals together. *minabsaccum* performs the following operation on each pair of samples:

```
if  (abs(aInput) < aAccumulator)  aAccumulator = abs(aInput)
```

aAccumulator will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to some large enough positive value that will always be greater than the input signals to which it is compared.

See Also

maxabsaccum, *maxaccum*, *minaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

minaccum

minaccum — Accumulates the minimum value of audio signals.

Description

minaccum compares two audio-rate variables and stores the minimum value between them into the first.

Syntax

```
minaccum aAccumulator, aInput
```

Performance

aAccumulator -- audio variable to store the minimum value

aInput -- signal that aAccumulator is compared to

The *minaccum* opcode is designed to accumulate the minimum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *min* opcode. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *minaccum* keeps the minimum value instead of adding the signals together. *minaccum* performs the following operation on each pair of samples:

```
if  (aInput < aAccumulator)  aAccumulator = aInput
```

aAccumulator will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to some large enough positive value that will always be greater than the input signals to which it is compared.

See Also

maxaccum, *maxabsaccum*, *minabsaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*

Credits

Author: Anthony Kozar
March 2006

New in Csound version 5.01

mirror

mirror — Reflects the signal that exceeds the low and high thresholds.

Description

Reflects the signal that exceeds the low and high thresholds.

Syntax

```
ares mirror asig, klow, khigh
```

```
ires mirror isig, ilow, ihigh
```

```
kres mirror ksig, klow, khigh
```

Initialization

isig -- input signal

ilow -- low threshold

ihigh -- high threshold

Performance

xsig -- input signal

klow -- low threshold

khigh -- high threshold

mirror « reflects » the signal that exceeds the low and high thresholds.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals.

See Also

limit, *wrap*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.49

MixerSetLevel

MixerSetLevel — Sets the level of a send to a buss.

Syntax

```
MixerSetLevel isend, ibuss, kgain
```

Description

Sets the level at which signals from the send are added to the buss. The actual sending of the signal to the buss is performed by the *MixerSend* opcode.

Initialization

isend -- The number of the send, for example the number of the instrument sending the signal (but any integer can be used).

ibuss -- The number of the buss, for example the number of the instrument receiving the signal (but any integer can be used).

Setting the gain for a buss also creates the buss.

Performance

kgain -- The level (any real number) at which the signal from the send will be mixed onto the buss. The default is 0.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses before the next kperiod.

Examples

In the orchestra, define an instrument to control mixer levels:

```
instr 1
        MixerSetLevel         p4, p5, p6
endin
```

In the score, use that instrument to set mixer levels:

```
; SoundFonts
; to Chorus
i 1  0  0 100  200  0.9
; to Reverb
i 1  0  0  100  210  0.7
; to Output
i 1  0  0  100  220  0.3

; Kelley Harpsichord
; to Chorus
i 1  0  0  3  200  0.30
; to Reverb
```

```
i 1  0  0  3  210  0.9
; to Output
i 1  0  0  3  220  0.1

; Chorus to Reverb
i 1  0  0  200  210  0.5
; Chorus to Output
i 1  0  0  200  220  0.5
; Reverb to Output
i 1  0  0  210  220  0.2
```

Credits

Michael Gogins (gogins at pipeline dot com).

MixerGetLevel

MixerGetLevel — Gets the level of a send to a buss.

Syntax

```
kgain MixerGetLevel isend, ibuss
```

Description

Gets the level at which signals from the send are being added to the buss. The actual sending of the signal to the buss is performed by the *MixerSend* opcode.

Initialization

isend -- The number of the send, for example the number of the instrument sending the signal.

ibuss -- The number of the buss, for example the number of the instrument receiving the signal.

Performance

kgain -- The level (any real number) at which the signal from the send will be mixed onto the buss.

This opcode reports the level set by *MixerSetLevel* for a send and buss pair.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

Credits

Michael Gogins (gogins at pipeline dot com).

MixerSend

MixerSend — Mixes an arate signal into a channel of a buss.

Syntax

```
MixerSend asignal, isend, ibuss, ichannel
```

Description

Mixes an arate signal into a channel of a buss.

Initialization

isend -- The number of the send, for example the number of the instrument sending the signal. The gain of the send is controlled by the *MixerSetLevel* opcode. The reason that the sends are numbered is to enable different levels for different sends to be set independently of the actual level of the signals.

ibuss -- The number of the buss, for example the number of the instrument receiving the signal.

ichannel -- The number of the channel. Each buss has `nchnls` channels.

Performance

asignal -- The signal that will be mixed into the indicated channel of the buss.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

Examples

```
instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
iamplitude                 =                          ampdb(p5) * 2.0
; AUDIO
aleft, aright                 fluidAllOut         giFluidsynth
asig1                         =                         aleft * iamplitude
asig2                         =                         aright * iamplitude
                        ; To the chorus.
                        MixerSend         asig1, 100, 200, 0
                        MixerSend         asig2, 100, 200, 1
                        ; To the reverb.
                        MixerSend         asig1, 100, 210, 0
                        MixerSend         asig2, 100, 210, 1
                        ; To the output.
                        MixerSend         asig1, 100, 220, 0
                        MixerSend         asig2, 100, 220, 1
endin
```

Credits

Michael Gogins (gogins at pipeline dot com).

MixerReceive

MixerReceive — Receives an arate signal from a channel of a buss.

Syntax

```
asignal MixerReceive ibuss, ichannel
```

Description

Receives an arate signal that has been mixed onto a channel of a buss.

Initialization

ibuss -- The number of the buss, for example the number of the instrument receiving the signal.

ichannel -- The number of the channel. Each buss has `nchnls` channels.

Performance

asignal -- The signal that has been mixed onto the indicated channel of the buss.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

Examples

```
instr 220 ; Master output
    ; It applies a bass enhancement, compression and fadeout
    ; to the whole piece, outputs signals, and clears the mixer.
  a1  MixerReceive 220, 0
  a2  MixerReceive 220, 1
  ; Bass enhancement
  al1 butterlp a1, 100
  al2 butterlp a2, 100
  a1 = al1*1.5 +a1
  a2 = al2*1.5 +a2

  ; Global amplitude shape
  kenv   linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
  a1=a1*kenv
  a2=a2*kenv

  ; Compression
  a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
  a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

  ; Remove DC bias
  a1blocked         dcblock         a1
  a2blocked dcblock         a2

  ; Output signals
  outs a1blocked, a2blocked
  MixerClear
endin
```

Credits

Michael Gogins (gogins at pipeline dot com).

MixerClear

MixerClear — Resets all channels of a buss to 0.

Syntax

```
MixerClear
```

Description

Resets all channels of a buss to 0.

Performance

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

Examples

```
instr 220 ; Master output
    ; It applies a bass enhancement, compression and fadeout
    ; to the whole piece, outputs signals, and clears the mixer.
  a1  MixerReceive 220, 0
  a2  MixerReceive 220, 1
  ; Bass enhancement
  al1 butterlp a1, 100
  al2 butterlp a2, 100
  a1 = al1*1.5 +a1
  a2 = al2*1.5 +a2

  ; Global amplitude shape
  kenv   linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
  a1=a1*kenv
  a2=a2*kenv

  ; Compression
  a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
  a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

  ; Remove DC bias
  a1blocked         dcblock         a1
  a2blocked dcblock         a2

  ; Output signals
  outs a1blocked, a2blocked
  MixerClear
endin
```

Credits

Michael Gogins (gogins at pipeline dot com).

mode

mode — A filter that simulates a mass-spring-damper system

Description

Filters the incoming signal with the specified resonance frequency and quality factor. It can also be seen as a signal generator for high quality factor, with an impulse for the excitation. You can combine several modes to built complex instruments such as bells or guitar tables.

Syntax

```
aout mode ain, kfreq, kQ [, iskip]
```

Initialization

iskip (optional, default=0) -- if non zero skip the initialisation of the filter.

Performance

aout -- filtered signal

ain -- signal to filter

kfreq -- resonant frequency of the filter

Warning

This filter becomes unstable if sr/ifreq < pi (e.g ifreq > 14037 Hz @44kHz)

kQ -- quality factor of the filter

The resonance time is roughly proportionnal to kQ/kfreq.

See *Modal Frequency Ratios* for frequency ratios of real intruments which can be used to determine the values of *kfreq*.

Examples

Here is an example of the mode opcode. It uses the file *mode.csd* [examples/mode.csd].

Exemple 303. Example of the mode opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
```

```
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1; 2 modes excitator

idur init p3
ifreq11 init p4
ifreq12 init p5
iQ11    init p6
iQ12    init p7
iamp    init ampdb(p8)
ifreq21 init p9
ifreq22 init p10
iQ21    init p11
iQ22    init p12

; to simulate the shock between the excitator and the resonator
ashock  mpulse  3,0

aexc1  mode ashock,ifreq11,iQ11
aexc1 = aexc1*iamp
aexc2  mode ashock,ifreq12,iQ12
aexc2 = aexc2*iamp


aexc = (aexc1+aexc2)/2

;"Contact" condition : when aexc reaches 0, the excitator looses
;contact with the resonator, and stops "pushing it"
aexc limit aexc,0,3*iamp

; 2modes resonator

ares1  mode aexc,ifreq21,iQ21
ares2  mode aexc,ifreq22,iQ22

ares = (ares1+ares2)/2

display aexc+ares,p3
outs  aexc+ares,aexc+ares

endin


</CsInstruments>
<CsScore>

;wooden excitator against glass resonator
i1 0 8  1000   3000  12  8  70  440   888   500  420

;felt against glass
i1 4 8  80   188  8  3  70  440   888   500  420

;wood against wood
i1 8 8  1000   3000  12  8  70  440  630   60  53

;felt against wood
i1 12 8  80   180  8  3  70  440  630   60  53


i1 16 8  1000   3000  12  8  70  440  888   2000  1630
i1 23 8  80   180  8  3  70  440  888   2000  1630


;With a metallic excitator

i1 33 8 1000  1800  1000  720  70   440   882  500  500
i1 37 8 1000  1800  1000  850  70   440   630  60  53

i1 42 8 1000  1800  2000  1720  70   440   442  500  500
```

```
</CsScore>
</CsoundSynthesizer>
```

Credits

Original UDO and documentation/example by François Blanc

Opcode translation to C-code by Steven Yi

New in version 5.04

monitor

monitor — Returns the audio spout frame.

Description

Returns the audio spout frame (if active), otherwise it returns zero.

Syntax

```
aout1 [,aout2 ... aoutX] monitor
```

Performance

This opcode can be used for monitoring the output signal from csound. It should not be used for processing the signal further.

See the entry for the *fout* opcode for an example of usage of *monitor*.

See also

fout, the *Mixer opcodes* and the *Zak Patching System*.

Credits

Istvan Varga 2006

moog

moog — Emulation d'un synthétiseur mini-Moog.

Description

Emulation d'un synthétiseur mini-Moog.

Syntaxe

```
ares moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn
```

Initialisation

iafn, iwfn, ivfn -- les trois numéros des tables contenant la forme d'onde de l'attaque (non bouclée), la forme d'onde de la boucle principale, et la forme d'onde du vibrato. Les fichiers *mandpluk.aiff* [examples/mandpluk.aiff] et *impuls20.aiff* [examples/impuls20.aiff] conviennent bien pour les deux premières et une sinusoïde fera l'affaire pour la troisième.

Note

Les fichiers « mandpluk.aiff » et « impuls20.aiff » sont aussi disponibles à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

Exécution

kamp -- amplitude de la note.

kfreq -- fréquence de la note.

kfiltq -- Q du filtre, compris entre 0,8 et 0,9

kfiltrate -- taux de contrôle pour le filtre, compris entre 0 et 0,0002

kvibf -- fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12

kvamp -- amplitude du vibrato

Exemples

Voici un exemple de l'opcode moog. Il utilise les fichiers *moog.csd* [examples/moog.csd], *mandpluk.aiff* [examples/mandpluk.aiff] et *impuls20.aiff* [examples/impuls20.aiff].

Exemple 304. Exemple de l'opcode moog.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moog.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kfiltq = 0.81
  kfiltrate = 0
  kvibf = 1.4
  kvamp = 2.22
  iafn = 1
  iwfn = 2
  ivfn = 3

  am moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

  ; It tends to get loud, so clip moog's amplitude at 30,000.
  a1 clip am, 2, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: the "mandpluk.aiff" audio file
f 1 0 8192 1 "mandpluk.aiff" 0 0 0
; Table #2: the "impuls20.aiff" audio file
f 2 0 256 1 "impuls20.aiff" 0 0 0
; Table #3: a sine wave
f 3 0 256 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

moogladder

moogladder — Moog ladder lowpass filter.

Description

Moogladder is an new digital implementation of the Moog ladder filter based on the work of Antti Huovilainen, described in the paper "Non-Linear Digital Implementation of the Moog Ladder Filter" (Proceedings of DaFX04, Univ of Napoli). This implementation is probably a more accurate digital representation of the original analogue filter.

Syntax

```
asig moogladder ain, kcf, kres[, istor]
```

Initialization

istor --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal.

kcf -- filter cutoff frequency

kres -- resonance, generally < 1, but not limited to it. Higher than 1 resonance values might cause aliasing, analogue synths generally allow resonances to be above 1.

Examples

Exemple 305. Example

```
kfe         expseg 500, p3*0.9, 1800, p3*0.1, 3000
kenv        linen  10000, 0.05, p3, 0.05
asig        buzz   kenv, 100, sr/(200), 1
afil        moogladder  asig, kfe, 1

            out afil
```

Credits

Author: Victor Lazzarini;
January 2005

New plugin in version 5

January 2005.

moogvcf

moogvcf — A digital emulation of the Moog diode ladder filter configuration.

Description

A digital emulation of the Moog diode ladder filter configuration.

Syntax

```
ares moogvcf asig, xfco, xres [,iscale, iskip]
```

Initialization

iscale (optional, default=1) -- internal scaling factor. Use if *asig* is not in the range +/-1. Input is first divided by *iscale*, then output is mutliplied *iscale*. Default value is 1. (New in Csound version 3.50)

iskip (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

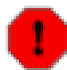
Performance

asig -- input signal

xfco -- filter cut-off frequency in Hz. As of version 3.50, may i-,k-, or a-rate.

xres -- amount of resonance. Self-oscillation occurs when *xres* is approximately one. As of version 3.50, may a-rate, i-rate, or k-rate.

moogvcf is a digital emulation of the Moog diode ladder filter configuration. This emulation is based loosely on the paper « Analyzing the Moog VCF with Considerations for Digital Implemnetation » by Stilson and Smith (CCRMA). This version was originally coded in Csound by Josep Comajuncosas. Some modifications and conversion to C were done by Hans Mikelson



Avertissement

This filter requires that the input signal be normalized to one. This can be easily achieved using *0dbfs*, like this:

ares **moogvcf** asig, kfco, kres, 0dbfs

You can also use *moogvcf2* which defaults scaling to *0dbfs*.

Examples

Here is an example of the moogvcf opcode. It uses the file *moogvcf.csd* [examples/moogvcf.csd].

Exemple 306. Example of the moogvcf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 32000, 220, 1

  ; Vary the filter-cutoff frequency from .2 to 2 KHz.
  kfco line 200, p3, 2000

  ; Set the resonance amount to one.
  krez init 1

  ; Scale the amplitude to 32768.
  iscale = 32768

  a1 moogvcf asig, kfco, krez, iscale

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

moogvcf2, *biquad*, *rezzy*

Credits

Author: Hans Mikelson
October 1998

Example written by Kevin Conder.

New in Csound version 3.49

moogvcf2

moogvcf2 — A digital emulation of the Moog diode ladder filter configuration.

Description

A digital emulation of the Moog diode ladder filter configuration.

Syntax

```
ares moogvcf2 asig, xfco, xres [,iscale, iskip]
```

Initialization

iscale (optional, default=0dBfs) -- internal scaling factor, as the operation of the code requires the signal to be in the range +/-1. Input is first divided by *iscale*, then output is mutliplied by *iscale*.

iskip (optional, default=0) -- if non zero skip the initialisation of the filter.

Performance

asig -- input signal

xfco -- filter cut-off frequency in Hz. which may be i-,k-, or a-rate.

xres -- amount of resonance. Self-oscillation occurs when *xres* is approximately one. May be a-rate, i-rate, or k-rate.

moogvcf2 is a digital emulation of the Moog diode ladder filter configuration. This emulation is based loosely on the paper « Analyzing the Moog VCF with Considerations for Digital Implemnetation » by Stilson and Smith (CCRMA). This version was originally coded in Csound by Josep Comajuncosas. Some modifications and conversion to C were done by Hans Mikelson and then adjusted.

moogvcf2 is identical to *moogvcf*, except that the
iscale parameter defaults to *0dbfs* instead of 0, guaranteeing that amplitude will usually be OK.

Examples

Here is an example of the moogvcf2 opcode. It uses the file *moogvcf2.csd* [examples/moogvcf2.csd].

Exemple 307. Example of the moogvcf2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 32000, 220, 1

  ; Vary the filter-cutoff frequency from .2 to 2 KHz.
  kfco line 200, p3, 2000

  ; Set the resonance amount to one.
  krez init 1

  a1 moogvcf2 asig, kfco, krez

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

moogvcf, *biquad*, *rezzy*

Credits

Author: Hans Mikelson and John ffitch
October 1998/ July 2006

Example written by Kevin Conder.

New in Csound version 5.03

moscil

moscil — Sends a stream of the MIDI notes.

Description

Sends a stream of the MIDI notes.

Syntax

```
moscil kchn, knum, kvel, kdur, kpause
```

Performance

kchn -- MIDI channel number (1-16)

knum -- note number (0-127)

kvel -- velocity (0-127)

kdur -- note duration in seconds

kpause -- pause duration after each noteoff and before new note in seconds

moscil and *midion* are the most powerful MIDI OUT opcodes. *moscil* (MIDI oscil) plays a stream of notes of *kdur* duration. Channel, pitch, velocity, duration and pause can be controlled at k-rate, allowing very complex algorithmically generated melodic lines. When current instrument is deactivated, the note played by current instance of *moscil* is forcedly truncated.

Any number of *moscil* opcodes can appear in the same Csound instrument, allowing a counterpoint-style polyphony within a single instrument.

Examples

Here is an example of the moscil opcode. It uses the file *moscil.csd* [examples/moscil.csd].

Exemple 308. Example of the moscil opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a stream of notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007


instr 1  ;Triggered by MIDI notes on channel 1

  inote notnum
  ivel          veloc

  kpitch = 40
  kfreq  = 2

  kdur   =  .04
  kpause =  .1

  k1         lfo         kpitch, kfreq,5

  ;plays a stream of notes of kdur duration on MIDI channel 1
  moscil  1, inote + k1, ivel,   kdur, kpause

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>
```

See Also

midion, *midion2*, *noteon*, *noteoff*, *noteondur*, *noteondur2*

Credits

Author: Gabriel Maldonado
Italy
May 1997

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

mpulse

mpulse — Génère un ensemble d'impulsions.

Description

Génère un ensemble d'impulsions d'amplitude *kamp* séparées par *kintvl* secondes (ou échantillons si *kintvl* est négatif). La première impulsion est générée après un délai de *ioffset* secondes.

Syntaxe

```
ares mpulse kamp, kintvl [, ioffset]
```

Initialisation

ioffset (facultatif, par défaut 0) -- le délai avant la première impulsion. S'il est négatif, la valeur est interprétée comme le nombre d'échantillons, sinon il représente des secondes. La valeur par défaut est zéro.

Exécution

kamp -- amplitude des impulsions générées

kintvl -- intervalle de temps en secondes (ou en nombre d'échantillons si *kintvl* est négatif) jusqu'à la prochaine impulsion.

Après le délai initial, une impulsion d'amplitude *kamp* est générée comme échantillon unique. Immédiatement après la génération de l'impulsion, la date de la suivante est déterminée par la valeur de *kintvl* à ce moment précis. Cela signifie que tous les changements de *kintvl* entre les impulsions sont ignorés. Si *kintvl* est nul, il y a un temps d'attente infini jusqu'à la prochaine impulsion. Si *kintvl* est négatif, l'intervalle est compté en nombre d'échantillons plutôt qu'en secondes.

Exemples

Voici un exemple de l'opcode mpulse. Il utilise le fichier *mpulse.csd* [examples/mpulse.csd].

Exemple 309. Exemple de l'opcode mpulse.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mpulse.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

gkfreq init 0.1

instr 1
  kamp = 10000

  a1 mpulse kamp, gkfreq
  out a1
endin

instr 2
; Assign the value of p4 to gkfreq
gkfreq init p4
endin
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 10
i 2 2 1    0.05
i 2 4 1    0.01
i 2 6 1    0.005
i 2 8 1    0.001
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Ecrit par John ffitch.

Nouveau dans la version 4.08

Exemple écrit par Kevin Conder.

mrtmsg

mrtmsg — Send system real-time messages to the MIDI OUT port.

Description

Send system real-time messages to the MIDI OUT port.

Syntax

```
mrtmsg imsgtype
```

Initialization

imsgtype -- type of real-time message:

- 1 sends a START message (0xFA);
- 2 sends a CONTINUE message (0xFB);
- 0 sends a STOP message (0xFC);
- -1 sends a SYSTEM RESET message (0xFF);
- -2 sends an ACTIVE SENSING message (0xFE)

Performance

Sends a real-time message once, in init stage of current instrument. *imsgtype* parameter is a flag to indicate the message type.

See Also

mclock

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

multitap

multitap — Multitap delay line implementation.

Description

Multitap delay line implementation.

Syntax

```
ares multitap asig [, itime1] [, igain1] [, itime2] [, igain2] [...]
```

Initialization

The arguments *itime* and *igain* set the position and gain of each tap.

The delay line is fed by *asig*.

Examples

```
a1      oscil      1000, 100, 1
a2      multitap   a1, 1.2, .5, 1.4, .2
        out        a2
```

This results in two delays, one with length of 1.2 and gain of .5, and one with length of 1.4 and gain of .2.

Credits

Author: Paris Smaragdis
MIT, Cambridge
1996

mute

mute — Mutes/unmutes new instances of a given instrument.

Description

Mutes/unmutes new instances of a given instrument.

Syntax

```
mute insnum [, iswitch]
```

```
mute "insname" [, iswitch]
```

Initialization

insnum -- instrument number. Equivalent to *p1* in a score *i statement*.

« *insname* » -- A string (in double-quotes) representing a named instrument.

iswitch (optional, default=0) -- represents a switch to mute/unmute an instrument. A value of 0 will mute new instances of an instrument, other values will unmute them. The default value is 0.

Performance

All new instances of instrument inst will me muted (iswitch = 0) or unmuted (iswitch not equal to 0). There is no difficulty with muting muted instruments or unmuting unmuted instruments. The mechanism is the same as used by the score *q statement*. For example, it is possible to mute in the score and unmute in some instrument.

Muting/Unmuting is indicated by a message (depending on message level).

Examples

Here is an example of the mute opcode. It uses the file *mute.csd* [examples/mute.csd].

Exemple 310. Example of the mute opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mute.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Mute Instrument #2.
mute 2

; Instrument #1.
instr 1
  a1 oscils 10000, 440, 0
  out a1
endin

; Instrument #2.
instr 2
  a1 oscils 10000, 880, 0
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Example written by Kevin Conder.

New in version 4.22

mxadsr

mxadsr — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

Syntaxe

```
ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

Initialisation

iatt -- durée de l'attaque (attack)

idec -- durée de la première chute (decay)

islev -- niveau d'entretien (sustain)

irel -- durée de la chute (release)

idel (facultatif, 0 par défaut) -- délai de niveau zéro avant le démarrage de l'enveloppe

ireltim (facultatif, -1 par défaut) -- Contrôle la durée du relâchement après la réception d'un évènement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

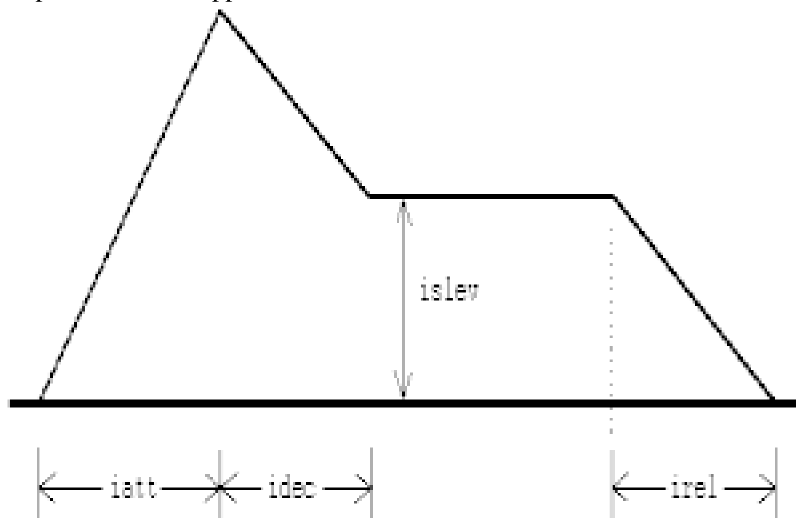


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des évènements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI. L'opcode *mxadsr* est identique à *madsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *mxadsr*, car la durée est allongée automatiquement.

mxadsr est nouveau dans la version 3.51 de Csound.

Voir Aussi

linsegr, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *adsr*, *expon*, *expseg*, *expsega line*, *linseg*, *xtratim*

Crédits

Auteur : John ffitch

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Novembre 2003. Merci à Kanata Motohashi pour avoir fixé le lien vers l'opcode *linsegr*.

nchnls

nchnls — Fixe le nombre de canaux de la sortie audio.

Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

Syntaxe

```
nchnls = iarg
```

Initialisation

nchnls = (facultatif) -- fixe le nombre de canaux de la sortie audio à *iarg*. (1 = mono, 2 = stéréo, 4 = quadriphonique.) La valeur par défaut est 1 (mono).

De plus, toute *variable globale* [54] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Voir Aussi

kr, *ksmps*, *sr*

nestedap

nestedap — Three different nested all-pass filters.

Description

Three different nested all-pass filters, useful for implementing reverbs.

Syntax

```
ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \
      [, idel3] [, igain3] [, istor]
```

Initialization

imode -- operating mode of the filter:

- 1 = simple all-pass filter
- 2 = single nested all-pass filter
- 3 = double nested all-pass filter

idel1, *idel2*, *idel3* -- delay times of the filter stages. Delay times are in seconds and must be greater than zero. *idel1* must be greater than the sum of *idel2* and *idel3*.

igain1, *igain2*, *igain3* -- gain of the filter stages.

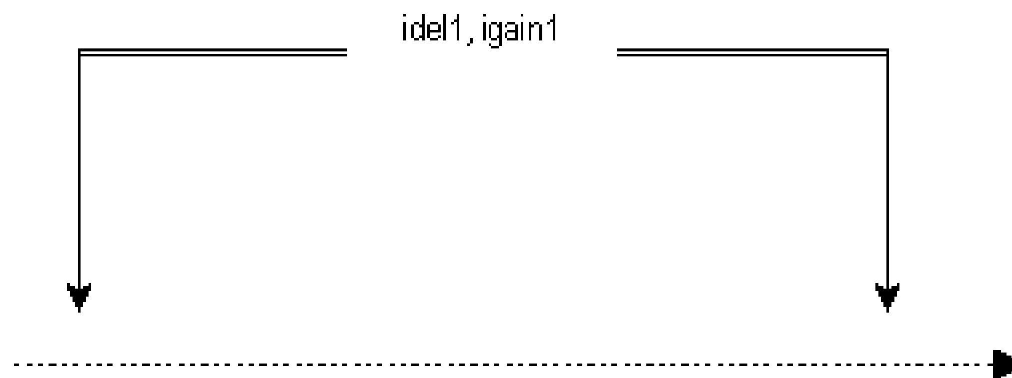
imaxdel -- will be necessary if k-rate delays are implemented. Not currently used.

istor -- Skip initialization if non-zero (default: 0).

Performance

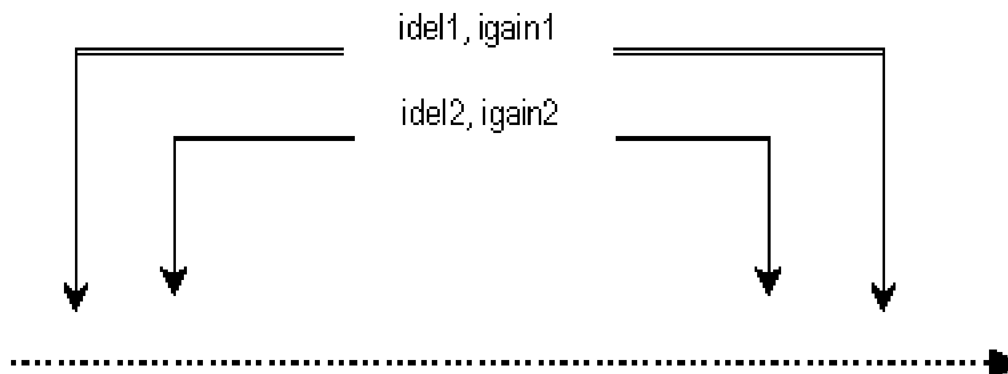
asig -- input signal

If *imode* = 1, the filter takes the form:



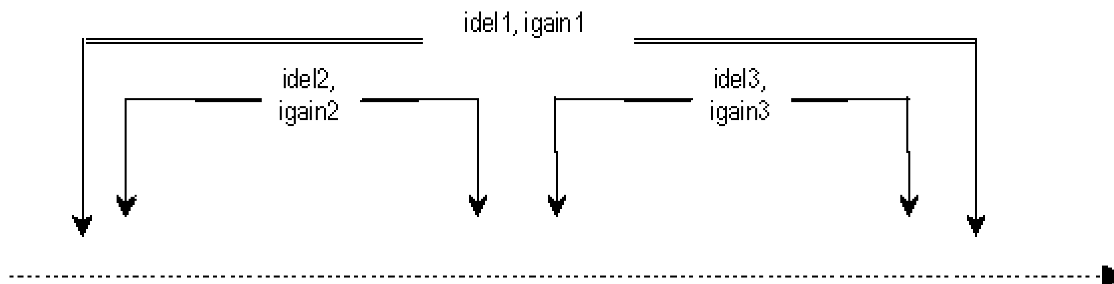
Picture of imode 1 filter.

If *imode* = 2, the filter takes the form:



Picture of imode 2 filter.

If *imode* = 3, the filter takes the form:



Picture of imode 3 filter.

Examples

Here is an example of the nestedap opcode. It uses the file *nestedap.csd* [examples/nestedap.csd], and *beats.wav* [examples/beats.wav].

Exemple 311. Example of the nestedap opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nestedap.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 5
```

```
  insnd     =           p4
  gasig     diskin insnd, 1
endin

instr 10
  imax      =           1
  idel1     =           p4/1000
  igain1    =           p5
  idel2     =           p6/1000
  igain2    =           p7
  idel3     =           p8/1000
  igain3    =           p9
  idel4     =           p10/1000
  igain4    =           p11
  idel5     =           p12/1000
  igain5    =           p13
  idel6     =           p14/1000
  igain6    =           p15

  afdbk     init 0

  aout1     nestedap gasig+afdbk*.4, 3, imax, idel1, igain1, idel2, igain2, idel3, igain3

  aout2     nestedap aout1, 2, imax, idel4, igain4, idel5, igain5

  aout      nestedap aout2, 1, imax, idel6, igain6

  afdbk     butterlp aout, 1000

            outs gasig+(aout+aout1)/2, gasig-(aout+aout1)/2

gasig     =           0
endin

</CsInstruments>
<CsScore>

f1 0 8192 10 1

; Diskin
;    Sta  Dur   Soundin
i5   0    3     "beats.wav"

; Reverb
;    St  Dur   Del1 Gn1  Del2  Gn2  Del3  Gn3  Del4  Gn4  Del5  Gn5  Del6  Gn6
i10 0    4     97   .11  23    .07  43    .09  72    .2   53    .2   119   .3
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Hans Mikelson
February 1999

New in Csound version 3.53

The example was updated May 2002, thanks to Hans Mikelson

nlfilt

nlfilt — A filter with a non-linear effect.

Description

Implements the filter:

Y{n} =a Y{n-1} + b Y{n-2} + d Y^2{n-L} + X{n} - C

described in Dobson and Fitch (ICMC'96)

Syntax

```
ares nlfilt ain, ka, kb, kd, kC, kL
```

Performance

1. Non-linear effect. The range of parameters are:

   a = b = 0
   d = 0.8, 0.9, 0.7
   C = 0.4, 0.5, 0.6
   L = 20

   This affects the lower register most but there are audible effects over the whole range. We suggest that it may be useful for coloring drums, and for adding arbitrary highlights to notes.

2. Low Pass with non-linear. The range of parameters are:

   a = 0.4
   b = 0.2
   d = 0.7
   C = 0.11
   L = 20, ... 200

   There are instability problems with this variant but the effect is more pronounced of the lower register, but is otherwise much like the pure comb. Short values of *L* can add attack to a sound.

3. High Pass with non-linear. The range of parameters are:

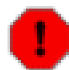
   a = 0.35
   b = -0.3
   d = 0.95
   C = 0,2, ... 0.4

L = 200

4. High Pass with non-linear. The range of parameters are:

   a = 0.7
   b = -0.2, ... 0.5
   d = 0.9
   C = 0.12, ... 0.24
   L = 500, 10

   The high pass version is less likely to oscillate. It adds scintillation to medium-high registers. With a large delay *L* it is a little like a reverberation, while with small values there appear to be formant-like regions. There are arbitrary color changes and resonances as the pitch changes. Works well with individual notes.



Warning

The "useful" ranges of parameters are not yet mapped.

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
1997

New in version 3.44

noise

noise — Un générateur de bruit blanc avec un filtre passe-bas à RII.

Description

Un générateur de bruit blanc avec un filtre passe-bas à RII.

Syntaxe

```
ares noise xamp, kbeta
```

Exécution

xamp -- amplitude de la sortie finale

kbeta -- beta du filtre passe-bas. Doit être compris entre -1 et 1.

L'équation du filtre est :

$$y_n = \sqrt{(1-\beta^2)} * x_n + \beta y_{(n-1)}$$

où x_n est le bruit blanc original et y_n est le bruit filtré. Plus # est élevé, plus basse est la fréquence de coupure du filtre. La fréquence de coupure vaut approximativement *sr* * ((1 - *kbeta*) / 2).

Exemples

Voici un exemple de l'opcode noise. Il utilise le fichier *noise.csd* [examples/noise.csd].

Exemple 312. Exemple de l'opcode noise.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  kamp = 30000

  ; Change the beta value linearly from 0 to 1.
  kbeta line 0, p3, 1

  a1 noise kamp, kbeta
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode noise dans lequel on contrôle le paramètre *kbeta* au moyen d'une interface graphique. Il utilise le fichier *noise-2.csd* [examples/noise-2.csd].

Exemple 313. Exemple de l'opcode noise contrôlé par une interface graphique.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac     ;  -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

FLpanel "noise", 200, 50, -1 , -1
    gkbeta, gislider1 FLslider "kbeta", -1, 1, 0, 5, -1, 180, 20, 10, 10
FLpanelEnd
FLrun

instr 1
  iamp = 0dbfs / 4  ; Peaks 12 dB below 0dbfs
  print iamp

  a1 noise iamp, gkbeta
  printk2 gkbeta
  outs a1,a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch
Université de Bath, Codemist. Ltd.
Bath, UK
Décembre 2000

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.10 de Csound

noteoff

noteoff — Send a noteoff message to the MIDI OUT port.

Description

Send a noteoff message to the MIDI OUT port.

Syntax

```
noteoff ichn, inum, ivel
```

Initialization

ichn -- MIDI channel number (1-16)

inum -- note number (0-127)

ivel -- velocity (0-127)

Performance

noteon (i-rate note on) and *noteoff* (i-rate note off) are the simplest MIDI OUT opcodes. *noteon* sends a MIDI noteon message to MIDI OUT port, and *noteoff* sends a noteoff message. A *noteon* opcode must always be followed by an *noteoff* with the same channel and number inside the same instrument, otherwise the note will play endlessly.

These *noteon* and *noteoff* opcodes are useful only when introducing a *timout* statement to play a non-zero duration MIDI note. For most purposes, it is better to use *noteondur* and *noteondur2*.

See Also

noteon, *noteondur*, *noteondur2*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

noteon

noteon — Send a noteon message to the MIDI OUT port.

Description

Send a noteon message to the MIDI OUT port.

Syntax

```
noteon ichn, inum, ivel
```

Initialization

ichn -- MIDI channel number (1-16)

inum -- note number (0-127)

ivel -- velocity (0-127)

Performance

noteon (i-rate note on) and *noteoff* (i-rate note off) are the simplest MIDI OUT opcodes. *noteon* sends a MIDI noteon message to MIDI OUT port, and *noteoff* sends a noteoff message. A *noteon* opcode must always be followed by an *noteoff* with the same channel and number inside the same instrument, otherwise the note will play endlessly.

These *noteon* and *noteoff* opcodes are useful only when introducing a *timout* statement to play a non-zero duration MIDI note. For most purposes, it is better to use *noteondur* and *noteondur2*.

See Also

noteoff, *noteondur*, *noteondur2*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

noteondur

noteondur — Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

Description

Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

Syntax

```
noteondur ichn, inum, ivel, idur
```

Initialization

ichn -- MIDI channel number (1-16)

inum -- note number (0-127)

ivel -- velocity (0-127)

idur -- how long, in seconds, this note should last.

Performance

noteondur (i-rate note on with duration) sends a noteon and a noteoff MIDI message both with the same channel, number and velocity. Noteoff message is sent after *idur* seconds are elapsed by the time *noteondur* was active.

noteondur differs from *noteondur2* in that *noteondur* truncates note duration when current instrument is deactivated by score or by real-time playing, while *noteondur2* will extend performance time of current instrument until *idur* seconds have elapsed. In real-time playing, it is suggested to use *noteondur* also for undefined durations, giving a large *idur* value.

Any number of *noteondur* opcodes can appear in the same Csound instrument, allowing chords to be played by a single instrument.

Examples

Here is an example of the noteondur opcode. It uses the file *noteondur.csd* [examples/noteondur.csd].

Exemple 314. Example of the noteondur opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1  ;Turned on by MIDI notes on channel 1

  ifund   notnum
  ivel         veloc
  idur = 1

  ;chord with single key
  noteondur         1, ifund,   ivel, idur
  noteondur         1, ifund+3, ivel, idur
  noteondur         1, ifund+7, ivel, idur
  noteondur         1, ifund+9, ivel, idur

endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for 60 seconds.

i1 0 60

</CsScore>
</CsoundSynthesizer>
```

See Also

noteoff, *noteon*, *noteondur2*, *midion*, *midion2*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

noteondur2

noteondur2 — Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

Description

Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

Syntax

```
noteondur2 ichn, inum, ivel, idur
```

Initialization

ichn -- MIDI channel number (1-16)

inum -- note number (0-127)

ivel -- velocity (0-127)

idur -- how long, in seconds, this note should last.

Performance

noteondur2 (i-rate note on with duration) sends a noteon and a noteoff MIDI message both with the same channel, number and velocity. Noteoff message is sent after *idur* seconds are elapsed by the time *noteondur2* was active.

noteondur differs from *noteondur2* in that *noteondur* truncates note duration when current instrument is deactivated by score or by real-time playing, while *noteondur2* will extend performance time of current instrument until *idur* seconds have elapsed. In real-time playing, it is suggested to use *noteondur* also for undefined durations, giving a large *idur* value.

Any number of *noteondur2* opcodes can appear in the same Csound instrument, allowing chords to be played by a single instrument.

Examples

Here is an example of the noteondur2 opcode. It uses the file *noteondur2.csd* [examples/noteondur2.csd].

Exemple 315. Example of the noteondur2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1

  ifund   notnum
  ivel         veloc
  idur = 1

  ;chord with single key
  noteondur2         1, ifund,   ivel, idur
  noteondur2         1, ifund+3, ivel, idur
  noteondur2         1, ifund+7, ivel, idur
  noteondur2         1, ifund+9, ivel, idur

endin


</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>
```

See Also

noteoff, *noteon*, *noteondur*, *midion*, *midion2*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

notnum

notnum — Donne un numéro de note à partir d'un évènement MIDI.

Description

Donne un numéro de note à partir d'un évènement MIDI.

Syntaxe

```
ival notnum
```

Exécution

Donne la valeur de l'octet MIDI (0 - 127) représentant le numéro de note de l'évènement courant.

Exemples

Voici un exemple de l'opcode notnum. Il utilise le fichier *notnum.csd* [examples/notnum.csd].

Exemple 316. Exemple de l'opcode notnum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o notnum.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 notnum

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode notnum utilisé pour produire une sortie audio sortie. Il utilise le fichier *notnum_complex.csd* [examples/notnum_complex.csd]

Exemple 317. Exemple complexe de l'opcode notnum.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac          -iadc      -d          -M0  ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr      =       44100
ksmps   =       10
nchnls  =       2

; Set MIDI channel 1 to play instr 1.
        massign 1, 1

                instr   1

; Returns MIDI note number - an integer in range (0-127)
iNum    notnum

; Convert MIDI note number to Hz
iHz     = (440.0*exp(log(2.0)*((iNum)-69.0)/12.0))

; Generate audio by indexing a table; fixed amplitude.
aosc    oscil   10000, iHz, 1

; Since there is no enveloping, there will be clicks.
        outs    aosc, aosc

                endin

</CsInstruments>
<CsScore>

; Generate a Sine-wave to be indexed at audio rate
;  by the oscil opcode.
f1      0       16384   10      1

; Keep the score "open" for 1 hour so that MIDI
;  notes can allocate new note events, arbitrarily.
f0      3600

e
</CsScore>
</CsoundSynthesizer>
```

Voir aussi

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Exemples écrits par Kevin Conder et David Akbari.

nreverb

nreverb — A reverberator consisting of 6 parallel comb-lowpass filters.

Description

This is a reverberator consisting of 6 parallel comb-lowpass filters being fed into a series of 5 allpass filters. *nreverb* replaces *reverb2* (version 3.48) and so both opcodes are identical.

Syntax

```
ares nreverb asig, ktime, khdif [, iskip] [,inumCombs] [, ifnCombs] \
      [, inumAlpas] [, ifnAlpas]
```

Initialization

iskip (optional, default=0) -- Skip initialization if present and non-zero.

inumCombs (optional) -- number of filter constants in comb filter. If omitted, the values default to the nreverb constants. New in Csound version 4.09.

ifnCombs - function table with *inumCombs* comb filter time values, followed the same number of gain values. The ftable should not be rescaled (use negative fgen number). Positive time values are in seconds. The time values are converted internally into number of samples, then set to the next greater prime number. If the time is negative, it is interpreted directly as time in sample frames, and no processing is done (except negation). New in Csound version 4.09.

inumAlpas, *ifnAlpas* (optional) -- same as *inumCombs/ifnCombs*, for allpass filter. New in Csound 4.09.

Performance

The input signal asig is reverberated for *ktime* seconds. The parameter *khdif* controls the high frequency diffusion amount. The values of *khdif* should be from 0 to 1. If *khdif* is set to 0 the all the frequencies decay with the same speed. If *khdif* is 1, high frequencies decay faster than lower ones. If *ktime* is inadvertently set to a non-positive number, *ktime* will be reset automatically to 0.01. (New in Csound version 4.07.)

As of Csound version 4.09, *nreverb* may read any number of comb and allpass filter from an ftable.

Examples

Here is a simple example of the nreverb opcode. It uses the file *nreverb.csd* [examples/nreverb.csd].

Exemple 318. Simple example of the nreverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  a1 oscil 10000, 440, 1
  a2 nreverb a1, 2.5, .3
  out a1 + a2 * .2
endin


</CsInstruments>
<CsScore>

; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

i 1 0.0 0.5
i 1 1.0 0.5
i 1 2.0 0.5
i 1 3.0 0.5
i 1 4.0 0.5
e


</CsScore>
</CsoundSynthesizer>
```

Here is an example of the nreverb opcode using an ftable for filter constants. It uses the file *nreverb_ftable.csd* [examples/nreverb_ftable.csd], and *beats.wav* [examples/beats.wav].

Exemple 319. An example of the nreverb opcode using an ftable for filter constants.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb_ftable.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  a1  soundin "beats.wav"
  a2  nreverb a1, 1.5, .75, 0, 8, 71, 4, 72
  out a1 + a2 * .4
endin


</CsInstruments>
<CsScore>

; freeverb time constants, as direct (negative) sample, with arbitrary gains
f71 0 16   -2  -1116 -1188 -1277 -1356 -1422 -1491 -1557 -1617  0.8  0.79  0.78  0.77  0.76  0.75  0.74

f72 0 16   -2  -556 -441 -341 -225  0.7  0.72  0.74  0.76
```

```
i1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Authors: Paris Smaragdis (*reverb2*)
MIT, Cambridge
1995

Author: Richard Karpen (*nreverb*)
Seattle, Wash
1998

nrpn

nrpn — Sends a Non-Registered Parameter Number to the MIDI OUT port.

Description

Sends a NPRN (Non-Registered Parameter Number) message to the MIDI OUT port each time one of the input arguments changes.

Syntax

```
nrpn kchan, kparmnum, kparmvalue
```

Performance

kchan -- MIDI channel (1-16)

kparmnum -- number of NRPN parameter

kparmvalue -- value of NRPN parameter

This opcode sends new message when the MIDI translated value of one of the input arguments changes. It operates at k-rate. Useful with the MIDI instruments that recognize NRPNs (for example with the newest sound-cards with internal MIDI synthesizer such as SB AWE32, AWE64, GUS etc. in which each patch parameter can be changed during the performance via NRPN)

Credits

Author: Gabriel Maldonado
Italy
1998

New in Csound version 3.492

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

nsamp

nsamp — Returns the number of samples loaded into a stored function table number.

Description

Returns the number of samples loaded into a stored function table number.

Syntax

```
nsamp(x) (init-rate args only)
```

Performance

Returns the number of samples loaded into stored function table number *x* by GEN01. This is useful when a sample is shorter than the power-of-two function table that holds it. New in Csound version 3.49.

As of Csound version 5.02, *ftlen* works with deferred-length function tables (see GEN01).

nsamp differs from *ftlen* in that *nsamp* gives the number of sample frames loaded, while *ftlen* gives the total number of samples. For example, with a stereo sound file of 10000 samples, ftlen() would return 19999 (i.e. a total of 20000 mono samples, not including a guard point), but nsamp() returns 10000.

Examples

Here is an example of the nsamp opcode. It uses the file *nsamp.csd* [examples/nsamp.csd], and *mary.wav* [examples/mary.wav].

Exemple 320. Example of the nsamp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nsamp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the size (in samples) of Table #1.
  isz = nsamp(1)
  print isz
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: Use an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Since the audio file « mary.wav » has 154390 samples, its output should include a line like this:

```
instr 1:  isz = 154390.000
```

See Also

ftchnls, *ftlen*, *ftlptim*, *ftsr*

Credits

Author: Gabriel Maldonado
Italy
October 1998

Example written by Kevin Conder.

nstrnum

nstrnum — Returns the number of a named instrument.

Description

Returns the number of a named instrument.

Syntax

```
insno nstrnum "name"
```

Initialization

insno -- the instrument number of the named instrument.

Performance

"name" -- the named instrument's name.

If an instrument with the specified name does not exist, an init error occurs, and -1 is returned.

Credits

Author: Istvan Varga
New in version 4.23
Written in the year 2002.

ntrpol

ntrpol — Calculates the weighted mean value of two input signals.

Description

Calculates the weighted mean value (i.e. linear interpolation) of two input signals

Syntax

```
ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
```

```
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
```

```
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]
```

Initialization

imin -- minimum xpoint value (optional, default 0)

imax -- maximum xpoint value (optional, default 1)

Performance

xsig1, *xsig2* -- input signals

xpoint -- interpolation point between the two values

ntrpol opcode outputs the linear interpolation between two input values. *xpoint* is the distance of evaluation point from the first value. With the default values of *imin* and *imax*, (0 and 1) a zero value indicates no distance from the first value and the maximum distance from the second one. With a 0.5 value, *ntrpol* will output the mean value of the two inputs, indicating the exact half point between *xsig1* and *xsig2*. A 1 value indicates the maximum distance from the first value and no distance from the second one. The range of *xpoint* can be also defined with *imin* and *imax* to make its management easier.

These opcodes are useful for crossfading two signals.

Credits

Author: Gabriel Maldonado
Italy
October 1998

New in Csound version 3.49

octave

octave — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

Syntaxe

```
octave(x)
```

Cette fonction travaille aux taux-i, -k et -a.

Initialisation

x -- une valeur exprimée en octaves.

Exécution

La valeur retournée par la fonction *octave* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre d'octaves spécifié.

Exemples

Voici un exemple de l'opcode octave. Il utilise le fichier *octave.csd* [examples/octave.csd].

Exemple 321. Exemple de l'opcode octave.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octave.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; The root note is A above middle-C (440 Hz)
  iroot = 440

  ; Raise the root note by two octaves.
  ioctaves = 2

  ; Calculate the new note.
```

```
  ifactor = octave(ioctaves)
  inew = iroot * ifactor

  ; Print out of all of the values.
  print iroot
  print ifactor
  print inew
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1:  iroot = 440.000
instr 1:  ifactor = 4.000
instr 1:  inew = 1760.149
```

Voir Aussi

cent, *db*, *semitone*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

octcps

octcps — Convertit des cycles par seconde en valeur octave-point-partie-décimale.

Description

Convertit des cycles par seconde en valeur octave-point-partie-décimale.

Syntaxe

```
octcps (cps)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

octcps et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 13. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode octcps. Il utilise le fichier *octcps.csd* [examples/octcps.csd].

Exemple 322. Exemple de l'opcode octcps.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octcps.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Convert a cycles-per-second value into an
  ; octave value.
  icps = 440
  ioct = octcps(icps)

  print ioct
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

Voir Aussi

cpsoct, *cpspch*, *octpch*, *pchoct*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Crédits

Exemple écrit par Kevin Conder.

octmidi

octmidi — Get the note number, in octave-point-decimal units, of the current MIDI event.

Description

Get the note number, in octave-point-decimal units, of the current MIDI event.

Syntax

```
ioct octmidi
```

Performance

Get the note number of the current MIDI event, expressed in octave-point-decimal units, for local processing.

octmidi vs. octmidinn

The *octmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *octmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *octmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *octmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

Examples

Here is an example of the octmidi opcode. It uses the file *octmidi.csd* [examples/octmidi.csd].

Exemple 323. Example of the octmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; This example expects MIDI note inputs on channel 1
  i1 octmidi

  print i1
endin


</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e


</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

octmidib

octmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in octave-point-decimal.

Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in octave-point-decimal.

Syntax

```
ioct octmidib [irange]
```

```
koct octmidib [irange]
```

Initialization

irange (optional) -- the pitch bend range in semitones

Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in octave-point-decimal units. Available as an i-time value or as a continuous k-rate value.

Examples

Here is an example of the octmidib opcode. It uses the file *octmidib.csd* [examples/octmidib.csd].

Exemple 324. Example of the octmidib opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidib.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; This example expects MIDI note inputs on channel 1
  i1 octmidib
```

```
  print i1
endin


</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e


</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *pchbend*, *pchmidi*, *pchmidib*, *veloc*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

octmidinn

octmidinn — Convertit un numéro de note Midi en octave-point-partie-décimale.

Description

Convertit un numéro de note Midi en octave-point-partie-décimale.

Syntaxe

```
octmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

octmidinn est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave-point-partie-décimale de Csound. Cette conversion suppose que le do médian (8.000 en *oct*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.

octmidinn vs. octmidi

L'opcode *octmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un évènement Midi, un évènement de partition, un évènement en ligne, ou depuis un autre instrument. La valeur d'entrée de *octmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *octmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *octmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

octmidinn et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 14. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode octmidinn. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

Exemple 325. Exemple de l'opcode octmidinn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
  ; i-time loop to print conversion table
  imidiNN =   0
  loop1:
    icps   = cpsmidinn(imidiNN)
    ioct   = octmidinn(imidiNN)
    ipch   = pchmidinn(imidiNN)

    print   imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
  if (imidiNN < 128) igoto loop1
```

```
endin

instr 2
  ; test k-rate converters
  kMiddleC  =   60
  kcps  = cpsmidinn(kMiddleC)
  koct  = octmidinn(kMiddleC)
  kpch  = pchmidinn(kMiddleC)

  printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cpsmidinn, *pchmidinn*, *octmidi*, *notnum*, *cpspch*, *cpsoct*, *octcps*, *octpch*, *pchoct*

Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

octpch

octpch — Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

Description

Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

Syntaxe

```
octpch (pch)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

octpch et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 15. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode octpch. Il utilise le fichier *octpch.csd* [examples/octpch.csd].

Exemple 326. Exemple de l'opcode octpch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octpch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Convert a pitch-class value into an
  ; octave-point-decimal value.
  ipch = 8.09
  ioct = octpch(ipch)

  print ioct
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

Voir Aussi

cpsoct, *cpspch*, *octcps*, *pchoct*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Crédits

Exemple écrit par Kevin Conder.

opcode

opcode — Commence un bloc d'opcode défini par l'utilisateur.

Définir des opcodes

Les instructions *opcode* et *endop* permettent de définir un nouvel opcode qui peut être utilisé de la même façon qu'un opcode original de Csound. Ces blocs d'opcode ressemblent beaucoup aux instruments (et sont, en fait, implémentés comme des instruments spéciaux), mais on ne peut pas les appeler comme des intruments normaux, par exemple avec des *instructions i*.

Un bloc d'opcode défini par l'utilisateur doit précéder l'instrument (ou l'opcode) depuis lequel on l'utilise. Mais un opcode peut aussi s'appeler lui-même. Cela permet une récursivité dont la profondeur n'est limitée que par la mémoire disponible. De plus, on peut, à titre expérimental, exécuter l'opcode défini à un taux de contrôle plus élevé que la valeur de *kr* spécifiée dans l'en-tête de l'orchestre.

Comme pour les intruments, les variables et les étiquettes d'un bloc d'opcode défini par l'utilisateur sont locales et ne sont pas visible depuis l'instrument appelant (de même que l'opcode n'a pas accès aux variables de l'instrument qui l'a appelé).

Cependant, certains paramètres sont copiés automatiquement à l'initialisation :

- tous les p-champs (*p1* inclus)
- le temps supplémentaire (voir aussi *xtratim*, *linsegr*, et les opcodes correspondants). Ceci peut affecter le fonctionnement de *linsegr*/*expsegr*/*linenr*/*envlpxr* dans le bloc d'opcode défini par l'utilisateur.
- les paramètres MIDI, s'il y en a.

Le drapeau de release (voir l'opcode *release*) est également copié durant l'exécution.

La modification de la durée de la note dans la définition de l'opcode en assignant une valeur à *p3*, ou l'utilisation de *ihold*, *turnoff*, *xtratim*, *linsegr*, ou d'autres opcodes similaires affecteront aussi l'instrument appelant. Les changements sur des contrôleurs MIDI (par exemple avec *ctrlinit*) s'appliqueront aussi à l'instrument qui a appelé l'opcode.

Utilisez l'opcode *setksmps* pour fixer la valeur locale de *ksmps*.

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.

Notes

- *xin* et *xout* ne doivent être appelés qu'une seule fois, et *xin* doit précéder *xout*, sinon une erreur d'initialisation et une désactivation de l'instrument courant peuvent se produire.
- Ces deux opcodes n'agissent qu'à l'initialisation. La copie durant l'exécution est réalisée par l'appel de l'opcode de l'utilisateur. Cela signifie que sauter *xin* ou *xout* avec *kgoto* n'a aucun effet, alors que les sauter avec *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

Syntaxe

```
opcode nom, outtypes, intypes
```

Initialisation

nom -- nom de l'opcode. Il est constitué de n'importe quelle combinaison de lettres, chiffres et traits de soulignement mais il ne doit pas commencer par un chiffre. Si un opcode du même nom existe déjà, il est redéfini (un avertissement est imprimé dans ce cas). Certains mots réservés (comme *instr* et *endin*) ne peuvent pas être redéfinis.

intypes -- liste des types en entrée, combinaison de caractères pris parmi : a, k, K, i, o, p, et j. Un caractère 0 unique peut être utilisé s'il n'y a pas d'argument en entrée. Il n'y a *pas* besoin d'apostrophes doubles et de délimiteurs (comme la virgule).

La signification des différent *intypes* est montrée dans le tableau suivant :

Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
j	facultatif de taux-i, -1 par défaut	taux-i, constante	initialisation
k	variable de taux-k	taux-k et -i, constante	taux-k
K	taux-k avec initialisation	taux-k et -i, constante	taux-i et taux-k
o	facultatif à l'initialisation, 0 par défaut	taux-i, constante	initialisation
p	facultatif à l'initialisation, 1 par défaut	taux-i, constante	initialisation
S	variable chaîne de caractères	chaîne de caractères de taux-i	initialisation

Le nombre maximum d'arguments en entrée autorisé est 256.

outtypes -- liste des types en sortie. Le format est le même que celui utilisé pour *intypes*.

Voici les *outtypes* disponibles :

Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
k	variable de taux-k	taux-k	taux-k
K	taux-k avec initialisation	taux-k	taux-i et taux-k

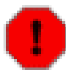
Le nombre maximum d'arguments en sortie autorisé est 256.

iksmps (facultatif, 0 par défaut) -- fixe la valeur locale de *ksmps*. Doit être un nombre entier positif, et le *ksmps* de l'instrument appelant doit être un multiple entier de cette valeur. Par exemple, si *ksmps* vaut 10 dans l'instrument depuis lequel l'opcode a été appelé, les valeurs permises pour *iksmps* sont 1, 2, 5, et 10.

Si *iksmps* vaut zéro, le *ksmps* de l'instrument ou de l'opcode appelant est utilisé (c'est le comportement par défaut).

Note

Le *ksmps* local est implémenté en divisant une période de contrôle en sous-périodes-k plus petites et en modifiant temporairement les variables globales internes de Csound. Ceci nécessite aussi la conversion du taux des arguments d'entrée et de sortie de taux-k (les variables d'entrée reçoivent la même valeur dans tous les sous-périodes-k, tandis que les valeurs de sortie ne sont écrites que pendant la dernière).



Avertissement au sujet du *ksmps* local

Lorsque le *ksmps* local est différent du *ksmps* de l'orchestre (celui spécifié dans l'en-tête de l'orchestre), il ne faut pas utiliser d'opération globale de taux-a dans le bloc d'opcode défini par l'utilisateur.

Ceci comprend :

- tous les accès aux variables « ga »
- les opcodes zak de taux-a (*zar*, *zaw*, etc.)
- *tablera* et *tablewa* (ces deux opcodes peuvent fonctionner en fait, mais il faut prendre des précautions)
- La famille d'ocpode *in* et *out* (ils lisent depuis et écrivent dans des tampons globaux de taux-a)

En général, il faut utiliser le *ksmps* local avec précaution car c'est une fonctionnalité expérimentale, bien qu'elle fonctionne correctement dans la plupart des cas.

L'instruction *setksmps* peut être utilisée pour fixer la valeur du *ksmps* local du bloc d'opcode défini par l'utilisateur. Elle a un paramètre de taux-i spécifiant la nouvelle valeur de *ksmps* (qui reste inchangée si l'on utilise zéro, voir aussi les notes au sujet de *iksmps* ci-dessus). *setksmps* doit être utilisé avant tout autre opcode (mais il est autorisé après *xin*), autrement des résultats imprévisibles peuvent se produire.

On peut lire les paramètres d'entrée avec l'opcode *xin*, et la sortie est écrite par l'opcode *xout*. On ne doit utiliser qu'une seule instance de ces unités, car *xout* écrase la sortie sans accumuler les valeurs. Le nombre et le type des arguments pour *xin* et *xout* doit être le même que dans la déclaration du bloc d'opcode défini par l'utilisateur (voir les tableaux ci-dessus).

Les arguments d'entrée et de sortie doivent se conformer à la définition à la fois en nombre (sauf si des entrées de taux-i facultatives sont utilisées) et en genre. Un paramètre d'entrée facultatif de taux-i (*iksmps*) est automatiquement ajouté à la liste des *intypes* et (comme pour setksmps) fixe la valeur du *ksmps* local.

Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode  nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
[setksmps  iksmps]
... the rest of the instrument's code.
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xoutarg1] [, xoutarg2] ... [xoutargN] nom  [xinarg1] [, xinarg2] ... [xinargN] [, iksmps]
```

Note

L'opcode est toujours appelé à la fois durant l'initialisation et durant l'exécution, même s'il n'y a pas d'arguments de taux-k ou -a. Si l'on sait que plusieurs opcodes définis par l'utilisateur n'ont pas d'effet durant l'exécution (taux-k) dans un instrument, on peut épargner du temps CPU en sautant ces groupes d'opcodes avec *kgoto*.

Exemples

Voici un exemple d'opcode défini par l'utilisateur. Il utilise le fichier *opcode.csd* [examples/opcode_example.csd].

Exemple 327. Exemple d'opcode défini par l'utilisateur.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o opcode_example.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr      =  44100
ksmps   =  50
nchnls  =  1

/* example opcode 1: simple oscillator */

        opcode Oscillator, a, kk

kamp, kcps      xin             ; read input parameters
a1      vco2 kamp, kcps         ; sawtooth oscillator
        xout a1                 ; write output

        endop

/* example opcode 2: lowpass filter with local ksmps */

        opcode Lowpass, a, akk

        setksmps 1              ; need sr=kr
ain, ka1, ka2   xin             ; read input parameters
aout    init 0                  ; initialize output
aout    =  ain*ka1 + aout*ka2   ; simple tone-like filter
        xout aout               ; write output
```

```
        endop

/* example opcode 3: recursive call */

        opcode RecursiveLowpass, a, akkpp

ain, ka1, ka2, idep, icnt       xin     ; read input parameters
        if (icnt >= idep) goto skip1    ; check if max depth reached
ain     RecursiveLowpass ain, ka1, ka2, idep, icnt + 1
skip1:
aout    Lowpass ain, ka1, ka2           ; call filter
        xout aout                       ; write output

        endop

/* example opcode 4: de-click envelope */

        opcode DeClick, a, a

ain     xin
aenv    linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
        xout ain * aenv         ; apply envelope and write output

        endop

/* instr 1 uses the example opcodes */

        instr 1

kamp    =  20000                ; amplitude
kcps    expon 50, p3, 500       ; pitch
a1      Oscillator kamp, kcps                   ; call oscillator
kflt    linseg 0.4, 1.5, 0.4, 1, 0.8, 1.5, 0.8  ; filter envelope
a1      RecursiveLowpass a1, kflt, 1 - kflt, 10 ; 10th order lowpass
a1      DeClick a1
        out a1

        endin

</CsInstruments>
<CsScore>

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

endop, *setksmps*, *xin*, *xout*

Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

OSCsend

OSCsend — Sends data to other processes using the OSC protocol

Description

Uses the OSC protocol to send message to other OSC listening processes.

Syntax

```
OSCsend kwhen, ihost, iport, idestination, itype [, kdata1, kdata2, ...]
```

Initialization

ihost -- a string that is the intended host computer domain name. An empty string is interpreted as the current computer.

iport -- the number of the port that is used for the communication.

idest -- a string that is the destination address. This takes the form of a file name with directories. Csound just passes this string to the raw sending code and makes no interpretation.

itype -- a string that indicates the types of the optional arguments that are read at k-rate. The string can contain the characters "bcdfilmst" which stand for Boolean, character, double, float, 32-bit integer, 64-bit integer, MIDI, string and timestamp.

Performance

kwhen -- a message is sent whenebver this value changes. A message will always be sent on the first call.

The data is taken from the k-values that follow the format string. In a similar way to a printf format, the characters in order determine how the argument is interpreted. Note that a time stamp takes two arguments.

Example

The example shows a simple instrument, which when called, sends a group of 3 messages to a computer called "xenakis", on port 7770, to be read by a process that recognises /foo/bar as its address.

```
instr   1
    OSCsend     1, "xenakis.cs.bath.ac.uk",7770, "/foo/bar", "sis", "FOO", 42, "bar"
endin
```

See the entry for *OSClisten*, for an example of send/recieve usage using OSC.

See Also

OSClisten, *OSCinit*

Credits

Author: John ffitch
2005

OSCinit

OSCinit — Start a listening process for OSC messages to a particular port.

Description

Starts a listening process, which can be used by OSClisten.

Syntax

```
ihandle OSCinit iport
```

Initialization

ihandle -- handle returned that can be passed to any number of OSClisten opcodes to receive messages on this port.

iport -- the port on which to listen.

Performance

The listener runs in the background. See OSClisten for details.

Example

The example shows a pair of floating point numbers being received on port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinit 7770

  instr   1
    kf1 init 0
    kf2 init 0
nxtmsg:
    kk  OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
    printk 0,kf1
    printk 0,kf2
    kgoto nxtmsg
ex:
  endin
```

Credits

Author: John ffitch
2005

OSClisten

OSClisten — Listen for OSC messages to a particular path.

Description

On each k-cycle looks to see if an OSC message has been send to a given path of a given type.

Syntax

```
kans OSClisten ihandle, idest, itype [, xdata1, xdata2, ...]
```

Initialization

ihandle -- a handle returned by an earlier call to OSCinit, to associate OSClisten with a particular port number.

idest -- a string that is the destination address. This takes the form of a file name with directories. Csound uses this address to decide if messages are meant for csound.

itype -- a string that indicates the types of the optional arguments that are to be read. The string can contain the characters "cdfhis" which stand for character, double, float, 64-bit integer, 32-bit integer, and string. All types other than 's' require a k-rate variable, while 's' requires a string variable.

A handler is inserted into the listener (see OSCinit) to intercept messages of this pattern.

Performance

kans -- set to 1 if a new message was received, or zero if not. If multiple messages are received in a single control period, the messages are buffered, and OSClisten can be called again until zero is returned.

If there was a message the *xdata* variables are set to the incoming values, as interpretted by the *itype* parameter. Note that although the *xdata* variables are on the right of an operation they are actually outputs, and so must be variables of type k, gk, S, or gS, and may need to be declared with init, or = in the case of string variables, before calling OSClisten.

Example

The example shows a pair of floating point numbers being received on port 7770.

```
  sr = 44100
  ksmps = 100
  nchnls = 2

  gihandle OSCinit 7770

  instr   1
    kf1 init 0
    kf2 init 0
nxtmsg:
    kk  OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
    printk 0,kf1
    printk 0,kf2
```

```
    kgoto nxtmsg
ex:
  endin
```

Below are two .csd files which demonstrate the usage of the OSC opcodes. They use the files *OSCmidisend.csd* [examples/OSCmidisend.csd] and *OSCmidircv.csd* [examples/OSCmidircv.csd].

Exemple 328. Example of the OSC opcodes.

The following two .csd files demonstrate the usage of the OSC opcodes in csound. The first file, *OSCmidisend.csd* [examples/OSCmidisend.csd], transforms received real-time MIDI messages into OSC data. The second file, *OSCmidircv.csd* [examples/OSCmidircv.csd], can take these OSC messages, and intrepret them to generate sound from note messages, and store controller values. It will use controller number 7 to control volume. Note that these files are designed to be on the same machine, but if a different host address (in the IPADDRESS macro) is used, they can be separate machines on a network, or connected through the internet.

CSD file to send OSC messages:

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
<CsInstruments>

  sr      =  44100
  ksmps      =  128
  nchnls     =  1

; Example by David Akbari 2007
; Modified by Jonathan Murphy
; Use this file to generate OSC events for OSCmidircv.csd

#define IPADDRESS # "localhost" #
#define PORT                 # 47120 #

turnon 1000


    instr 1000

  kst, kch, kd1, kd2  midiin

  OSCsend   kst+kch+kd1+kd2, $IPADDRESS, $PORT, "/midi", "iiii", kst, kch, kd1, kd2

    endin


</CsInstruments>
<CsScore>
f 0 3600  ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>
```

CSD file to receive OSC messages:

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
</CsOptions>
```

```
<CsInstruments>

  sr      =  44100
  ksmps   =  128
  nchnls  =  1

; Example by Jonathan Murphy and Andres Cabrera 2007
; Use file OSCmidisend.csd to generate OSC events for this file

  0dbfs   =  1

  gilisten  OSCinit   47120

  gisin     ftgen     1, 0, 16384, 10, 1
  givel     ftgen     2, 0, 128, -2, 0
  gicc     ftgen     3, 0, 128, -7, 100, 128, 100  ;Default all controllers to 100

;Define scale tuning
  giji_12   ftgen     202, 0, 32, -2, 12, 2, 256, 60, 1, 16/15, 9/8, 6/5, 5/4, 4/3, 7/5, \
                               3/2, 8/5, 5/3, 9/5, 15/8, 2

#define DEST #"/midi"#
; Use controller number 7 for volume
#define VOL #7#

turnon 1000


    instr   1000

  kst     init      0
  kch     init      0
  kd1     init      0
  kd2     init      0

next:

  kk     OSClisten gilisten, $DEST, "iiii", kst, kch, kd1, kd2

if (kk == 0) goto done

printks "kst = %i, kch = %i, kd1 = %i, kd2 = %i\\n", \
         0, kst, kch, kd1, kd2

if (kst == 176) then
;Store controller information in a table
            tablew    kd2, kd1, gicc
endif

if (kst == 144) then
;Process noteon and noteoff messages.
  kkey     =  kd1
  kvel     =  kd2
  kcps     cpstun    kvel, kkey, giji_12
  kamp     =  kvel/127

if (kvel == 0) then
            turnoff2  1001, 4, 1
elseif (kvel > 0) then
            event     "i", 1001, 0, -1, kcps, kamp
endif
endif

        kgoto next  ;Process all events in queue

done:
    endin


    instr 1001   ;Simple instrument

  icps     init      p4
  kvol     table     $VOL, gicc  ;Read MIDI volume from controller table
  kvol     =  kvol/127

  aenv     linsegr    0, .003, p5, 0.03, p5 * 0.5, 0.3, 0
  aosc     oscil     aenv, icps, gisin

            out       aosc * kvol
    endin
```

```
</CsInstruments>
<CsScore>
f 0 3600  ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch
2005
Examples by: David Akbari, Andrés Cabrera and Jonathan Murphy 2007

oscbnk

oscbnk — Mélange la sortie de n'importe quel nombre d'oscillateurs.

Description

Ce générateur unitaire mélange la sortie de n'importe quel nombre d'oscillateurs. La fréquence, la phase et l'amplitude de chaque oscillateur peuvent être modulées par deux LFO (tous les oscillateurs ont un jeu de LFO séparé, avec différentes phase et fréquence) ; de plus, la sortie de chaque oscillateur peut être filtrée au travers d'un égaliseur paramétrique (aussi contrôlé par les LFO). Cet opcode trouve sa plus grande utilité dans des instruments de rendu d'ensemble (cordes, chœur, etc.).

Bien que les LFO fonctionnent au taux-k, les modulations d'amplitude, de phase et de filtrage sont interpolées en interne, et il est ainsi possible (et recommandé dans la plupart des cas) d'utiliser cette unité avec de faibles taux de contrôle (~1000 Hz) sans dégradation audible de la qualité.

La phase et la fréquence initiale de tous les oscillateurs et LFO peuvent être fixées par un générateur intégré de nombres aléatoires sur 31 bit amorçable par une « graine », ou spécifiées manuellement dans une table de fonction (GEN2).

Syntaxe

```
ares oscbnk  kcps, kamd, kfmd, kpmd, iovrlap, iseed, kl1minf, kl1maxf, \
      kl2minf, kl2maxf, ilfomode, keqminf, keqmaxf, keqminl, keqmaxl, \
      keqminq, keqmaxq, ieqmode, kfn [, il1fn] [, il2fn] [, ieqffn]   \
      [, ieqlfn] [, ieqqfn] [, itabl] [, ioutfn]
```

Initialisation

iovrlap -- Nombre d'oscillateurs.

iseed -- Valeur de la graine du générateur de nombres aléatoires (entier positif dans l'intervalle 1 à 2147483646 (2 ^ 31 - 2)). Si *iseed* <= 0 la graine est l'heure courante.

ieqmode -- Mode de l'égaliseur paramétrique

- -1 : désactive l'EQ (plus rapide)
- 0 : crête
- 1 : à plateau low shelf
- 2 : à plateau high shelf
- 3 : crête (filtrage sans interpolation)
- 4 : à plateau low shelf (sans interpolation)
- 5 : à plateau high shelf (sans interpolation)

Les modes sans interpolation sont plus rapides, et dans certains cas (par exemple filtre à plateau high shelf aux fréquences de coupure basses) également plus stables ; cependant, l'interpolation est utile pour éviter le « bruit de fermeture éclair » aux faibles taux de contrôle.

ilfomode -- Type de la modulation par les LFO, somme de :

- 128 : LFO1 module la fréquence
- 64 : LFO1 module l'amplitude
- 32 : LFO1 module la phase
- 16 : LFO1 module l'EQ
- 8 : LFO2 module la fréquence
- 4 : LFO2 module l'amplitude
- 2 : LFO2 module la phase
- 1 : LFO2 module l'EQ

Si un LFO ne module rien, il n'est pas calculé, et le numéro de sa ftable (*il1fn* ou *il2fn*) peut être omis.

il1fn (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO1. La forme d'onde dans cette table doit être normalisée (valeur absolue <= 1), et elle est lue avec une interpolation linéaire.

il2fn (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO2. La forme d'onde dans cette table doit être normalisée (valeur absolue <= 1), et elle est lue avec une interpolation linéaire.

ieqffn, ieqlfn, ieqqfn (facultatif : par défaut 0) -- Tables de lecture pour la fréquence, le niveau et le Q de EQ (facultatif si EQ est désactivé). La position de lecture dans une table est 0 si le signal de modulation est inférieur ou égal à -1, (longueur de table / 2) si le signal de modulation vaut zero, et le point de garde si le signal de modulation est supérieur ou égal à 1. Ces tables doivent être normalisées dans l'intervalle 0 - 1, et ont un point de garde étendu (longueur de table = puissance de deux + 1). Toutes les tables sont lues avec une interpolation linéaire.

itabl (facultatif : par défaut 0) -- Table de fonction stockant les valeurs de phase et de fréquence pour tous les oscillateurs (facultatif). Les valeurs dans cette table sont dans l'ordre suivant (5 pour chaque oscillateur) :

phase de l'oscillateur, phase de lfo1, fréquence de lfo1, phase de lfo2, fréquence de lfo2, ...

Toutes les valeurs sont dans l'intervalle 0 à 1 ; si le nombre spécifié est supérieur à 1, il est ramené cycliquement (phase) ou limité (fréquence) à l'intérieur de l'intervalle permis. Une valeur négative (ou la fin de la table) utilisera la sortie du générateur de nombres aléatoires. La valeur aléatoire est toujours calculée (même si aucun nombre aléatoire n'est utilisé), si bien que le fait de basculer entre une valeur aléatoire et une valeur fixe n'altérera pas les autres valeurs.

ioutfn (facultatif : par défaut 0) -- Table de fonction pour écrire les valeurs de phase et de fréquence (facultatif). Le format est le même que celui de *itabl*. Cette table est utile lors de l'expérimentation avec des nombres aléatoires pour enregistrer les meilleures valeurs.

L'accès aux deux tables facultatives (*itabl* et *ioutfn*) n'a lieu que pendant l'initialisation. Il est utile de savoir cela, car les tables peuvent être réécrites en toute sécurité après l'initialisation de l'opcode, permettant le pré-calcul des paramètres pendant le temps-i et le stockage dans une table temporaire avant l'initialisation de *oscbnk*.

Exécution

ares -- Signal de sortie.

kcps -- Fréquence de l'oscillateur en Hz.

kamd -- Profondeur de la modulation d'amplitude (0 - 1).

(sortie MA) = (entrée MA) * ((1 - (prof MA)) + (prof MA) * (modulateur))

Si *ilfomode* n'est pas réglé pour moduler l'amplitude, alors (sortie MA) = (entrée MA) quelque soit la valeur de *kamd*. Dans ce cas, *kamd* n'aura pas d'effet.

Note : La modulation d'amplitude est appliquée avant l'égaliseur paramétrique.

kfmd -- Profondeur de la MF (en Hz).

kpmd -- Profondeur de la modulation de phase.

kl1minf, kl1maxf -- Fréquence minimale et maximale de LFO1 en Hz.

kl2minf, kl2maxf -- Fréquence minimale et maximale de LFO2 en Hz. (Note : il est permis d'avoir des fréquences nulles ou négatives pour l'oscillateur et les LFO.)

keqminf, keqmaxf -- Fréquence minimale et maximale de l'égaliseur paramétrique en Hz.

keqminl, keqmaxl -- Niveau minimum et maximum de l'égaliseur paramétrique.

keqminq, keqmaxq -- Q minimum et maximum de l'égaliseur paramétrique.

kfn -- Table de la forme d'onde de l'oscillateur. Le numéro de la table peut être changé au taux-k (c'est utile pour choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter les erreurs de repliement). La table est lue avec une interpolation linéaire.

Note

oscbnk utilise le même générateur de nombres aléatoires que *rnd31*. C'est pourquoi il est également recommandé de lire *sa documentation*.

Exemples

Voici un exemple de l'opcode oscbnk. Il utilise le fichier *oscbnk.csd* [examples/oscbnk.csd].

Exemple 329. Exemple de l'opcode oscbnk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscbnk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr =  48000
```

```
kr =  750
ksmps =  64
nchnls =  2

ga01 init 0
ga02 init 0

/* sawtooth wave */
i_ ftgen 1, 0, 16384, 7, 1, 16384, -1
/* FM waveform */
i_ ftgen 3, 0, 4096, 7, 0, 512, 0.25, 512, 1, 512, 0.25, 512, \
                             0, 512, -0.25, 512, -1, 512, -0.25, 512, 0
/* AM waveform */
i_ ftgen 4, 0, 4096, 5, 1, 4096, 0.01
/* FM to EQ */
i_ ftgen 5, 0, 1024, 5, 1, 512, 32, 512, 1
/* sine wave */
i_ ftgen 6, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11,                  \
                            1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
                            1,  1.753, 0.05, 0.85,  5000, 0.8, 0.5, 2, \
                            1, 39.451, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 33.503, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 36.151, 0.05, 0.85,  7000, 0.8, 0.5, 2, \
                            1, 29.633, 0.05, 0.85,  7000, 0.8, 0.5, 2

/* generate bandlimited sawtooth waves */

i0 =  0
loop1:
imaxh =  sr / (2 * 440.0 * exp (log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 =  i0 + 1
        if (i0 < 127.5) igoto loop1

        instr 1

p3 =  p3 + 0.4

; note frequency
kcps =  440.0 * exp (log(2.0) * (p4 - 69) / 12)
; lowpass max. frequency
klpmaxf limit 64 * kcps, 1000.0, 12000.0
; FM depth in Hz
kfmd1 =  0.02 * kcps
; AM frequency
kamfr =  kcps * 0.02
kamfr2 =  kcps * 0.1
; table number
kfnum =  (256 + 69 + 0.5 + 12 * log(kcps / 440.0) / log(2.0))
; amp. envelope
aenv linseg 0, 0.1, 1.0, p3 - 0.5, 1.0, 0.1, 0.5, 0.2, 0, 1.0, 0

/* oscillator / left */

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 200, 0.1, 0.2, 0, 0, 144,       \
                0.0, klpmaxf, 0.0, 0.0, 1.5, 1.5, 2,                  \
                kfnum, 3, 0, 5, 5, 5
a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 201, 0.1, 0.2, kamfr, kamfr2, 148,  \
                0, 0, 0, 0, 0, 0, -1,                                   \
                kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 =  a1 + a2 * 0.12
/* delay */
adel =  0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.0
adel =  adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 =  a01 + a02

ga01 =  ga01 + a0 * aenv * 2500

/* oscillator / right */

; lowpass max. frequency

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 202, 0.1, 0.2, 0, 0, 144,       \
                0.0, klpmaxf, 0.0, 0.0, 1.0, 1.0, 2,                  \
                kfnum, 3, 0, 5, 5, 5
```

```
a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148,  \
                0, 0, 0, 0, 0, 0, -1,                                      \
                kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 =  a1 + a2 * 0.12
/* delay */
adel =  0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.25
adel =  adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 =  a01 + a02

ga02 =  ga02 + a0 * aenv * 2500


        endin

/* output / left */

        instr 81

i1 =  0.000001
aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, -8.0, 4.0, 0.0, 0.3, 7, 4
ga01 =  0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

        outs aLl + aLh, aRl + aRh

        endin

/* output / right */

        instr 82

i1 =  0.000001
aLl, aLh, aRl, aRh spat3di ga02 + i1*i1*i1*i1, 8.0, 4.0, 0.0, 0.3, 7, 4
ga02 =  0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

        outs aLl + aLh, aRl + aRh

        endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60

i 1 0 4 41
i 1 0 4 60
i 1 0 4 65
i 1 0 4 69

i 81 0 5.5
i 82 0 5.5
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Istvan Varga
2001

Nouveau dans la version 4.15

Mis à jour en avril 2002 par Istvan Varga

oscil

oscil — Un oscillateur simple.

Description

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Syntaxe

```
ares oscil xamp, xcps, ifn [, iphs]
```

```
kres oscil kamp, kcps, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

iphs (facultatif, par défaut 0) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

kamp, xamp -- amplitude

kcps, xcps -- fréquence en cycles par seconde.

L'opcode *oscil* génère des signaux de contrôle (ou audio) constitués de la valeur de *kamp* (*xamp*) fois la valeur de la lecture au taux de contrôle (ou au taux audio) d'une table de fonction stockée. La phase interne est simultanément incrémentée selon la valeur en entrée de *kcps* ou de *xcps*.

Exemples

Voici un exemple de l'opcode oscil. Il utilise le fichier *oscil.csd* [examples/oscil.csd].

Exemple 330. Exemple de l'opcode oscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

oscili, *oscil3*

Crédits

Exemple écrit par Kevin Conder.

oscil1

oscil1 — Accède aux valeurs d'une table par échantillonnage incrémentiel.

Description

Accède aux valeurs d'une table par échantillonnage incrémentiel.

Syntaxe

```
kres oscil1 idel, kamp, idur, ifn
```

Initialisation

idel -- délai en secondes avant que l'échantillonnage incrémentiel d'*oscil1* ne commence.

idur -- durée en secondes de l'unique passe d'échantillonnage dans la table d'*oscil1*. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

ifn -- numéro de la table de fonction. *tablei, oscil1i* nécessitent un point de garde.

Exécution

kamp -- facteur d'amplitude.

oscil1 accède aux valeurs en échantillonnant une fois la table de fonction à un taux déterminé par *idur*. Pendant les premières *idel* secondes, le point de lecture reste sur la première position de la table ; ensuite il traverse la table à vitesse constante, atteignant la fin au bout de *idur* secondes ; à partir de ce moment (c-à-d après *idel* + *idur* secondes) il reste sur la dernière position. Chaque valeur lue par échantillonnage est multipliée par le facteur d'amplitude *kamp* avant d'être écrite dans le résultat.

Voir Aussi

table, *tablei*, *table3*, *oscil1i*, *osciln*

oscil1i

oscil1i — Accesses table values by incremental sampling with linear interpolation.

Description

Accesses table values by incremental sampling with linear interpolation.

Syntax

```
kres oscil1i idel, kamp, idur, ifn
```

Initialization

idel -- delay in seconds before *oscil1* incremental sampling begins.

idur -- duration in seconds to sample through the *oscil1* table just once. A zero or negative value will cause all initialization to be skipped.

ifn -- function table number. *oscil1i* requires the extended guard point.

Performance

kamp -- amplitude factor

oscil1i is an interpolating unit in which the fractional part of index is used to interpolate between adjacent table entries. The smoothness gained by interpolation is at some small cost in execution time (see also *oscili*, etc.), but the interpolating and non-interpolating units are otherwise interchangeable.

See Also

table, *tablei*, *table3*, *oscil1*, *osciln*

oscil3

oscil3 — Un oscillateur simple avec interpolation cubique.

Description

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Syntaxe

```
ares oscil3 xamp, xcps, ifn [, iphs]
```

```
kres oscil3 kamp, kcps, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

iphs (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

kamp, xamp -- amplitude

kcps, xcps -- fréquence en cycles par seconde.

oscil3 est expérimental, et identique à *oscili*, sauf qu'il utilise l'interpolation cubique. (Nouveau dans la version 3.50 de Csound.)

Exemples

Voici un exemple de l'opcode oscil3. Il utilise le fichier *oscil3.csd* [examples/oscil3.csd].

Exemple 331. Exemple de l'opcode oscil3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscil3.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```

```
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

; Instrument #2 - the basic oscillator with cubic interpolation.
instr 2
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscil3 kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 32 10 0 1

; Play Instrument #1, the basic oscillator, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the cubic interpolated oscillator, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

oscil, *oscili*

Crédits

Auteur : John ffitch

Exemple écrit par Kevin Conder.

oscili

oscili — Un oscillateur simple avec interpolation linéaire.

Description

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Syntaxe

```
ares oscili xamp, xcps, ifn [, iphs]
```

```
kres oscili kamp, kcps, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

iphs (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

kamp, xamp -- amplitude

kcps, xcps -- fréquence en cycles par seconde.

oscili diffère de *oscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

Exemples

Voici un exemple de l'opcode oscili. Il utilise le fichier *oscili.csd* [examples/oscili.csd].

Exemple 332. Exemple de l'opcode oscili.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscili.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

; Instrument #2 - the basic oscillator with extra interpolation.
instr 2
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscili kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 32 10 0 1

; Play Instrument #1, the basic oscillator, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the interpolated oscillator, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

oscil, *oscil3*

Crédits

Exemple écrit par Kevin Conder.

oscilikt

oscilikt — Un oscillateur avec interpolation linéaire qui permet de changer le numéro de table au taux-k.

Description

oscilikt ressemble beaucoup à *oscili*, mais il permet de changer le numéro de table au taux-k. Il est légèrement plus lent que *oscili* (spécialement avec des taux de contrôle élevés), mais en contrepartie il est plus précis car il utilise un accumulateur de phase sur 31 bit au lieu de celui sur 24 bit utilisé par *oscili*.

Syntaxe

```
ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
```

```
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]
```

Initialisation

iphs (facultatif, par défaut 0) -- phase initiale dans l'intervalle 0 à 1. Les autres valeurs sont ramenées cycliquement dans l'intervalle autorisé.

istor (facultatif, par défaut 0) -- ignorer l'initialisation.

Exécution

kamp, *xamp* -- amplitude.

kcps, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

kfn -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

Exemples

Voici un exemple de l'opcode oscilikt. Il utilise le fichier *oscilikt.csd* [examples/oscilikt.csd].

Exemple 333. Exemple de l'opcode oscilikt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikt.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a uni-polar (0-1) square wave.
  kamp1 init 1
  kcps1 init 2
  itype = 3
  ksquare lfo kamp1, kcps1, itype

  ; Use the square wave to switch between Tables #1 and #2.
  kamp2 init 20000
  kcps2 init 220
  kfn = ksquare + 1

  a1 oscilikt kamp2, kcps2, kfn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine waveform.
f 1 0 4096 10 0 1
; Table #2: a sawtooth wave
f 2 0 3 -2 1 0 -1

; Play Instrument #1 for two seconds.
i 1 0 2


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

osciliktp et *oscilikts*.

Crédits

Auteur : Istvan Varga

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.22

osciliktp

osciliktp — Un oscillateur avec interpolation linéaire qui permet la modulation de phase.

Description

osciliktp permet la modulation de phase (qui est implémentée comme une modulation de fréquence au taux-k, en différenciant la phase en entrée). Le désavantage est qu'il n'y a pas de contrôle d'amplitude, et que la fréquence ne peut varier qu'au taux de contrôle. Cet opcode peut être plus rapide ou plus lent que *oscilikt*, en fonction du taux de contrôle.

Syntaxe

```
ares osciliktp kcps, kfn, kphs [, istor]
```

Initialisation

istor (facultatif, par défaut 0) -- ignorer l'initialisation.

Exécution

ares -- signal de sortie au taux audio.

kcps, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/*2).

kfn -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

kphs -- phase (taux-k), l'intervalle attendu est 0 à 1. La valeur absolue de la différence entre les valeurs courante et précédente de *kphs* doit être inférieure à *ksmps*.

Exemples

Voici un exemple de l'opcode osciliktp Il utilise le fichier *osciliktp.csd* [examples/osciliktp.csd].

Exemple 334. Exemple de l'opcode osciliktp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o osciliktp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: osciliktp example
instr 1
  kphs line 0, p3, 4

  a1x osciliktp 220.5, 1, 0
  a1y osciliktp 220.5, 1, -kphs
  a1 =  a1x - a1y

  out a1 * 14000
endin


</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

oscilikt et *oscilikts*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

oscilikts

oscilikts — Un oscillateur avec interpolation linéaire et statut de synchronisation qui permet de changer le numéro de table au taux-k.

Description

oscilikts est pareil à *oscilikt*. Sauf qu'il a une entrée de synchronisation que l'on peut utiliser pour réinitialiser l'oscillateur à une valeur de phase de taux-k. Il est plus lent que *oscilikt* et que *osciliktp*.

Syntaxe

```
ares oscilikts xamp, xcps, kfn, async, kphs [, istor]
```

Initialisation

istor (facultatif, par défaut 0) -- ignorer l'initialisation.

Exécution

xamp -- amplitude.

kcps, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

kfn -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

async -- n'importe quelle valeur positive réinitialise la valeur de la phase de *oscilikts* à *kphs*. Zero ou des valeurs négatives n'ont aucun effet.

kphs -- fixe la phase, initialement et lorsqu'elle est réinitialisée avec async.

Exemples

Voici un exemple de l'opcode oscilikts. Il utilise le fichier *oscilikts.csd* [examples/oscilikts.csd].

Exemple 335. Exemple de l'opcode oscilikts.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikts.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: oscilikts example.
instr 1
  ; Frequency envelope.
  kfrq expon 400, p3, 1200
  ; Phase.
  kphs line 0.1, p3, 0.9

  ; Sync 1
  atmp1 phasor 100
  ; Sync 2
  atmp2 phasor 150
  async diff 1 - (atmp1 + atmp2)

  a1 oscilikts 14000, kfrq, 1, async, 0
  a2 oscilikts 14000, kfrq, 1, async, -kphs

  out a1 - a2
endin


</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

oscilikt et *osciliktp*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

osciln

osciln — Lit des valeurs dans une table à une fréquence définie par l'utilisateur.

Description

Lit des valeurs dans une table à une fréquence définie par l'utilisateur. On peut également écrire cet opcode comme *oscilx*.

Syntaxe

```
ares osciln kamp, ifrq, ifn, itimes
```

Initialisation

ifrq, itimes -- taux de lecture et nombre de passages à travers la table.

ifn -- numéro de la table de fonction.

Exécution

kamp -- facteur d'amplitude

osciln parcourera plusieurs fois la table stockée en prélevant un échantillon *ifrq* fois par seconde, après quoi il retournera des zéros. Il génère seulement des signaux audio, avec les valeurs de sortie pondérées par *kamp.*

Voir aussi

table, *tablei*, *table3*, *oscil1*, *oscil1i*

oscils

oscils — Un oscillateur sinus simple et rapide.

Description

Oscillateur sinus simple et rapide, qui utilise seulement une multiplication et deux additions pour générer un échantillon en sortie, et qui ne nécessite pas de table de fonction.

Syntaxe

```
ares oscils iamp, icps, iphs [, iflg]
```

Initialisation

iamp -- amplitude en sortie.

icps -- fréquence en Hz (peut être nulle ou négative, cependant la valeur absolue doit être inférieure à *sr/*2).

iphs -- phase initiale entre 0 et 1.

iflg -- sommme des valeurs suivantes :

- *2* : utiliser la double précision même si Csound a été compilé pour utiliser des floats. Ceci améliore la qualité (spécialement dans le cas d'une longue exécution), mais le temps de calcul peut varier du simple au double.
- *1* : ignorer l'initialisation.

Exécution

ares -- sortie audio

Exemples

Voici un exemple de l'opcode oscils. Il utilise le fichier *oscils.csd* [examples/oscils.csd].

Exemple 336. Exemple de l'opcode oscils.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscils.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a fast sine oscillator.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Istvan Varga
Janvier 2002

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.18

oscilx

oscilx — Identique à l'opcode osciln.

Description

Voir l'opcode *osciln*.

out

out — Writes mono audio data to an external device or stream.

Description

Writes mono audio data to an external device or stream.

Syntax

```
out asig
```

Performance

Sends mono audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with *nchnls* statement.

See Also

outh, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

Original in Csound v1

out32

out32 — Writes 32-channel audio data to an external device or stream.

Description

Writes 32-channel audio data to an external device or stream.

Syntax

```
out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \
      asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \
      asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \
      asig27, asig28, asig29, asig30, asig31, asig32
```

Performance

out32 outputs 32 channels of audio.

Credits

outc, *outch*, *outx*, *outz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

outc

outc — Writes audio data with an arbitrary number of channels to an external device or stream.

Description

Writes audio data with an arbitrary number of channels to an external device or stream.

Syntax

```
outc asig1 [, asig2] [...]
```

Performance

outc outputs as many channels as provided. Any channels greater than *nchnls* are ignored. Zeros are added as necessary

Credits

out32, *outch*, *outx*, *outz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

outch

outch — Writes multi-channel audio data, with user-controllable channels, to an external device or stream.

Description

Writes multi-channel audio data, with user-controllable channels, to an external device or stream.

Syntax

```
outch ksig1, asig1 [, ksig2] [, asig2] [...]
```

Performance

outch outputs *asig1* on the channel determined by *ksig1*, *asig2* on the channel determined by *ksig2*, etc.

Credits

out32, *outc*, *outx*, *outz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

outh

outh — Writes 6-channel audio data to an external device or stream.

Description

Writes 6-channel audio data to an external device or stream.

Syntax

```
outh asig1, asig2, asig3, asig4, asig5, asig6
```

Performance

Sends 6-channel audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with *nchnls* statement.

See Also

out, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: John ffitch

Introduced before Version 3

outiat

outiat — Sends MIDI aftertouch messages at i-rate.

Description

Sends MIDI aftertouch messages at i-rate.

Syntax

```
outiat ichn, ivalue, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

ivalue -- floating point value

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

Performance

outiat (i-rate aftertouch output) sends aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outic14, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outic

outic — Sends MIDI controller output at i-rate.

Description

Sends MIDI controller output at i-rate.

Syntax

```
outic ichn, inum, ivalue, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

inum -- controller number (0-127 for example 1 = ModWheel; 2 = BreathControl etc.)

ivalue -- floating point value

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

Performance

outic (i-rate MIDI controller output) sends controller messages to the MIDI OUT device. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outiat, *outic14*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outic14

outic14 — Sends 14-bit MIDI controller output at i-rate.

Description

Sends 14-bit MIDI controller output at i-rate.

Syntax

```
outic14 ichn, imsb, ilsb, ivalue, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

imsb -- most significant byte controller number when using 14-bit parameters (0-127)

ilsb -- least significant byte controller number when using 14-bit parameters (0-127)

ivalue -- floating point value

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 16383 (14-bit))

Performance

outic14 (i-rate MIDI 14-bit controller output) sends a pair of controller messages. This opcode can drive 14-bit parameters on MIDI instruments that recognize them. The first control message contains the most significant byte of *ivalue* argument while the second message contains the less significant byte. *imsb* and *ilsb* are the number of the most and less significant controller.

This opcode can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outiat, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outipat

outipat — Sends polyphonic MIDI aftertouch messages at i-rate.

Description

Sends polyphonic MIDI aftertouch messages at i-rate.

Syntax

```
outipat ichn, inotenum, ivalue, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

inotenum -- MIDI note number (used in polyphonic aftertouch messages)

ivalue -- floating point value

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

Performance

outipat (i-rate polyphonic aftertouch output) sends polyphonic aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outiat, *outic14*, *outic*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outipb

outipb — Sends MIDI pitch-bend messages at i-rate.

Description

Sends MIDI pitch-bend messages at i-rate.

Syntax

```
outipb ichn, ivalue, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

ivalue -- floating point value

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

Performance

outipb (i-rate pitch bend output) sends pitch bend messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outipc

outipc — Sends MIDI program change messages at i-rate

Description

Sends MIDI program change messages at i-rate

Syntax

```
outipc ichn, iprog, imin, imax
```

Initialization

ichn -- MIDI channel number (1-16)

iprog -- program change number in floating point

imin -- minimum floating point value (converted in MIDI integer value 0)

imax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

Performance

outipc (i-rate program change output) sends program change messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkat

outkat — Sends MIDI aftertouch messages at k-rate.

Description

Sends MIDI aftertouch messages at k-rate.

Syntax

```
outkat kchn, kvalue, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

kvalue -- floating point value

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 127)

outkat (k-rate aftertouch output) sends aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkc

outkc — Sends MIDI controller messages at k-rate.

Description

Sends MIDI controller messages at k-rate.

Syntax

```
outkc kchn, knum, kvalue, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

knum -- controller number (0-127 for example 1 = ModWheel; 2 = BreathControl etc.)

kvalue -- floating point value

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

outkc (k-rate MIDI controller output) sends controller messages to MIDI OUT device. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkc14

outkc14 — Sends 14-bit MIDI controller output at k-rate.

Description

Sends 14-bit MIDI controller output at k-rate.

Syntax

```
outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

kmsb -- most significant byte controller number when using 14-bit parameters (0-127)

klsb -- least significant byte controller number when using 14-bit parameters (0-127)

kvalue -- floating point value

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 16383 (14-bit))

outkc14 (k-rate MIDI 14-bit controller output) sends a pair of controller messages. It works only with MIDI instruments which recognize them. These opcodes can drive 14-bit parameters on MIDI instruments that recognize them. The first control message contains the most significant byte of *kvalue* argument while the second message contains the less significant byte. *kmsb* and *klsb* are the number of the most and less significant controller.

It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc*, *outkpat*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkpat

outkpat — Sends polyphonic MIDI aftertouch messages at k-rate.

Description

Sends polyphonic MIDI aftertouch messages at k-rate.

Syntax

```
outkpat kchn, knotenum, kvalue, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

knotenum -- MIDI note number (used in polyphonic aftertouch messages)

kvalue -- floating point value

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

outkpat (k-rate polyphonic aftertouch output) sends polyphonic aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpb*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkpb

outkpb — Sends MIDI pitch-bend messages at k-rate.

Description

Sends MIDI pitch-bend messages at k-rate.

Syntax

```
outkpb kchn, kvalue, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

kvalue -- floating point value

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

outkpb (k-rate pitch-bend output) sends pitch-bend messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpc*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outkpc

outkpc — Sends MIDI program change messages at k-rate.

Description

Sends MIDI program change messages at k-rate.

Syntax

```
outkpc kchn, kprog, kmin, kmax
```

Performance

kchn -- MIDI channel number (1-16)

kprog -- program change number in floating point

kmin -- minimum floating point value (converted in MIDI integer value 0)

kmax -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

outkpc (k-rate program change output) sends program change messages. It works only with MIDI instruments which recognize them. These opcodes can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

Examples

Here is an example of the outkpc opcode. It uses the file *outkpc.csd* [examples/outkpc.csd].

Exemple 337. Example of the outkpc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a program change and a note on Csound's MIDI output port whenever a note is received on channel 1. Be sure to have something connected to Csound's MIDI out port to hear the result.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d         -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

kprogram init 0

instr 1 ;Triggered by MIDI notes on channel 1

  ifund   notnum
  ivel         veloc
  idur = 1

; Sends a MIDI program change message according to
; the triggering note's velocity
outkpc     1 ,ivel ,0 ,127

noteondur  1 ,ifund ,ivel ,idur

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the outkpc opcode. It uses the file *outkpc_flkt.csd* [examples/outkpc_fltk.csd].

Exemple 338. Example of the outkpc opcode using FLTK.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

FLpanel "outkpc",200,100,90,90;start of container
gkpg, gihandle FLcount "Midi-Program change",0,127,1,5,1,152,40,16,23,-1
FLpanelEnd

FLrun

instr 1

ktrig changed gkpg
outkpc     ktrig,gkpg,0,127

endin


</CsInstruments>
<CsScore>
; Run instrument 1 for 60 seconds
i 1 0  60
</CsScore>
</CsoundSynthesizer>
```

See Also

outiat, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

outo

outo — Writes 8-channel audio data to an external device or stream.

Description

Writes 8-channel audio data to an external device or stream.

Syntax

```
outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8
```

Performance

Sends 8-channel audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with *nchnls* statement.

See Also

out, *outh*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: John ffitch

New after 3.30

outq

outq — Writes 4-channel audio data to an external device or stream.

Description

Writes 4-channel audio data to an external device or stream.

Syntax

```
outq asig1, asig2, asig3, asig4
```

Performance

Sends 4-channel audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outq1

outq1 — Writes samples to quad channel 1 of an external device or stream.

Description

Writes samples to quad channel 1 of an external device or stream.

Syntax

```
outq1 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outq2

outq2 — Writes samples to quad channel 2 of an external device or stream.

Description

Writes samples to quad channel 2 of an external device or stream.

Syntax

```
outq2 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outq3

outq3 — Writes samples to quad channel 3 of an external device or stream.

Description

Writes samples to quad channel 3 of an external device or stream.

Syntax

```
outq3 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outq4

outq4 — Writes samples to quad channel 4 of an external device or stream.

Description

Writes samples to quad channel 4 of an external device or stream.

Syntax

```
outq4 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outs*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outrg

outrg — Allow output to a range of adjacent audio channels on the audio input device

Description

outrg outputs audio to a range of adjacent audio channels on the audio output device.

Syntax

```
outrg kstart, aout1 [,aout2, aout3, ..., aoutN]
```

Performance

kstart - the number of the first channel of the output device to be accessed (channel numbers starts with 1, which is the first channel)

aout1, aout2, ... aoutN - the arguments containing the audio to be output to the corresponding output channels.

outrg allows to output a range of adjacent channels to the output device. *kstart* indicates the first channel to be accessed (channel 1 is the first channel). The user must be sure that the number obtained by summing kstart plus the number of accessed channels -1 is <= *nchnls*.

Credits

Author: Gabriel Maldonado

New in version 5.06

outs

outs — Writes stereo audio data to an external device or stream.

Description

Writes stereo audio data to an external device or stream.

Syntax

```
outs asig1, asig2
```

Performance

Sends stereo audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs1*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outs1

outs1 — Writes samples to stereo channel 1 of an external device or stream.

Description

Writes samples to stereo channel 1 of an external device or stream.

Syntax

```
outs1 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs2*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outs2

outs2 — Writes samples to stereo channel 2 of an external device or stream.

Description

Writes samples to stereo channel 2 of an external device or stream.

Syntax

```
outs2 asig
```

Performance

Sends audio samples to an accumulating output buffer (created at the beginning of performance) which serves to collect the output of all active instruments before the sound is written to disk. There can be any number of these output units in an instrument.

The type (mono, stereo, quad, hex, or oct) should agree with *nchnls*. But as of version 3.50, Csound will attempt to change an incorrect opcode to agree with the *nchnls* statement. Opcodes can be chosen to direct sound to any particular channel: *outs1* sends to stereo channel 1, *outq3* to quad channel 3, etc.

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *soundout*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

outvalue

outvalue — Sends a k-rate signal or string to a user-defined channel.

Description

Sends a k-rate signal or string to a user-defined channel.

Syntax

```
outvalue "channel name", kvalue
```

```
outvalue "channel name", "string"
```

Performance

"channel name" -- An integer or string (in double-quotes) representing channel.

kvalue -- The k-rate value that is sent to the channel.

string -- The string or string variable that is sent to the channel.

See Also

invalue

Credits

Author: Matt Ingalls

outx

outx — Writes 16-channel audio data to an external device or stream.

Description

Writes 16-channel audio data to an external device or stream.

Syntax

```
outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \
      asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16
```

Performance

outx outputs 32 channels of audio.

Credits

out32, *outc*, *outch*, *outz*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

outz

outz — Writes multi-channel audio data from a ZAK array to an external device or stream.

Description

Writes multi-channel audio data from a ZAK array to an external device or stream.

Syntax

```
outz ksig1
```

Performance

outz outputs from a ZAK array for *nchnls* of audio.

Credits

out32, *outc*, *outch*, *outx*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.06

p

p — Show the value in a given p-field.

Description

Show the value in a given p-field.

Syntax

```
p(x)
```

This function works at i-rate and k-rate.

Initialization

x -- the number of the p-field.

Performance

The value returned by the *p* function is the value in a p-field.

Examples

Here is an example of the p opcode. It uses the file *p.csd* [examples/p.csd].

Exemple 339. Example of the p opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o p.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Get the value in the fourth p-field, p4.
  i1 = p(4)

  print i1
endin


</CsInstruments>
```

```
<CsScore>

; p4 = value to be printed.
; Play Instrument #1 for one second, p4 = 50.375.
i 1 0 1 50.375
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
instr 1:  i1 = 50.375
```

Credits

Example written by Kevin Conder.

pan

pan — Distribute an audio signal amongst four channels.

Description

Distribute an audio signal amongst four channels with localization control.

Syntax

```
a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]
```

Initialization

ifn -- function table number of a stored pattern describing the amplitude growth in a speaker channel as sound moves towards it from an adjacent speaker. Requires extended guard-point.

imode (optional) -- mode of the *kx, ky* position values. 0 signifies raw index mode, 1 means the inputs are normalized (0 - 1). The default value is 0.

ioffset (optional) -- offset indicator for *kx, ky*. 0 infers the origin to be at channel 3 (left rear); 1 requests an axis shift to the quadraphonic center. The default value is 0.

Performance

pan takes an input signal *asig* and distributes it amongst four outputs (essentially quad speakers) according to the controls *kx* and *ky*. For normalized input (mode=1) and no offset, the four output locations are in order: left-front at (0,1), right-front at (1,1), left-rear at the origin (0,0), and right-rear at (1,0). In the notation (*kx*, *ky)*, the coordinates *kx* and *ky*, each ranging 0 - 1, thus control the 'rightness' and 'forwardness' of a sound location.

Movement between speakers is by amplitude variation, controlled by the stored function table *ifn*. As *kx* goes from 0 to 1, the strength of the right-hand signals will grow from the left-most table value to the right-most, while that of the left-hand signals will progress from the right-most table value to the left-most. For a simple linear pan, the table might contain the linear function 0 - 1. A more correct pan that maintains constant power would be obtained by storing the first quadrant of a sinusoid. Since pan will scale and truncate *kx* and *ky* in simple table lookup, a medium-large table (say 8193) should be used.

kx, ky values are not restricted to 0 - 1. A circular motion passing through all four speakers (inscribed) would have a diameter of root 2, and might be defined by a circle of radius R = root 1/2 with center at (.5,.5). *kx, ky* would then come from Rcos(angle), Rsin(angle), with an implicit origin at (.5,.5) (i.e. *ioffset* = 1). Unscaled raw values operate similarly. Sounds can thus be located anywhere in the polar or Cartesian plane; points lying outside the speaker square are projected correctly onto the square's perimeter as for a listener at the center.

Examples

```
instr     1
  k1          phasor    1/p3                          ; fraction of circle
  k2          tablei    k1, 1, 1                      ; sin of angle (sinusoid in f1)
  k3          tablei    k1, 1, 1, .25, 1              ; cos of angle (sin offset 1/4 circle)
```

```
  a1            oscili    10000,440, 1             ; audio signal..
  a1,a2,a3,a4   pan       a1, k2/2, k3/2, 2, 1, 1  ; sent in a circle (f2=1st quad sin)

                outq a1, a2, a3, a4
endin
```

pan2

pan2 — Distribute an audio signal across two channels.

Description

Distribute an audio signal across two channels with a choice of methods.

Syntax

```
a1, a2 pan2 asig, xp [, imode]
```

Initialization

imode (optional) -- mode of the stereo positioning algorithm. 0 signifies equal power (harmonic) panning, 1 means the square root method, and 2 means simple linear. The default value is 0.

Performance

pan2 takes an input signal *asig* and distributes it across two outputs (essentially stereo speakers) according to the control *xp* which can be k- or a-rate. A zero value for *xp* indicates hard left, and a 1 is hard right.

Examples

```
instr     1
  kline  line    0, p3, 1       ; straight line
  ain    oscili  10000,440, 1   ; audio signal..
  a1,a2  pan2    ain, kline     ; sent across image

         outs   a1, a2
endin
```

Credits

Author: John ffitch
University of Bath, Codemist Ltd.
Bath, UK
September 2007

New in version 5.07

pareq

pareq — Implementation of Zoelzer's parametric equalizer filters.

Description

Implementation of Zoelzer's parametric equalizer filters, with some modifications by the author.

The formula for the low shelf filter is:

```
omega = 2*pi*f/sr
K     = tan(omega/2)

b0    = 1 + sqrt(2*V)*K + V*K^2
b1    = 2*(V*K^2 - 1)
b2    = 1 - sqrt(2*V)*K + V*K^2

a0    = 1 + K/Q + K^2
a1    = 2*(K^2 - 1)
a2    = 1 - K/Q + K^2
```

The formula for the high shelf filter is:

```
omega = 2*pi*f/sr
K     = tan((pi-omega)/2)

b0    = 1 + sqrt(2*V)*K + V*K^2
b1    = -2*(V*K^2 - 1)
b1    = 1 - sqrt(2*V)*K + V*K^2

a0    = 1 + K/Q + K^2
a1    = -2*(K^2 - 1)
a2    = 1 - K/Q + K^2
```

The formula for the peaking filter is:

```
omega = 2*pi*f/sr
K     = tan(omega/2)

b0 =  1 + V*K/2 + K^2
b1 =  2*(K^2 - 1)
b2 =  1 - V*K/2 + K^2

a0 =  1 + K/Q + K^2
a1 =  2*(K^2 - 1)
a2 =  1 - K/Q + K^2
```

Syntax

```
ares pareq asig, kc, kv, kq [, imode] [, iskip]
```

Initialization

imode (optional, default: 0) -- operating mode

- 0 = Peaking
- 1 = Low Shelving
- 2 = High Shelving

iskip (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

Performance

kc -- center frequency in peaking mode, corner frequency in shelving mode.

kv -- amount of boost or cut. A value less than 1 is a cut. A value greater than 1 is a boost. A value of 1 is a flat response.

kq -- Q of the filter (sqrt(.5) is no resonance)

asig -- the incoming signal

Examples

Here is an example of the pareq opcode. It uses the file *pareq.csd* [examples/pareq.csd].

Exemple 340. Example of the pareq opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pareq.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr  = 4410
ksmps = 10
nchnls = 2

instr 15
  ifc     =       p4                        ; Center / Shelf
  kq      =       p5                        ; Quality factor sqrt(.5) is no resonance
  kv      =       ampdb(p6)                 ; Volume Boost/Cut
  imode   =       p7                        ; Mode 0=Peaking EQ, 1=Low Shelf, 2=High Shelf
```

```
  kfc     linseg  ifc*2, p3, ifc/2
  asig    rand    5000                      ; Random number source for testing
  aout    pareq   asig, kfc, kv, kq, imode ; Parmetric equalization
          outs    aout, aout                ; Output the results
endin

</CsInstruments>
<CsScore>

; SCORE:
  ;   Sta  Dur  Fcenter  Q         Boost/Cut(dB)  Mode
  i15 0    1    10000    .2          12             1
  i15 +    .    5000     .2          12             1
  i15 .    .    1000     .707       -12             2
  i15 .    .    5000     .1         -12             0
  e

</CsScore>
</CsoundSynthesizer>
```

Credits

Hans Mikelson
December 1998

New in Csound version 3.50

partials

partials — Partial track spectral analysis.

Description

The partials opcode takes two input PV streaming signals containg AMP_FREQ and AMP_PHASE signals (as generated for instance by pvsifd or in the first case, by pvsanal) and performs partial track analysis, as described in Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. It generates a TRACKS PV streaming signal, containing amplitude, frequency, phase and track ID for each output track. This type of signal will contain a variable number of output tracks, up to the total number of analysis bins contained in the inputs (fftsize/2 + 1 bins). The second input (AMP_PHASE) is optional, as it can take the same signal as the first input. In this case, however, all phase information will be NULL and resynthesis using phase information cannot be performed.

Syntax

```
ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks
```

Performance

ftrks -- output pv stream in TRACKS format

ffr -- input pv stream in AMP_FREQ format

fphs -- input pv stream in AMP_PHASE format

kthresh -- analysis threshold. Tracks below ktresh*max_magnitude will be discarded (1 > ktresh >= 0).

kminpoints -- minimum number of time points for a detected peak to make a track (1 is the minimum). Since this opcode works with streaming signals, larger numbers will increase the delay between input and output, as we have to wait for the required minimum number of points.

kmaxgap -- maximum gap between time-points for track continuation (> 0). Tracks that have no continuation after kmaxgap will be discarded.

imaxtracks -- maximum number of analysis tracks (number of bins >= imaxtracks)

Examples

Exemple 341. Example

```
ain inch 1                    ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
      aout  resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis with pitch shifting.

Credits

Author: Victor Lazzarini;
June 2005

New plugin in version 5

November 2004.

partikkel

partikkel — Synthétiseur granulaire avec un contrôle "par grain" grâce à ses nombreux paramètres. Il a une entrée sync pour synchroniser son horloge interne de distribution des grains avec une horloge externe.

Description

partikkel a été conçu après la lecture du livre de Curtis Road "Microsound", et le but était de créer un opcode capable de réaliser toutes les variétés temporelles de synthèse granulaire décrites dans ce livre. L'idée étant que la plupart des techniques ne diffèrent que par les valeurs des paramètres, et que si l'on a un opcode unique qui peut produire toutes les variétés de synthèse granulaire, l'interpolation entre ces techniques devient possible. La synthèse granulaire est parfois appelée synthèse par particules et il m'a semblé approprié de nommer l'opcode *partikkel* afin de le distinguer des autres opcodes granulaires.

Certains des paramètres d'entrée de *partikkel* sont des numéros de table, pointant sur des tables dans lesquelles sont mémorisées des valeurs pour les changements de paramètre "par grain". *partikkel* peut utiliser une période d'une forme d'onde ou des formes d'onde complexes (par exemple un son échantillonné) comme source de forme d'onde pour les grains. Chaque grain est constitué du mélange de 4 formes d'onde source. On peut accorder séparément la fréquence de base de chacune des 4 formes d'onde source. La modulation de fréquence à l'intérieur de chaque grain est activée via une entrée audio auxiliaire (*awavfm*). La synthèse par trainlet (un trainlet est un bref train d'impulsions) est possible, et les trainlets peuvent être mélangés avec des grains basés sur des tables d'onde. On peut utiliser jusqu'à 8 sorties audio séparées.

Syntaxe

```
a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
              kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
              ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmasks, \
              kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
              ifmamptab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
              ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, \
              kwaveform4, iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
              asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
              [, iopcode_id]
```

Initialisation

idisttab -- numéro d'une table de fonction, distribution des déplacements aléatoires du grain dans le temps. Les valeurs de la table sont interprétées comme la "quantité de déplacement" pondérée par 1/(rythme des grains). Cela signifie qu'une valeur de 0,5 dans la table déplacera un grain de la moitié de la période du rythme des grains. Les valeurs de la table sont lues aléatoirement, et pondérées par *kdistribution*. Pour obtenir des résultats stochastiques réalistes, il vaut mieux ne pas utiliser une taille de table trop petite, car cela limite le nombre des valeurs de déplacement possibles. On peut l'exploiter à d'autres fins, par exemple utiliser des valeurs de déplacement quantifiées pour travailler avec des décalages contrôlés à partir de la période du rythme des grains. Si *kdistribution* est négatif, les valeurs de la table seront lues séquentiellement. On peut sélectionner une table par défaut au moyen du numéro de table -1, pour lequel *idisttab* fournit une distribution nulle (pas de déplacement).

ienv_attack -- numéro d'une table de fonction, forme de l'attaque du grain. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv_attack* fournit une fenêtre rectangulaire (pas d'enveloppe).

ienv_decay -- numéro d'une table de fonction, forme de la chute du grain. Il faut un point de garde

d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv_decay* fournit une fenêtre rectangulaire (pas d'enveloppe).

ienv2tab -- numéro d'une table de fonction, enveloppe additionnelle appliquée au grain après les enveloppes d'attaque et de chute. On peut l'utiliser par exemple pour la synthèse par formant fof. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv2tab* fournit une fenêtre rectangulaire (pas d'enveloppe).

icosine -- numéro d'une table de fonction, devant contenir un cosinus, utilisée pour les trainlets. La table doit avoir une taille d'au moins 2048 pour obtenir des trainlets de bonne qualité.

igainmasks -- numéro d'une table de fonction, gain par grain. La suite des valeurs dans la table a la signification suivante : la valeur d'indice 0 est le point de début d'une boucle de lecture des valeurs, la valeur d'indice 1 étant le point de fin de cette boucle. Les entrées aux autres indices contiennent les valeurs de gain (normalement dans l'intervalle 0 - 1, mais d'autres valeurs sont permises, les valeurs négatives inversant la phase de la forme d'onde du grain) pour une suite de grains ; ces valeurs sont lues au rythme des grains, ce qui permet une correspondance exacte de "gain par grain". Les points du début et de la fin de la boucle sont basés sur zéro avec une origine à l'indice 2, par exemple une valeur de début de boucle de 0 et une valeur de fin de boucle de 3 provoqueront la lecture des valeurs d'indice 2, 3, 4, 5 dans une boucle évoluant au rythme des grains. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *igainmasks* désactive le masquage du gain (tous les grains reçoivent un masque de gain égal à 1).

ichannelmasks -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. L'intervalle des valeurs va de 0 à N, où N est le nombre de canaux de sortie moins 1. Une valeur de zéro enverra le grain sur la sortie audio 1 de l'opcode. On peut utiliser des valeurs non entières, par exemple 3,5 répartira le grain également entre les sorties 4 et 5. L'utilisateur doit éviter les dépassements de niveau, aucun test n'étant effectué. L'opcode plantera si des valeurs dépassent le niveau maximal. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ichannelmasks* désactive le masquage des canaux (tous les grains reçoivent un masque de canal de 0 et sont envoyés sur la sortie audio 1 de *partikkel*).

iwavfreqstarttab -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Multiplicateur de la fréquence de départ de chaque grain. La hauteur glissera de la fréquence de départ jusqu'à la fréquence de fin suivant une droite ou une courbe fixée par *ksweepshape*. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqstarttab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de départ.

iwavfreqendtab -- numéro d'une table de fonction, voir *iwavfreqstarttab*. Multiplicateur de la fréquence de fin de chaque grain. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqendtab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de fin.

ifmamptab -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Indice de modulation de fréquence par grain. Le signal *awavfm* sera multiplié par les valeurs lues dans cette table. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ifmamptab* fournit 1 comme indice de modulation, activant la modulation de fréquence pour tous les grains.

iwaveamptab -- numéro d'une table de fonction, les indices sont parcourus de la même manière que pour *igainmasks*. La valeur d'indice 0 sert de point de début de boucle et la valeur d'indice 1 de point de fin. Les autres indices sont lus par groupes de 5, dans lesquels chaque valeur représente une valeur de gain pour chacune des 4 formes d'onde source, et la cinquième valeur représente l'amplitude de trainlet. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwaveamptab* fournit un mélange égal des 4 formes d'onde source (chacune avec une amplitude de 0,5) et une amplitude de trainlet nulle.

Le calcul des trainlets étant très gourmand en ressources CPU, on peut éviter la plupart des calculs de

trainlet en fixant *ktrainamp* à zéro. Les trainlets sont normalisés au niveau de crête (*ktrainamp*), en compensation des variations d'amplitude causées par les variations de *kpartials* et de *kchroma*.

imax_grains -- nombre maximum de grains par k-periode. Une grande valeur ne devrait pas affecter l'exécution, le dépassement de cette valeur conduira à l'effacement des grains les "plus anciens".

iopcode_id -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*, laquelle fournira en sortie des impulsions de déclenchement synchronisées pour le distributeur de grains de *partikkel*. La valeur par défaut est zéro, ce qui signifie aucune connexion à une instance de *partikkelsync*.

Exécution

xgrainfreq -- nombre de grains par seconde. On peut spécifier une valeur nulle, ce qui déléguera la distribution des grains à l'entrée de synchronisation.

async -- entrée de synchronisation. Les valeurs entrées sont ajoutées à la phase de l'horloge interne du distributeur de grains, ce qui permet une synchronisation de tempo avec une horloge externe. Comme c'est un signal de taux-a, les entrées sont généralement des impulsions de longueur 1/*sr*. A l'aide de telles impulsions on peut "faire bouger" la phase interne en avant ou en arrière, ce qui permet une synchronisation plus ou moins forte. Des valeurs d'entrée négatives décrémentent la phase interne, tandis que des valeurs positives dans l'intervalle de 0 à 1 incrémentent la phase interne. Une valeur d'entrée de 1 forcera toujours *partikkel* à générer un grain. Si la valeur reste à 1, l'horloge interne du distributeur de grain marquera une pause mais tous les grains en cours d'exécution continueront jusqu'à leur terme.

kdistribution -- distribution périodique ou stochastique des grains, 0 = périodique. L'intervalle usuel va de 0 à 1, mais on peut utiliser des valeurs plus grandes pour obtenir l'effet classique de distribution stochastique des grains. Si *kdistribution* est négatif, le résultat est un déplacement déterministe comme celui décrit par *idisttab*.

kenv2amt -- dosage de l'enveloppe secondaire dans l'enveloppe de chaque grain. L'intervalle va de 0 à 1, où 0 signifie pas d'enveloppe secondaire (fenêtre rectangulaire), 0,5 provoquera une interpolation entre une fenêtre rectangulaire et la forme fixée par *ienv2tab*.

ksustain_amount -- durée d'entretien exprimée comme une fraction de la durée du grain. C-à-d la proportion entre le temps d'enveloppe (attaque + chute) et le temps d'entretien. Le niveau d'entretien est celui de la dernière valeur de la ftable *ienv_attack*.

ka_d_ratio -- proportion entre le temps d'attaque et le temps de chute. Par exemple, avec *ksustain_amount* à 0,5 et *ka_d_ratio* à 0,5, l'eveloppe d'attaque de chaque grain prendra 25% de la durée du grain, l'amplitude maximale (entretien) sera tenue pendant 50% de la durée du grain, et l'enveloppe de chute prendra les 25% restants de la durée du grain.

kduration -- durée du grain en millisecondes.

kamp -- facteur de pondération de l'amplitude en sortie de l'opcode. Multiplié par l'amplitude de chaque grain lue à partir de *igainmasks*.

kwavfreq -- facteur de transposition. Multiplié par les valeurs de transposition de départ et de fin lues à partir de *iwavfreqstarttab* et de *iwavfreqendtab*.

ksweepshape -- forme de la progression de la transposition, contrôle la courbure de la progression de la transposition. Dans l'intervalle de 0 à 1. Avec les valeurs faibles, la transposition sera maintenue plus longtemps près de la valeur de départ puis ira rapidement vers la valeur de fin, tandis qu'avec les valeurs fortes la transposition ira tout de suite rapidement vers la valeur de fin. Une valeur de 0,5 donnera une progression linéaire. La valeur 0 supprimera la progression et ne gardera que la fréquence de départ, tandis que la valeur 1 supprimera la progression et ne gardera que la fréquence de fin. Le générateur de la progression peut être légèrement imprécis lorsqu'il atteint la fréquence finale si l'on utilise une courbe

raide avec des grains très longs.

awavfm -- entrée audio pour la modulation de fréquence du grain.

kfmenv -- numéro d'une table de fonction, enveloppe du signal modulateur de la modulation de fréquence provoquant un changement de l'indice de modulation sur toute la durée du grain.

ktraincps -- fréquence fondamentale des trainlets.

knumpartials -- nombre de partiels dans les trainlets.

kchroma -- couleur spectrale des trainlets. Une valeur de 1 donne une amplitude égale à chaque partiel, des valeurs plus grandes réduiront l'amplitude des partiels inférieurs tout en renforçant l'amplitude des partiels supérieurs.

krandommask -- masquage aléatoire (escamotage) de grains individuels. Dans l'intervalle de 0 à 1, où la valeur 0 signifie pas de masquage (tous les grains sont joués), et la valeur 1 escamote tous les grains.

kwaveform1 -- numéro de la table pour la forme d'onde source 1.

kwaveform2 -- numéro de la table pour la forme d'onde source 2.

kwaveform3 -- numéro de la table pour la forme d'onde source 3.

kwaveform4 -- numéro de la table pour la forme d'onde source 4.

asamplepos1 -- position de départ pour la lecture de la forme d'onde source 1.

asamplepos2 -- position de départ pour la lecture de la forme d'onde source 2.

asamplepos3 -- position de départ pour la lecture de la forme d'onde source 3.

asamplepos4 -- position de départ pour la lecture de la forme d'onde source 4.

kwavekey1 -- hauteur originale de la forme d'onde source 1. On peut l'utiliser pour transposer chaque forme d'onde source indépendamment.

kwavekey2 -- comme *kwavekey1*, mais pour la forme d'onde source 2.

kwavekey3 -- comme *kwavekey1*, mais pour la forme d'onde source 3.

kwavekey4 -- comme *kwavekey1*, mais pour la forme d'onde source 4.

Exemples

Voici un exemple de l'opcode partikkel. Il utilise le fichier *PartikkelExample1.csd* [examples/PartikkelExample1.csd].

Exemple 342. Exemple de l'opcode partikkel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac           ;;;RT audio
; For Non-realtime ouput leave only the line below:
```

```
; -o partikkel.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Joachim Heintz and Oeyvind Brandtsegg 2008

giCosine ftgen 0, 0, 8193, 9, 1, 1, 90          ; cosine
giDisttab ftgen 0, 0, 32768, 7, 0, 32768, 1 ; for kdistribution
giFile         ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 0 ; soundfile for source waveform
giWin          ftgen 0, 0, 4096, 20, 9, 1          ; grain envelope
giPan          ftgen 0, 0, 32768, -21, 1          ; for panning (random values between 0 and 1)


; **************************************************
; partikkel example, processing of soundfile
; uses the file "fox.wav"
; **************************************************
                instr 1

/*score parameters*/
ispeed                 = p4         ; 1 = original speed
igrainrate          = p5         ; grain rate
igrainsize          = p6         ; grain size in ms
icent                  = p7         ; transposition in cent
iposrand          = p8         ; time position randomness (offset) of the pointer in ms
icentrand          = p9         ; transposition randomness in cents
ipan                  = p10         ; panning narrow (0) to wide (1)
idist                  = p11         ; grain distribution (0=periodic, 1=scattered)

/*get length of source wave file, needed for both transposition and time pointer*/
ifilen                 tableng giFile
ifildur                 = ifilen / sr

/*sync input (disabled)*/
async                  = 0

/*grain envelope*/
kenv2amt          = 1         ; use only secondary envelope
ienv2tab                = giWin         ; grain (secondary) envelope
ienv_attack          = -1                 ; default attack envelope (flat)
ienv_decay          = -1                 ; default decay envelope (flat)
ksustain_amount          = 0.5         ; no meaning in this case (use only secondary envelope, ienv2tab,
ka_d_ratio          = 0.5                 ; no meaning in this case (use only secondary envelope, ienv2t

/*amplitude*/
kamp                  = 0.4*0dbfs ; grain amplitude
igainmasks          = -1         ; (default) no gain masking

/*transposition*/
kcentrand          rand icentrand ; random transposition
iorig                  = 1 / ifildur ; original pitch
kwavfreq          = iorig * cent(icent + kcentrand)

/*other pitch related (disabled)*/
ksweepshape          = 0         ; no frequency sweep
iwavfreqstarttab          = -1         ; default frequency sweep start
iwavfreqendtab          = -1         ; default frequency sweep end
awavfm                 = 0         ; no FM input
ifmamptab          = -1         ; default FM scaling (=1)
kfmenv                 = -1         ; default FM envelope (flat)

/*trainlet related (disabled)*/
icosine                 = giCosine ; cosine ftable
kTrainCps          = igrainrate ; set trainlet cps equal to grain rate for single-cycle trainlet in each
knumpartials          = 1         ; number of partials in trainlet
kchroma                 = 1         ; balance of partials in trainlet

/*panning, using channel masks*/
imid                  = .5; center
ileftmost          = imid - ipan/2
irightmost          = imid + ipan/2
giPanthis          ftgen 0, 0, 32768, -24, giPan, ileftmost, irightmost ; rescales giPan according to ip
                        tableiw         0, 0, giPanthis                           ; change index 0 ...
                        tableiw         32766, 1, giPanthis                   ; ... and 1 for ichannelmas
ichannelmasks          = giPanthis         ; ftable for panning

/*random gain masking (disabled)*/
```

```
krandommask         = 0

/*source waveforms*/
kwaveform1         = giFile ; source waveform
kwaveform2         = giFile ; all 4 sources are the same
kwaveform3         = giFile
kwaveform4         = giFile
iwaveamptab         = -1         ; (default) equal mix of source waveforms and no amplitude for trainle

/*time pointer*/
afilposphas         phasor ispeed / ifildur
/*generate random deviation of the time pointer*/
iposrandsec         = iposrand / 1000 ; ms -> sec
iposrand         = iposrandsec / ifildur ; phase values (0-1)
krndpos              linrand  iposrand ; random offset in phase values
/*add random deviation to the time pointer*/
asamplepos1         = afilposphas + krndpos; resulting phase values (0-1)
asamplepos2         = asamplepos1
asamplepos3         = asamplepos1
asamplepos4         = asamplepos1

/*original key for each source waveform*/
kwavekey1         = 1
kwavekey2         = kwavekey1
kwavekey3         = kwavekey1
kwavekey4         = kwavekey1

/* maximum number of grains per k-period*/
imax_grains         = 100

aL, aR         partikkel igrainrate, idist, giDisttab, async, kenv2amt, ienv2tab, \
                ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, igrainsize, kamp, igainmasks, \
                kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
                ifmamptab, kfmenv, icosine, kTrainCps, knumpartials, \
                kchroma, ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
                iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
                kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains

                outs                 aL, aR

endin


</CsInstruments>
<CsScore>
;i1 st dur speed grate gsize cent posrnd cntrnd pan dist
i1 0 2.757 1 200 15 0 0 0 0 0
s
i1 0 2.757 1 200 15 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0.4
s
i1 0 2.757 1 200 15 0 400 0 0 1
s
i1 0 5.514 .5 200 20 0 0 600 .5 1
s
i1 0 11.028 .25 200 15 0 1000 400 1 1


</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode partikkel. Il utilise le fichier *partikkel_softsync.csd* [examples/partikkel_softsync.csd].

Exemple 343. Exemple avec une synchronisation légère de deux générateurs partikkel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out
-odac           ;;;RT audio
; For Non-realtime ouput leave only the line below:
; -o partikkel_softsync.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Oeyvind Brandtsegg 2007, revised 2008

giSine         ftgen 0, 0, 65537, 10, 1
giCosine ftgen 0, 0, 8193, 9, 1, 1, 90
giSigmoRise ftgen        0, 0, 8193, 19, 0.5, 1, 270, 1                ; rising sigmoid
giSigmoFall ftgen        0, 0, 8193, 19, 0.5, 1, 90, 1                ; falling sigmoid

; ***************************************************
; example of soft synchronization of two partikkel instances
; ***************************************************
        instr 1


/*score parameters*/
igrainrate = p4         ; grain rate
igrainsize = p5         ; grain size in ms
igrainFreq = p6         ; fundamental frequency of source waveform
iosc2Dev = p7         ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8         ; max soft sync amount (increasing to this value during length of note)

/*overall envelope*/
iattack         = 0.001
idecay         = 0.2
isustain = 0.7
irelease = 0.2
amp         linsegr 0, iattack, 1, idecay, isustain, 1, isustain, irelease, 0

kgrainfreq = igrainrate         ; grains per second
kdistribution = 0                 ; periodic grain distribution
idisttab = -1                 ; (default) flat distribution used
                                          ; for grain distribution
async         = 0                 ; no sync input
kenv2amt = 0                 ; no secondary enveloping
ienv2tab = -1                 ; default secondary envelope (flat)
ienv_attack = giSigmoRise         ; default attack envelope (flat)
ienv_decay = giSigmoFall         ; default decay envelope (flat)
ksustain_amount = 0.3                 ; time (in fraction of grain dur) at
                                          ; sustain level for each grain
ka_d_ratio = 0.2                         ; balance between attack and decay time
kduration = igrainsize         ; set grain duration in ms
kamp         = 0.2*0dbfs                 ; amp
igainmasks = -1                 ; (default) no gain masking
kwavfreq = igrainFreq         ; fundamental frequency of source waveform
ksweepshape = 0                 ; shape of frequency sweep (0=no sweep)
iwavfreqstarttab = -1                 ; default frequency sweep start
                                          ; (value in table = 1, which give
                                          ; no frequency modification)
iwavfreqendtab = -1                 ; default frequency sweep end
                                          ; (value in table = 1, which give
                                          ; no frequency modification)
awavfm         = 0                 ; no FM input
ifmamptab = -1                 ; default FM scaling (=1)
kfmenv         = -1                 ; default FM envelope (flat)
icosine         = giCosine         ; cosine ftable
kTrainCps = kgrainfreq         ; set trainlet cps equal to grain
                                          ; rate for single-cycle trainlet in
                                          ; each grain
knumpartials = 3                 ; number of partials in trainlet
kchroma         = 1                 ; balance of partials in trainlet
ichannelmasks = -1                 ; (default) no channel masking,
                                          ; all grains to output 1
krandommask = 0                 ; no random grain masking
kwaveform1 = giSine         ; source waveforms
kwaveform2 = giSine         ;
kwaveform3 = giSine         ;
kwaveform4 = giSine         ;
iwaveamptab = -1                 ; mix of 4 source waveforms and
                                          ; trainlets (set to default)
asamplepos1 = 0                 ; phase offset for reading source waveform
asamplepos2 = 0                 ;
```

```
asamplepos3 = 0                  ;
asamplepos4 = 0                  ;
kwavekey1 = 1                   ; original key for source waveform
kwavekey2 = 1                   ;
kwavekey3 = 1                   ;
kwavekey4 = 1                   ;
imax_grains = 100                   ; max grains per k period
iopcode_id = 1                   ; id of opcode, linking partikkel
                                          ; to partikkelsync

a1  partikkel kgrainfreq, kdistribution, idisttab, async, kenv2amt, \
       ienv2tab,ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
       kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
       iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamptab, kfmenv, \
       icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
       krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
       iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
       kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains, iopcode_id

async1         partikkelsync iopcode_id   ; clock pulse output of the
                                             ; partikkel instance above
ksyncGravity         line 0, p3, iMaxSync      ; strength of synchronization
aphase2          init 0
asyncPolarity limit (int(aphase2*2)*2)-1, -1, 1
; use the phase of partikkelsync instance 2 to find sync
; polarity for partikkel instance 2.
; If the phase of instance 2 is less than 0.5, we want to
; nudge it down when synchronizing,
; and if the phase is > 0.5 we want to nudge it upwards.
async1         = async1*ksyncGravity*asyncPolarity  ; prepare sync signal
                                                  ; with polarity and strength

kgrainfreq2 = igrainrate * iosc2Dev         ; grains per second for second partikkel instance
iopcode_id2 = 2
a2 partikkel kgrainfreq2, kdistribution, idisttab, async1, kenv2amt, \
       ienv2tab, ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
       kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
       iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamptab, kfmenv, \
       icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
       krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
       iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
       asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, \
       imax_grains, iopcode_id2

async2, aphase2 partikkelsync iopcode_id2
; clock pulse and phase
; output of the partikkel instance above,
; we will only use the phase

outs a1*amp, a2*amp

endin

</CsInstruments>
<CsScore>

/*score parameters
igrainrate = p4         ; grain rate
igrainsize = p5         ; grain size in ms
igrainFreq = p6         ; frequency of source wave within grain
iosc2Dev = p7         ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8         ; max soft sync amount (increasing to this value during length of note)
*/
;          GrRate GrSize GrFund Osc2Dev MaxSync

i1 0          10 2 20 880 1.3 0.3
s
i1 0          10 5 20 440 0.8 0.3
s
i1 0          6 55 15 660 1.8 0.45
s
i1 0          6 110 10 440 0.6 0.6
s
i1 0          6 220 3 660 2.6 0.45
s
i1 0          6 220 3 660 2.1 0.45
s
i1 0          6 440 3 660 0.8 0.22
s

e
```

```
e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

fof, *fof2*, *fog*, *grain*, *grain2*, *grain3*, *granule*, *sndwarp*, *sndwarpst*, *syncgrain*, *syncloop*, *partikkelsync*

Crédits

Auteur : Thom Johansen
Auteur : Torgeir Strand Henriksen
Auteur : Oeyvind Brandtsegg
Avril 2007

Exemples écrits par Joachim Heintz et Oeyvind Brandtsegg.

Nouveau dans la version 5.06

partikkelsync

partikkelsync — Produit l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel* pour synchroniser plusieurs instances de l'opcode *partikkel* à la même source d'horloge.

Description

partikkelsync est un opcode dont la tâche est de produire l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel*. On peut utiliser la sortie de *partikkelsync* pour synchroniser d'autres instances de l'opcode *partikkel* à la même horloge.

Syntaxe

```
async [,aphase] partikkelsync iopcode_id
```

Initialisation

iopcode_id -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*.

Exécution

async -- signal d'impulsion de déclenchement. Envoie des impulsions de déclenchement synchronisées à l'horloge du distributeur de grain d'un opcode *partikkel*. Une impulsion de déclenchement est générée pour le démarrage de chaque grain dans l'opcode *partikkel* ayant le même *opcode_id*. Dans une utilisation normale, on enverra ce signal à l'entrée *async* d'un autre opcode *partikkel* pour synchroniser plusieurs instances de *partikkel*.

aphase -- phase de l'horloge. Sort un signal de phase linéaire. On peut l'utiliser par exemple pour une synchronisation légère, ou simplement comme un générateur de phase à la *phasor*.

Voir Aussi

partikkel

Crédits

Auteur : Thom Johansen
Auteur : Torgeir Strand Henriksen
Auteur : Oeyvind Brandtsegg
Avril 2007

Nouveau dans la version 5.06

pcauchy

pcauchy — Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares pcauchy kalpha
```

```
ires pcauchy kalpha
```

```
kres pcauchy kalpha
```

Exécution

pcauchy kalpha -- contrôle l'étalement à partir de zéro (grand *kalpha* = grand étalement). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode pcauchy. Il utilise le fichier *pcauchy.csd* [examples/pcauchy.csd].

Exemple 344. Exemple de l'opcode pcauchy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pcauchy.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; kalpha = 1

  i1 pcauchy 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.012
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *poisson*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

pchbend

pchbend — Donne la valeur actuelle du pitch-bend pour ce canal.

Description

Donne la valeur actuelle du pitch-bend pour ce canal.

Syntaxe

```
ibend pchbend [imin] [, imax]
```

```
kbend pchbend [imin] [, imax]
```

Initialisation

imin, imax (optionel) -- fixe les limites minimale et maximale pour les valeurs obtenues

Exécution

Donne la valeur actuelle du pitch-bend pour ce canal. Noter que l'on a la valeur du pitch-bend qui est indépendant du pitch MIDI, ce qui permet d'utiliser cette valeur pour n'importe quel but.

Exemples

Voici un exemple de l'opcode pchbend. Il utilise le fichier *pchbend.csd* [examples/pchbend.csd].

Exemple 345. Exemple de l'opcode pchbend.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchbend.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 pchbend

  print i1
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchmidi*, *pchmidib*, *veloc*

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Exemple écrit par Kevin Conder.

pchmidi

pchmidi — Get the note number of the current MIDI event, expressed in pitch-class units.

Description

Get the note number of the current MIDI event, expressed in pitch-class units.

Syntax

```
ipch pchmidi
```

Performance

Get the note number of the current MIDI event, expressed in pitch-class units for local processing.

pchmidi vs. pchmidinn

The *pchmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *pchmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *pchmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *pchmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

Examples

Here is an example of the pchmidi opcode. It uses the file *pchmidi.csd* [examples/pchmidi.csd].

Exemple 346. Example of the pchmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
  ; This example expects MIDI note inputs on channel 1
  i1 pchmidi

  print i1
endin


</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e


</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidib*, *veloc*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

pchmidib

pchmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in pitch-class units.

Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in pitch-class units.

Syntax

```
ipch pchmidib [irange]
```

```
kpch pchmidib [irange]
```

Initialization

irange (optional) -- the pitch bend range in semitones

Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in pitch-class units. Available as an i-time value or as a continuous k-rate value.

Examples

Here is an example of the pchmidib pchmidib. It uses the file *pchmidib.csd* [examples/pchmidib.csd].

Exemple 347. Example of the pchmidib pchmidib.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidib.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; This example expects MIDI note inputs on channel 1
  i1 pchmidib
```

```
  print i1
endin


</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e


</CsScore>
</CsoundSynthesizer>
```

See Also

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *veloc*

Credits

Author: Barry L. Vercoe - Mike Berry
MIT - Mills
May 1997

Example written by Kevin Conder.

pchmidinn

pchmidinn — Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

Description

Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

Syntaxe

```
pchmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

pchmidinn est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave point classe de hauteur. Cette conversion suppose que le do médian (8.00 en *pch*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.

pchmidinn vs. pchmidi

L'opcode *pchmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un évènement Midi, un évènement de partition, un évènement en ligne, ou depuis un autre instrument. La valeur d'entrée de *pchmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *pchmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *pchmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

pchmidinn et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 16. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode pchmidinn. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

Exemple 348. Exemple de l'opcode pchmidinn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
  ; i-time loop to print conversion table
  imidiNN =   0
  loop1:
    icps   = cpsmidinn(imidiNN)
    ioct   = octmidinn(imidiNN)
    ipch   = pchmidinn(imidiNN)

    print   imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
  if (imidiNN < 128) igoto loop1
```

```
endin

instr 2
  ; test k-rate converters
  kMiddleC  =   60
  kcps  = cpsmidinn(kMiddleC)
  koct  = octmidinn(kMiddleC)
  kpch  = pchmidinn(kMiddleC)

  printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cpsmidinn, *octmidinn*, *pchmidi*, *notnum*, *cpspch*, *cpsoct*, *octcps*, *octpch*, *pchoct*

Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

pchoct

pchoct — Convertit une valeur octave-point-partie-décimale en classe de hauteur.

Description

Convertit une valeur octave-point-partie-décimale en classe de hauteur.

Syntaxe

```
pchoct (oct)  (arguments de taux-i ou -k seulement)
```

où l'argument entre parenthèses peut être une expression.

Exécution

pchoct et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

Tableau 17. Valeurs de Hauteur et de Fréquence

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.

Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Exemples

Voici un exemple de l'opcode pchoct. Il utilise le fichier *pchoct.csd* [examples/pchoct.csd].

Exemple 349. Exemple de l'opcode pchoct.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pchoct.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Convert an octave-point-decimal value into a
  ; pitch-class value.
  ioct = 8.75
  ipch = pchoct(ioct)

  print ipch
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ipch = 8.090
```

Voir Aussi

cpsoct, *cpspch*, *octcps*, *octpch*, *cpsmidinn*, *octmidinn*, *pchmidinn*

Crédits

Exemple écrit par Kevin Conder.

pconvolve

convolve — Convolution based on a uniformly partitioned overlap-save algorithm

Description

Convolution based on a uniformly partitioned overlap-save algorithm. Compared to the *convolve* opcode, 'pconvolve' has these benefits:

- small delay
- possible to run in real-time for shorter impulse files
- no pre-process analysis pass
- can often render faster than convolve

Syntax

```
ar1 [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitionsize, ichannel]
```

Initialization

ifilcod -- integer or character-string denoting an impulse response soundfile. multichannel files are supported, the file must have the same sample-rate as the orc. [Note: cvanal files cannot be used!] Keep in mind that longer files require more calculation time [and probably larger partition sizes and more latency]. At current processor speeds, files longer than a few seconds may not render in real-time.

ipartitionsize (optional, defaults to the output buffersize [-b]) -- the size in samples of each partition of the impulse file. This is the parameter that needs tweaking for best performance depending on the impulse file size. Generally, a small size means smaller latency but more computation time. If you specify a value that is not a power-of-2 the opcode will find the next power-of-2 greater and use that as the actual partition size.

ichannel (optional) -- which channel to use from the impulse response data file.

Performance

ain -- input audio signal.

The overall latency of the opcode can be calculated as such [assuming ipartitionsize is a power of 2]

```
ilatency = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr
```

Examples

Instrument 1 shows an example of real-time convolution.

Instrument 2 shows how to do file-based convolution with a 'look ahead' method to remove all delay.

NOTE

You will need to download the impulse response files from noisevault.com or replace the filenames with your own impulse files

```
        sr = 44100
        ksmps = 100
        nchnls = 2


        instr   1
kmix = .5    ; Wet/dry mix. Vary as desired.
kvol = .5*kmix    ; Overall volume level of reverb. May need to adjust
                    ; when wet/dry mix is changed, to avoid clipping.

; do some safety checking to make sure we the parameters a good
kmix = (kmix < 0  || kmix > 1 ? .5 : kmix)
kvol = (kvol < 0 ? 0 : .5*kvol*kmix)

; size of each convolution partion -- for best performance, this parameter needs to be tweaked
ipartitionsize = p4

; calculate latency of pconvolve opcode
idel    = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr
prints "Convolving with a latency of %f seconds%n", idel

; actual processing
al, ar    ins

awetl, awetr pconvolve    kvol*(al+ar), "Mercedes-van.wav", ipartitionsize

; Delay dry signal, to align it with the convoled sig
adryl        delay  (1-kmix)*al, idel
adryr        delay   (1-kmix)*ar, idel

    outs    adryl+awetl, adryr+awetr

    endin



        instr   2
imix = 0.5         ; Wet/dry mix. Vary as desired.
ivol = .5*imix    ; Overall volume level of reverb. May need to adjust
                ; when wet/dry mix is changed, to avoid clipping.

ipartitionsize =    32768    ; size of each convolution partion
idel    = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr    ; latency of pconvol

kcount init idel*kr

; since we are using a soundin [instead of ins] we can
; do a kind of "look ahead" by looping during one k-pass
; without output, creating zero-latency

loop:
    al, ar    soundin "John_Cage_1.aif", 0

    awetl, awetr pconvolve ivol*(al+ar),"FactoryHall.aif", ipartitionsize

    adryl        delay  (1-imix)*al,idel  ; Delay dry signal, to align it with
    adryr        delay   (1-imix)*ar,idel  ;

    kcount = kcount - 1
if kcount > 0 kgoto loop

        outs    awetl+adryl, awetr+adryr

        endin
```

See also

convolve, *dconv*.

Credits

Author: Matt Ingalls
2004

pcount

pcount — Returns the number of pfields belonging to a note event.

Description

pcount returns the number of pfields belonging to a note event.

Syntax

```
icount pcount
```

Initialization

icount - stores the number of pfields for the current note event.

Note

Note that the reported number of pfields is not necessarily what's explicitly written in the score, but the pfields available to the instrument through mechanisms like *pfield carry*.

Examples

Here is an example of the pcount opcode. It uses the file *pcount.csd* [examples/pcount.csd].

Exemple 350. Example of the pcount opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     ; -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
;-o pcount.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Anthony Kozar Dec 2006
instr 1
    inum  pcount
    print inum
endin

</CsInstruments>
<CsScore>
i1   0 3 4 5        ; has 5 pfields
i1   1 3            ; has 5 due to carry
i1   2 3 4 5 6 7    ; has 7
e
</CsScore>
</CsoundSynthesizer>
```

The example will produce the following output:

```
SECTION 1:
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
instr 1:  inum = 5.000
B  0.000 ..  1.000 T  1.000 TT  1.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
instr 1:  inum = 5.000
B  1.000 ..  2.000 T  2.000 TT  2.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 7
instr 1:  inum = 7.000
```

The warnings occur because pfields are not used explicitly by the instrument.

See Also

pindex

Credits

Example by: Anthony Kozar

Dec. 2006

pdclip

pdclip — Performs linear clipping on an audio signal or a phasor.

Description

The *pdclip* opcode allows a percentage of the input range of a signal to be clipped to fullscale. It is similar to simply multiplying the signal and limiting the range of the result, but *pdclip* allows you to think about how much of the signal range is being distorted instead of the scalar factor and has a offset parameter for assymetric clipping of the signal range. *pdclip* is also useful for remapping phasors for phase distortion synthesis.

Syntax

```
aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]
```

Initialization

ibipolar -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

ifullscale -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or -*ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

Performance

ain -- the input signal to be clipped.

aout -- the output signal.

kWidth -- the percentage of the signal range that is clipped (must be between 0 and 1).

kCenter -- an offset for shifting the unclipped window of the signal higher or lower in the range (essentially a DC offset). Values should be in the range [-1, 1] with a value of zero representing no shift (regardless of whether bipolar or unipolar mode is used).

The *pdclip* opcode performs linear clipping on the input signal ain. *kWidth* specifies the percentage of the signal range that is clipped. The rest of the input range is mapped linearly from zero to ifullscale in unipolar mode and from -ifullscale to ifullscale in bipolar mode. When *kCenter* is zero, equal amounts of the top and bottom of the signal range are clipped. A negative value shifts the unclipped range more towards the bottom of the input range and a positive value shifts it more towards the top. ibipolar should be 1 for bipolar operation and 0 for unipolar mode. The default is unipolar mode (ibipolar = 0). ifullscale sets the maximum amplitude of the input and output signals (defaults to 1.0).

This amounts to waveshaping the input with the following transfer function (normalized to ifullscale=1.0 in bipolar mode):

```
 1|   _______      x-axis is input range, y-axis is output
  |  /
  | /          width of clipped region is 2*kWidth
```

```
-1       |/        1      width of unclipped region is 2*(1 - kWidth)
--------------------      kCenter shifts the unclipped region
       /|                left or right (up to kWidth)
      / |
     /  |
------   |-1
```

Bipolar mode can be used for direct, linear distortion of an audio signal. Alternatively, unipolar mode is useful for modifying the output of a phasor before it is used to index a function table, effectively making this a phase distortion technique.

See Also

pdhalf, *pdhalfy*, *limit*, *clip*, *distort1*

Examples

Here is an example of the pdclip opcode. It uses the file *pdclip.csd* [examples/pdclip.csd].

Exemple 351. Example of the pdclip opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; test instrument for pdclip opcode
instr 3

        idur        = p3
        iamp        = p4
        ifreq        = p5
        ifn                = p6

        kenv        linseg        0, .05, 1.0, idur - .1, 1.0, .05, 0
        aosc        oscil        1.0, ifreq, ifn

        kmod        expseg        0.00001, idur, 1.0
        aout        pdclip        aosc, kmod, 0.0, 1.0

                                out                kenv*aout*iamp
endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5

; pdclipped sine wave
i3 0 3 15000 440 1
i3 + 3 15000 330 1
i3 + 3 15000 220 1
s

; pdclipped composite wave
```

```
i3 0 3 15000 440 2
i3 + 3 15000 330 2
i3 + 3 15000 220 2
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

pdhalf

pdhalf — Distorts a phasor for reading the two halves of a table at different rates.

Description

The *pdhalf* opcode is designed to emulate the "classic" phase distortion synthesis method of the Casio CZ-series of synthesizers from the mid-1980's. This technique reads the first and second halves of a function table at different rates in order to warp the waveform. For example, *pdhalf* can smoothly transform a sine wave into something approximating the shape of a saw wave.

Syntax

```
aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]
```

Initialization

ibipolar -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

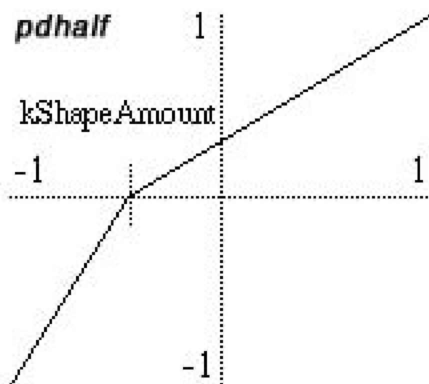
ifullscale -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or -*ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

Performance

ain -- the input signal to be distorted.

aout -- the output signal.

kShapeAmount -- the amount of distortion applied to the input. Must be between negative one and one (-1 to 1). An amount of zero means no distortion.



Transfer function created by pdhalf and a negative kShapeAmount.

The *pdhalf* opcode calculates a transfer function that is composed of two linear segments (see the graph). These segments meet at a "pivot point" which always lies on the same horizontal axis. (In unipolar mode, the axis is y = 0.5, and for bipolar mode it is the x axis). The *kShapeAmount* parameter speci-

fies where on the horizontal axis this point falls. When *kShapeAmount* is zero, the pivot point is in the middle of the input range, forming a straight line for the transfer function and thus causing no change in the input signal. As *kShapeAmount* changes from zero (0) to negative one (-1), the pivot point moves towards the left side of the graph, producing a phase distortion pattern like the Casio CZ's "sawtooth waveform". As it changes from zero (0) to positive one (1), the pivot point moves toward the right, producing an inverted pattern.

If the input to *pdhalf* is a phasor and the output is used to index a table, values for kShapeAmount that are less than zero will cause the first half of the table to be read more quickly than the second half. The reverse is true for values of kShapeAmount greater than zero. The rates at which the halves are read are calculated so that the frequency of the phasor is unchanged. Thus, this method of phase distortion can only produce higher partials in a harmonic series. It cannot produce inharmonic sidebands in the way that frequency modulation does.

pdhalf can work in either unipolar or bipolar modes. Unipolar mode is appropriate for signals like phasors that range between zero and some maximum value (selectable with *ifullscale*). Bipolar mode is appropriate for signals that range above and below zero by roughly equal amounts such as most audio signals. Applying *pdhalf* directly to an audio signal in this way results in a crude but adjustable sort of waveshaping/distortion.

A typical example of the use of pdhalf is

```
aphase    phasor    ifreq
apd       pdhalf    aphase, kamount
aout      tablei    apd, 1, 1
```

More information about Phase distortion synthesis can be found on Wikipedia at *http://en.wikipedia.org/wiki/Phase_distortion_synthesis* [http://en.wikipedia.org/wiki/Phase_distortion_synthesis]

See Also

pdhalfy, *pdclip*

Examples

Here is an example of the pdhalf opcode. It uses the file *pdhalf.csd* [examples/pdhalf.csd].

Exemple 352. Example of the pdhalf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; test instrument for pdhalf opcode
instr 4

    idur        = p3
```

```
    iamp        = p4
    ifreq       = p5
    itable      = p6

    aenv        linseg      0, .001, 1.0, idur - .051, 1.0, .05, 0
    aosc        phasor      ifreq
    kamount     linseg      0.0, 0.02, -0.99, 0.05, -0.9, idur-0.06, 0.0
    apd         pdhalf      aosc, kamount
    aout        tablei      apd, itable, 1

                out         aenv*aout*iamp
endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5
f3 0 16385  9 1 1 270           ; inverted cosine

; descending "just blues" scale

; pdhalf with cosine table
; (imitates the CZ-101 "sawtooth waveform")
t 0 100
i4 0 .333 10000 512     3
i. + .    .     448
i. + .    .     384
i. + .    .     358.4
i. + .    .     341.33
i. + .    .     298.67
i. + 2    .     256
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

pdhalfy

pdhalfy — Distorts a phasor for reading two unequal portions of a table in equal periods.

Description

The *pdhalfy* opcode is a variation on the phase distortion synthesis method of the *pdhalf* opcode. It is useful for distorting a phasor in order to read two unequal portions of a table in the same number of samples.

Syntax

```
aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]
```

Initialization

ibipolar -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

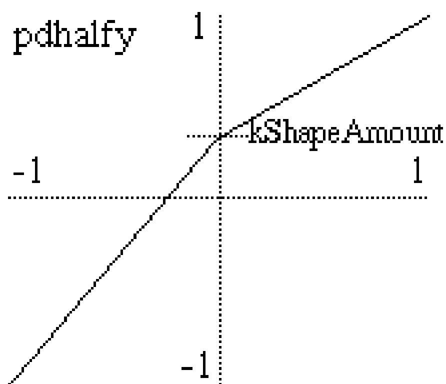
ifullscale -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or -*ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

Performance

ain -- the input signal to be distorted.

aout -- the output signal.

kShapeAmount -- the amount of distortion applied to the input. Must be between negative one and one (-1 to 1). An amount of zero means no distortion.



Transfer function created by pdhalfy and a negative kShapeAmount.

The *pdhalfy* opcode calculates a transfer function that is composed of two linear segments (see the graph). These segments meet at a "pivot point" which always lies on the same vertical axis. (In unipolar mode, the axis is x = 0.5, and for bipolar mode it is the y axis). So, *pdhalfy* is a variation of the *pdhalf*

opcode that places the pivot point of the phase distortion pattern on a vertical axis instead of a horizontal axis.

The *kShapeAmount* parameter specifies where on the vertical axis this point falls. When *kShapeAmount* is zero, the pivot point is in the middle of the output range, forming a straight line for the transfer function and thus causing no change in the input signal. As *kShapeAmount* changes from zero (0) to negative one (-1), the pivot point downward towards the bottom of the graph. As it changes from zero (0) to positive one (1), the pivot point moves upward, producing an inverted pattern.

If the input to *pdhalfy* is a phasor and the output is used to index a table, the use of *pdhalfy* will divide the table into two segments of different sizes with each segment being mapped to half of the oscillator period. Values for kShapeAmount that are less than zero will cause less than half of the table to be read in the first half of the period of oscillation. The rest of the table will be read in the second half of the period. The reverse is true for values of kShapeAmount greater than zero. Note that the frequency of the phasor is always unchanged. Thus, this method of phase distortion can only produce higher partials in a harmonic series. It cannot produce inharmonic sidebands in the way that frequency modulation does. *pdhalfy* tends to have a milder quality to its distortion than *pdhalf*.

pdhalfy can work in either unipolar or bipolar modes. Unipolar mode is appropriate for signals like phasors that range between zero and some maximum value (selectable with *ifullscale*). Bipolar mode is appropriate for signals that range above and below zero by roughly equal amounts such as most audio signals. Applying *pdhalfy* directly to an audio signal in this way results in a crude but adjustable sort of waveshaping/distortion.

A typical example of the use of pdhalfy is

```
aphase    phasor     ifreq
apd       pdhalfy    aphase, kamount
aout      tablei     apd, 1, 1
```

More information about Phase distortion synthesis can be found on Wikipedia at *http://en.wikipedia.org/wiki/Phase_distortion_synthesis* [http://en.wikipedia.org/wiki/Phase_distortion_synthesis]

See Also

pdhalf, *pdclip*

Examples

Here is an example of the pdhalfy opcode. It uses the file *pdhalfy.csd* [examples/pdhalfy.csd].

Exemple 353. Example of the pdhalfy opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>
; test instrument for pdhalfy opcode
instr 5

    idur        = p3
    iamp        = p4
    ifreq       = p5
    iamtinit    = p6      ; initial amount of phase distortion
    iatt        = p7      ; attack time
    isuslvl     = p8      ; sustain amplitude
    idistdec    = p9      ; time for distortion amount to reach zero
    itable      = p10

    idec        =  idistdec - iatt
    irel        =  .05
    isus        =  idur - (idistdec + irel)

    aenv        linseg      0, iatt, 1.0, idec, isuslvl, isus, isuslvl, irel, 0, 0, 0
    kamount     linseg      -iamtinit, idistdec, 0.0, idur-idistdec, 0.0
    aosc        phasor      ifreq
    apd         pdhalfy     aosc, kamount
    aout        tablei      apd, itable, 1

                out         aenv*aout*iamp
endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1                 ; sine
f3 0 16385  9 1 1 270           ; inverted cosine

; descending "just blues" scale

; pdhalfy with cosine table
t 0 100
i5 0 .333 10000 512     1.0   .02  0.5  .12   3
i. + .    .     448     <
i. + .    .     384     <
i. + .    .     358.4   <
i. + .    .     341.33  <
i. + .    .     298.67  <
i. + 2    .     256     0.5
s

; pdhalfy with sine table
t 0 100
i5 0 .333 10000 512     1.0   .001 0.1  .07   1
i. + .    .     448     <
i. + .    .     384     <
i. + .    .     358.4   <
i. + .    .     341.33  <
i. + .    .     298.67  <
i. + 2    .     256     0.5
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

peak

peak — Maintains the output equal to the highest absolute value received.

Description

These opcodes maintain the output k-rate variable as the peak absolute level so far received.

Syntax

```
kres peak asig
```

```
kres peak ksig
```

Performance

kres -- Output equal to the highest absolute value received so far. This is effectively an input to the opcode as well, since it reads *kres* in order to decide whether to write something higher into it.

ksig -- k-rate input signal.

asig -- a-rate input signal.

Examples

Here is an example of the peak opcode. It uses the file *peak.csd* [examples/peak.csd], and *beats.wav* [examples/beats.wav].

Exemple 354. Example of the peak opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o peak.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
  ; Capture the highest amplitude in the "beats.wav" file.
  asig soundin "beats.wav"
  kp peak asig

  ; Print out the peak value once per second.
  printk 1, kp
```

```
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
 i   1 time     0.00002:  4835.00000
 i   1 time     1.00002: 29312.00000
 i   1 time     2.00002: 32767.00000
```

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

peakk

peakk — Obsolète.

Description

Obsolète depuis la version 3.63. Utiliser plutôt l'opcode *peak*.

pgmassign

pgmassign — Affecte un numéro d'instrument à un numéro de programme MIDI spécifié.

Description

Affecte un numéro d'instrument à un (ou à tous) le(s) programme(s) MIDI spécifié(s).

Par défaut, le numéro de l'instrument est le même que celui du programme. Si l'intrument choisi est inférieur ou égal à zéro, ou n'existe pas, le changement de programme est ignoré. Cet opcode est normalement utilisé dans l'en-tête de l'orchestre. Cependant, comme *massign*, il fonctionne aussi dans les instruments.

Syntaxe

```
pgmassign ipgm, inst[, ichn]
```

```
pgmassign ipgm, "insname"[, ichn]
```

Initialisation

ipgm -- numéro de programme MIDI (1 à 128). Une valeur de zéro sélectionne tous les programmes.

inst -- numéro d'instrument. S'il est inférieur ou égal à zéro, les changements de programme MIDI à *ipgm* sont ignorés. Actuellement, l'affectation à un instrument qui n'existe pas a le même effet. Ceci pourra changer dans une version future afin d'imprimer un message d'erreur.

« insname » -- une chaîne de caractères (entre guillemets) représentant un nom d'instrument.

« ichn » (facultatif, par défaut zéro) -- numéro de canal. S'il vaut zéro, les changments de programme sont effectués sur tout les canaux.

Vous pouvez empêcher l'activation de n'importe quel instrument en utilisant l'en-tête ci-dessous :

```
massign 0, 0
pgmassign 0, 0
```

Exemples

Voici un exemple de l'opcode pgmassign. Il utilise le fichier *pgmassign.csd* [examples/pgmassign.csd].

Exemple 355. Exemple de l'opcode pgmassign.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
```

```
; For Non-realtime ouput leave only the line below:
; -o pgmassign.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Program 55 (synth vox) uses Instrument #10.
pgmassign 55, 10

; Instrument #10.
instr 10
  ; Just an example, no working code in here!
endin


</CsInstruments>
<CsScore>

; Play Instrument #10 for one second.
i 10 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode pgmassign qui ignorera les évènements de changement de programme. Il utilise le fichier *pgmassign_ignore.csd* [examples/pgmassign_ignore.csd].

Exemple 356. Exemple de l'opcode pgmassign qui ignorera les évènements de changement de programme.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pgmassign_ignore.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Ignore all program change events.
pgmassign 0, -1

; Instrument #1.
instr 1
  ; Just an example, no working code in here!
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple avancé de l'opcode pgmassign. Il utilise le fichier *pgmassign_advanced.csd* [examples/pgmassign_advanced.csd].

Ne pas oublier qu'il faut inclure l'*option -F* lorsque l'on utilise un fichier MIDI externe comme « pgmassign_advanced.mid ».

Exemple 357. Un exemple avancé de l'opcode pgmassign.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pgmassign_advanced.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr =  44100
ksmps =  10
nchnls =  1

        massign 1, 1 ; channels 1 to 4 use instr 1 by default
        massign 2, 1
        massign 3, 1
        massign 4, 1

; pgmassign.mid has 4 notes with these parameters:
;
;      Start time Channel Program
;
; note 1 0.5     1   10
; note 2 1.5     2   11
; note 3 2.5     3   12
; note 4 3.5     4   13

        pgmassign 0, 0         ; disable program changes
        pgmassign 11, 3         ; program 11 uses instr 3
        pgmassign 12, 2         ; program 12 uses instr 2

; waveforms for instruments
itmp ftgen 1, 0, 1024, 10, 1
itmp ftgen 2, 0, 1024, 10, 1, 0.5, 0.3333, 0.25, 0.2, 0.1667, 0.1429, 0.125
itmp ftgen 3, 0, 1024, 10, 1, 0, 0.3333, 0, 0.2, 0, 0.1429, 0, 0.10101

        instr 1         /* sine */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 1
        out asnd

        endin

        instr 2         /* band-limited sawtooth */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 2
        out asnd

        endin

        instr 3         /* band-limited square */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 3
        out asnd

        endin


</CsInstruments>
<CsScore>
```

```
t 0 120
f 0 8.5 2 -2 0
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

midichn et *massign*

Crédits

Auteur : Istvan Varga
Mai 2002

Nouveau dans la version 4.20

phaser1

phaser1 — First-order allpass filters arranged in a series.

Description

An implementation of *iord* number of first-order allpass filters in series.

Syntax

```
ares phaser1 asig, kfreq, kord, kfeedback [, iskip]
```

Initialization

iskip (optional, default=0) -- used to control initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

kfreq -- frequency (in Hz) of the filter(s). This is the frequency at which each filter in the series shifts its input by 90 degrees.

kord -- the number of allpass stages in series. These are first-order filters and can range from 1 to 4999.

Note

Although *kord* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

kfeedback -- amount of the output which is fed back into the input of the allpass chain. With larger amounts of feedback, more prominent notches appear in the spectrum of the output. *kfeedback* must be between -1 and +1. for stability.

phaser1 implements *iord* number of first-order allpass sections, serially connected, all sharing the same coefficient. Each allpass section can be represented by the following difference equation:

$$y(n) = C * x(n) + x(n-1) - C * y(n-1)$$

where x(n) is the input, x(n-1) is the previous input, y(n) is the output, y(n-1) is the previous output, and C is a coefficient which is calculated from the value of *kfreq*, using the bilinear z-transform.

By slowly varying *kfreq*, and mixing the output of the allpass chain with the input, the classic "phase shifter" effect is created, with notches moving up and down in frequency. This works best with *iord* between 4 and 16. When the input to the allpass chain is mixed with the output, 1 notch is generated for every 2 allpass stages, so that with *iord* = 6, there will be 3 notches in the output. With higher values for *iord*, modulating *kfreq* will result in a form of nonlinear pitch modulation.

Examples

Here is an example of the phaser1 opcode. It uses the file *phaser1.csd* [examples/phaser1.csd].

Exemple 358. Example of the phaser1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phaser1.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; demonstration of phase shifting abilities of phaser1.
instr 1
  ; Input mixed with output of phaser1 to generate notches.
  ; Shows the effects of different iorder values on the sound
  idur   = p3
  iamp   = p4 * .05
  iorder = p5        ; number of 1st-order stages in phaser1 network.
                     ; Divide iorder by 2 to get the number of notches.
  ifreq  = p6        ; frequency of modulation of phaser1
  ifeed  = p7        ; amount of feedback for phaser1

  kamp   linseg 0, .2, iamp, idur - .2, iamp, .2, 0

  iharms = (sr*.4) / 100

  asig   gbuzz 1, 100, iharms, 1, .95, 2  ; "Sawtooth" waveform modulation oscillator for phaser1 ugen.
  kfreq  oscili 5500, ifreq, 1
  kmod   = kfreq + 5600

  aphs   phaser1 asig, kmod, iorder, ifeed

  out    (asig + aphs) * iamp
endin


</CsInstruments>
<CsScore>

; inverted half-sine, used for modulating phaser1 frequency
f1 0  16384 9 .5 -1 0
; cosine wave for gbuzz
f2 0  8192 9 1 1 .25

; phaser1
i1 0  5 7000 4  .2 .9
i1 6  5 7000 6  .2 .9
i1 12 5 7000 8  .2 .9
i1 18 5 7000 16 .2 .9
i1 24 5 7000 32 .2 .9
i1 30 5 7000 64 .2 .9
e


</CsScore>
</CsoundSynthesizer>
```

Technical History

A general description of the differences between flanging and phasing can be found in Hartmann [1]. An early implementation of first-order allpass filters connected in series can be found in Beigel [2], where the bilinear z-transform is used for determining the phase shift frequency of each stage. Cronin [3] presents a similar implementation for a four-stage phase shifting network. Chamberlin [4] and Smith [5] both discuss using second-order allpass sections for greater control over notch depth, width, and frequency.

References

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, June 1978.
2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676,September 1979.
3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, July 1994, p. 78-83.
4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

See Also

phaser2

Credits

Author: Sean Costello
Seattle, Washington
1999

November 2002. Added a note about the *kord* parameter, thanks to Rasmus Ekman.

New in Csound version 4.0

phaser2

phaser2 — Second-order allpass filters arranged in a series.

Description

An implementation of *iord* number of second-order allpass filters in series.

Syntax

```
ares phaser2 asig, kfreq, kq, kord, kmode, ksep, kfeedback
```

Initialization

iskip (optional, default=0) -- used to control initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

kfreq -- frequency (in Hz) of the filter(s). This is the center frequency of the notch of the first allpass filter in the series. This frequency is used as the base frequency from which the frequencies of the other notches are derived.

kq -- Q of each notch. Higher Q values result in narrow notches. A Q between 0.5 and 1 results in the strongest "phasing" effect, but higher Q values can be used for special effects.

kord -- the number of allpass stages in series. These are second-order filters, and iord can range from 1 to 2499. With higher orders, the computation time increases.

kfeedback -- amount of the output which is fed back into the input of the allpass chain. With larger amounts of feedback, more prominent notches appear in the spectrum of the output. *kfeedback* must be between -1 and +1. for stability.

kmode -- used in calculation of notch frequencies.

Note

Although *kord* and *kmode* are listed as k-rate, they are in fact accessed only at init-time. So if you are using k-rate arguments, they must be assigned with *init*.

ksep -- scaling factor used, in conjunction with *imode*, to determine the frequencies of the additional notches in the output spectrum.

phaser2 implements *iord* number of second-order allpass sections, connected in series. The use of second-order allpass sections allows for the precise placement of the frequency, width, and depth of notches in the frequency spectrum. *iord* is used to directly determine the number of notches in the spectrum; e.g. for *iord* = 6, there will be 6 notches in the output spectrum.

There are two possible modes for determining the notch frequencies. When *imode* = 1, the notch frequencies are determined the following function:

frequency of notch N = kbf + (ksep * kbf * N-1)

For example, with *imode* = 1 and *ksep* = 1, the notches will be in harmonic relationship with the notch frequency determined by *kfreq* (i.e. if there are 8 notches, with the first at 100 Hz, the next notches will be at 200, 300, 400, 500, 600, 700, and 800 Hz). This is useful for generating a "comb filtering" effect, with the number of notches determined by *iord*. Different values of *ksep* allow for inharmonic notch frequencies and other special effects. *ksep* can be swept to create an expansion or contraction of the notch frequencies. A useful visual analogy for the effect of sweeping *ksep* would be the bellows of an accordion as it is being played - the notches will be seperated, then compressed together, as *ksep* changes.

When *imode* = 2, the subsequent notches are powers of the input parameter *ksep* times the initial notch frequency specified by *kfreq*. This can be used to set the notch frequencies to octaves and other musical intervals. For example, the following lines will generate 8 notches in the output spectrum, with the notches spaced at octaves of *kfreq*:

aphs *phaser2* ain, kfreq, 0.5, 8, 2, 2, 0
aout = ain + aphs

When *imode* = 2, the value of *ksep* must be greater than 0. *ksep* can be swept to create a compression and expansion of notch frequencies (with more dramatic effects than when *imode* = 1).

Examples

Here is an example of the phaser2 opcode. It uses the file *phaser2.csd* [examples/phaser2.csd].

Exemple 359. Example of the phaser2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phaser2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 2             ; demonstration of phase shifting abilities of phaser2.
  ; Input mixed with output of phaser2 to generate notches.
  ; Demonstrates the interaction of imode and ksep.
  idur   = p3
  iamp   = p4 * .04
  iorder = p5        ; number of 2nd-order stages in phaser2 network
  ifreq  = p6        ; not used
  ifeed  = p7        ; amount of feedback for phaser2
  imode  = p8        ; mode for frequency scaling
  isep   = p9        ; used with imode to determine notch frequencies
  kamp   linseg 0, .2, iamp, idur - .2, iamp, .2, 0
  iharms = (sr*.4) / 100
```

```
  ; "Sawtooth" waveform exponentially decaying function, to control notch frequencies
  asig   gbuzz 1, 100, iharms, 1, .95, 2
  kline  expseg 1, idur, .005
  aphs   phaser2 asig, kline * 2000, .5, iorder, imode, isep, ifeed

  out (asig + aphs) * iamp
endin


</CsInstruments>
<CsScore>

; cosine wave for gbuzz
f2 0  8192 9 1 1 .25

; phaser2, imode=1
i2 00 10 7000 8 .2 .9 1 .33
i2 11 10 7000 8 .2 .9 1 2

; phaser2, imode=2
i2 22 10 7000 8 .2 .9 2 .33
i2 33 10 7000 8 .2 .9 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Technical History

A general description of the differences between flanging and phasing can be found in Hartmann [1]. An early implementation of first-order allpass filters connected in series can be found in Beigel [2], where the bilinear z-transform is used for determining the phase shift frequency of each stage. Cronin [3] presents a similar implementation for a four-stage phase shifting network. Chamberlin [4] and Smith [5] both discuss using second-order allpass sections for greater control over notch depth, width, and frequency.

References

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, June 1978.

2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676,September 1979.

3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, July 1994, p. 78-83.

4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.

5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

See Also

phaser1

Credits

Author: Sean Costello
Seattle, Washington
1999

November 2002. Added a note about the *kord* and *kmode* parameters, thanks to Rasmus Ekman.

New in Csound version 4.0

phasor

phasor — Produit une valeur de phase mobile normalisée.

Description

Produit une valeur de phase mobile normalisée.

Syntaxe

```
ares phasor xcps [, iphs]
```

```
kres phasor kcps [, iphs]
```

Initialisation

iphs (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

Exécution

Une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle 0 <= phs < 1.

Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

Noter que *phasor* est une sorte d'intégrateur, accumulant les incréments de phase qui représent les réglages de fréquence.

Exemples

Voici un exemple de l'opcode phasor. Il utilise le fichier *phasor.csd* [examples/phasor.csd].

Exemple 360. Exemple de l'opcode phasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phasor.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Create an index that repeats once per second.
  kcps init 1
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kfreq table kndx, ifn, ixmode

  ; Generate a sine waveform, use our table values
  ; to vary its frequency.
  a1 oscil 20000, kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

Les opcodes d'*Accès aux Tables* comme : *table*, *tablei*, *table3* et *tab*.

Aussi : *phasorbnk*.

Crédits

Exemple écrit par Kevin Conder.

phasorbnk

phasorbnk — Produit un nombre arbitraire de valeurs de phase mobiles normalisées.

Description

Produit un nombre arbitraire de valeurs de phase mobiles normalisées, accessibles par un indice.

Syntaxe

```
ares phasorbnk xcps, kndx, icnt [, iphs]
```

```
kres phasorbnk kcps, kndx, icnt [, iphs]
```

Initialisation

icnt -- nombre maximum de phaseurs à utiliser.

iphs -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Si elle vaut -1, l'initialisation sera ignorée. Si *iphas*>1 chaque phaseur sera initialisé avec une valeur aléatoire.

Exécution

kndx -- valeur d'indice pour accéder aux phaseurs individuellement

Pour chaque phaseur indépendant, une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle 0 <= phs < 1. On accède à chaque phaseur individuel par l'indice *kndx*.

On peut utiliser cette banque de phaseurs dans une boucle de taux-k pour générer plusieurs voix indépendantes, ou en conjonction avec l'opcode *adsynt* pour changer les paramètres dans les tables utilisées par *adsynt*.

Exemples

Voici un exemple de l'opcode phasorbnk. Il utilise le fichier *phasorbnk.csd* [examples/phasorbnk.csd].

Exemple 361. Exemple de l'opcode phasorbnk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phasorbnk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1

; Instrument #1
instr 1
  ; Generate 10 voices.
  icnt = 10
  ; Empty the output buffer.
  asum = 0
  ; Reset the loop index.
  kindex = 0

; This loop is executed every k-cycle.
loop:
  ; Generate non-harmonic partials.
  kcps = (kindex+1)*100+30
  ; Get the phase for each voice.
  aphas phasorbnk kcps, kindex, icnt
  ; Read the wave from the table.
  asig table aphas, giwave, 1
  ; Accumulate the audio output.
  asum = asum + asig

  ; Increment the index.
  kindex = kindex + 1

  ; Perform the loop until the index (kindex) reaches
  ; the counter value (icnt).
  if (kindex < icnt) kgoto loop

  out asum*3000
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Génère plusieurs voix avec des partiels indépendants. Cet exemple est meilleur avec *adsynt*. Voir aussi l'exemple de la notice *adsynt* pour une utilisation de *phasorbnk* au taux-k.

Crédits

Auteur : Peter Neubäcker
Munich, Allemagne
Août 1999

Nouveau dans la version 3.58 de Csound

pindex

pindex — Returns the value of a specified pfield.

Description

pindex returns the value of a specified pfield.

Syntax

```
ivalue pindex ipfieldIndex
```

Initialization

ipfieldIndex - pfield number to query.

ivalue - value of the pfield.

Examples

Here is an example of the pindex opcode. It uses the file *pindex.csd* [examples/pindex.csd].

Exemple 362. Example of the pindex opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     ; -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
;-o pindex.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Anthony Kozar Dec 2006

instr 1
    inum    pcount
    index   init 1
    loop1:
        ivalue pindex index
        printf_i "p%d = %f\n", 1, index, ivalue
        index   = index + 1
    if  (index <= inum) igoto loop1
    print inum
endin

</CsInstruments>
<CsScore>
i1  0 3 40 50          ; has 5 pfields
i1  1 2 80             ; has 5 due to carry
i1  2 1 40 50 60 70    ; has 7
e
</CsScore>
</CsoundSynthesizer>
```

The example will produce the following output:

```
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 0.000000
p3 = 3.000000
p4 = 40.000000
p5 = 50.000000
instr 1:  inum = 5.000
B  0.000 ..  1.000 T  1.000 TT  1.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 1.000000
p3 = 2.000000
p4 = 80.000000
p5 = 50.000000
instr 1:  inum = 5.000
B  1.000 ..  2.000 T  2.000 TT  2.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 7
p1 = 1.000000
p2 = 2.000000
p3 = 1.000000
p4 = 40.000000
p5 = 50.000000
p6 = 60.000000
p7 = 70.000000
instr 1:  inum = 7.000
```

The warnings can be ignored, because the pfields are used indirectly through pindex instead of explicitly through p4, p5, etc.

See Also

pcount

Credits

Example by: Anthony Kozar

Dec. 2006

pinkish

pinkish — Génère une approximation d'un bruit rose.

Description

Génère une approximation d'un bruit rose (réponse à -3dB/oct) par une de ces deux méthodes :

- un générateur de bruit à taux multiples d'après Moore, codé par Martin Gardner
- un banc de filtres dessinés par Paul Kellet

Syntaxe

```
ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]
```

Initialisation

imethod (facultatif, par défaut=0) -- sélectionne la méthode de filtrage :

- 0 = méthode de Gardner (par défaut).
- 1 = banc de filtres de Kellet.
- 2 = Un banc de filtres quelque peu plus rapides par Kellet, avec une réponse moins précise.

inumbands (facultatif) -- ne fonctionne qu'avec la méthode de Gardner. Nombre de bandes de bruit à générer. Le maximum vaut 32 et le minimum vaut 4. Les valeurs plus élevées donnent un spectre plus lisse, mais au-delà de 20 bandes il y aura des fluctuations lentes presque comme une composante continue. La valeur par défaut est 20.

iseed (facultatif, par défaut=0) -- ne fonctionne qu'avec la méthode de Gardner. s'il est non nul, sert de graine au générateur de nombres aléatoires. S'il est nul, le générateur sera initialisé à partir de la valeur de l'horloge. Vaut 0 par défaut.

iskip (facultatif, par défaut=0) -- s'il est non nul, l'état interne n'est pas (ré)initialisé (utile pour les notes liées). Vaut 0 par défaut.

Exécution

xin -- pour la méthode de Gardner : amplitude de taux-k ou -a. Pour les filtres de Kellet : normalement un bruit de taux-a de distribution uniforme obtenu à partir de *rand* (31-bit) ou de *unirand*, mais ça peut être n'importe quel signal de taux-a. La valeur de crête de la sortie varie largement (±15%) même sur de longues périodes, et sera habituellement d'un niveau bien inférieur à celui de l'amplitude de l'entrée. Les valeurs de crête peuvent aussi dépasser occasionnellement l'amplitude de l'entrée ou celle du bruit.

pinkish tente de générer un bruit rose (c-à-d un bruit avec la même énergie dans chaque octave), par une des deux méthodes suivantes.

La première méthode, par Moore & Gardner, ajoute plusieurs signaux de bruit blanc (jusqu'à 32), géné-

rés à des taux en octave (*sr*, *sr*/2, *sr*/4, etc). Les valeurs pseudo aléatoires sont obtenues à partir d'un générateur interne sur 32 bit. Ce générateur est local à chaque instance de l'opcode et initialisable (comme pour *rand*).

La seconde méthode est un filtrage passe-bas avec une réponse d'environ -3dB/oct. Si l'entrée est un bruit blanc uniforme, la sortie sera un bruit rose. Avec cette méthode, on peut utiliser n'importe quel signal comme entrée. Le filtre de haute qualité est plus lent, mais il a moins d'ondulations et un intervalle de fréquences opératoires légèrement plus large que les versions moins gourmandes en calcul. Avec les filtres de Kellet, il n'y a pas de graine pour le générateur de nombres aléatoires.

La réponse en fréquence de la sortie dans la méthode de Gardner comporte quelques anomalies dans les intervalles basse-moyenne et moyenne-haute fréquence. On peut générer plus d'énergie en basse fréquence en augmentant le nombre de bandes. Cette méthode est aussi un peu plus rapide. Le filtre raffiné de Kellet a un spectre très lisse, mais un intervalle efficace plus limité. Le niveau augmente légèrement à l'extrémité haute du spectre.

Exemples

Voici un exemple de l'opcode pinkish. Il utilise le fichier *pinkish.csd* [examples/pinkish.csd].

Exemple 363. Exemple de l'opcode pinkish.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pinkish.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  awhite unirand 2.0

  ; Normalize to +/-1.0
  awhite = awhite - 1.0

  apink  pinkish awhite, 1, 0, 0, 1

  out apink * 30000
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Bruit filtré (Kellet) pour une note liée (*iskip* est non nul).

Crédits

Auteurs : Phil Burk et John ffitch
Université de Bath/Codemist Ltd.
Bath, UK
Mai 2000

Nouveau dans la version 4.07 de Csound

Adapté pour Csound par Rasmus Ekman

La méthode par bandes de bruit est dûe à F. R. Moore (ou R. F. Voss), et fut présentée par Martin Gardner dans un article de Scientific American souvent cité. La présente version fut codée par Phil Burk après une discussion sur la liste de diffusion de music-dsp, avec des optimisations significatives suggérées par James McCartney.

Le banc de filtres a été dessiné par Paul Kellet, et posté sur la liste de diffusion de music-dsp.

La discussssion complète sur le bruit rose a été rassemblée sur une page HTML par Robin Whittle, qui est actuellement consultable à *http://www.firstpr.com.au/dsp/pink-noise/*.

Notes ajoutées par Rasmus Ekman en Septembre 2002.

pitch

pitch — Tracks the pitch of a signal.

Description

Using the same techniques as *spectrum* and *specptrk*, pitch tracks the pitch of the signal in octave point decimal form, and amplitude in dB.

Syntax

```
koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh [, ifrqs] [, iconf] \
      [, istrt] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]
```

Initialization

iupdte -- length of period, in seconds, that outputs are updated

ilo, *ihi* -- range in which pitch is detected, expressed in octave point decimal

idbthresh -- amplitude, expressed in decibels, necessary for the pitch to be detected. Once started it continues until it is 6 dB down.

ifrqs (optional) -- number of divisons of an octave. Default is 12 and is limited to 120.

iconf (optional) -- the number of conformations needed for an octave jump. Default is 10.

istrt (optional) -- starting pitch for tracker. Default value is (*ilo* + *ihi*)/2.

iocts (optional) -- number of octave decimations in spectrum. Default is 6.

iq (optional) -- Q of analysis filters. Default is 10.

inptls (optional) -- number of harmonics, used in matching. Computation time increases with the number of harmonics. Default is 4.

irolloff (optional) -- amplitude rolloff for the set of filters expressed as fraction per octave. Values must be positive. Default is 0.6.

iskip (optional) -- if non-zero, skips initialization. Default is 0.

Performance

koct -- The pitch output, given in the octave point decimal format.

kamp -- The amplitude output.

pitch analyzes the input signal, *asig*, to give a pitch/amplitude pair of outputs, for the strongest frequency in the signal. The value is updated every *iupdte* seconds.

The number of partials and rolloff fraction can effect the pitch tracking, so some experimentation may be necessary. Suggested values are 4 or 5 harmonics, with rolloff 0.6, up to 10 or 12 harmonics with rolloff 0.75 for complex timbres, with a weak fundamental.

Examples

Here is an example of the pitch opcode. It uses the file *pitch.csd* [examples/pitch.csd] and *mary.wav* [examples/mary.wav].

Exemple 364. Example of the pitch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pitch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file without effects.
instr 1
  asig soundin "mary.wav"
  out asig
endin

; Instrument #2 - track the pitch of an audio file.
instr 2
  iupdte = 0.01
  ilo = 7
  ihi = 9
  idbthresh = 10
  ifrqs = 12
  iconf = 10
  istrt = 8

  asig soundin "mary.wav"

  ; Follow the audio file, get its pitch and amplitude.
  koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh, ifrqs, iconf, istrt

  ; Re-synthesize the audio file with a different sounding waveform.
  kamp2 = kamp * 10
  kcps = cpsoct(koct)
  a1 oscil kamp2, kcps, 1

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: A different sounding waveform.
f 1 0 32768 11 7 3 .7

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
; Play Instrument #2, the "re-synthesized" waveform, for three seconds.
i 2 3 3
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch
University of Bath, Codemist Ltd.
Bath, UK
April 1999

Example written by Kevin Conder.

New in Csound version 3.54

pitchamdf

pitchamdf — Follows the pitch of a signal based on the AMDF method.

Description

Follows the pitch of a signal based on the AMDF method (Average Magnitude Difference Function). Outputs pitch and amplitude tracking signals. The method is quite fast and should run in realtime. This technique usually works best for monophonic signals.

Syntax

```
kcps, krms pitchamdf asig, imincps, imaxcps [, icps] [, imedi] \
      [, idowns] [, iexcps] [, irmsmedi]
```

Initialization

imincps -- estimated minimum frequency (expressed in Hz) present in the signal

imaxcps -- estimated maximum frequency present in the signal

icps (optional, default=0) -- estimated initial frequency of the signal. If 0, icps = (*imincps*+*imaxcps*) / 2. The default is 0.

imedi (optional, default=1) -- size of median filter applied to the output *kcps*. The size of the filter will be *imedi**2+1. If 0, no median filtering will be applied. The default is 1.

idowns (optional, default=1) -- downsampling factor for *asig*. Must be an integer. A factor of *idowns* > 1 results in faster performance, but may result in worse pitch detection. Useful range is 1 - 4. The default is 1.

iexcps (optional, default=0) -- how frequently pitch analysis is executed, expressed in Hz. If 0, *iexcps* is set to *imincps*. This is usually reasonable, but experimentation with other values may lead to better results. Default is 0.

irmsmedi (optional, default=0) -- size of median filter applied to the output *krms*. The size of the filter will be *irmsmedi**2+1. If 0, no median filtering will be applied. The default is 0.

Performance

kcps -- pitch tracking output

krms -- amplitude tracking output

pitchamdf usually works best for monophonic signals, and is quite reliable if appropriate initial values are chosen. Setting *imincps* and *imaxcps* as narrow as possible to the range of the signal's pitch, results in better detection and performance.

Because this process can only detect pitch after an initial delay, setting *icps* close to the signal's real initial pitch prevents spurious data at the beginning.

The median filter prevents *kcps* from jumping. Experiment to determine the optimum value for *imedi* for a given signal.

Other initial values can usually be left at the default settings. Lowpass filtering of *asig* before passing it to *pitchamdf*, can improve performance, especially with complex waveforms.

Examples

Here is an example of the pitchamdf opcode. It uses the file *pitchamdf.csd* [examples/pitchamdf.csd] and *mary.wav* [examples/mary.wav].

Exemple 365. Example of the pitchamdf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pitchamdf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 2, 0, 1024, 10, 1, 1, 1, 1

; Instrument #1 - play an audio file with no effects.
instr 1
  ; get input signal with original freq.
  asig soundin "mary.wav"

  out asig
endin

; Instrument #2 - play the synth waveform using the
; same pitch and amplitude as the audio file.
instr 2
  ; get input signal with original freq.
  asig soundin "mary.wav"

  ; lowpass-filter
  asig tone asig, 1000
  ; extract pitch and envelope
  kcps, krms pitchamdf asig, 150, 500, 200
  ; "re-synthesize" with the synth waveform, giwave.
  asig1 oscil krms, kcps, giwave

  out asig1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
; Play Instrument #2, the "re-synthesized" waveform, for three seconds.
i 2 3 3
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Peter Neubäcker
Munich, Germany
August 1999

New in Csound version 3.59

planet

planet — Simulation d'un planète en orbite dans un système d'étoile binaire.

Description

planet simule l'orbite d'une planète dans un système d'étoile binaire. Les sorties sont les coordonnées x, y et z de la planète en orbite. Il est possible que la planète atteigne sa vitesse de libération si elle croise une étoile de très près. Cela rend le système quelque peu instable.

Syntaxe

```
ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \
      [, ifriction] [, iskip]
```

Initialisation

ix, iy, iz -- les coordonnées initiales x, y et z de la planète

ivx, ivy, ivz -- les composantes initiales du vecteur vitesse de la planète.

idelta -- la taille du pas utilisé dans l'approximation de l'équation différentielle.

ifriction (facultatif, 0 par défaut) -- une valeur de frottement que l'on peut utiliser pour empêcher le système de diverger

iskip (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

Exécution

ax, ay, az -- les coordonnées x, y et z de la planète en sortie

kmass1 -- la masse de la première étoile

kmass2 -- la masse de la seconde étoile

Exemples

Voici un exemple de l'opcode planet. Il utilise le fichier *planet.csd* [examples/planet.csd].

Exemple 366. Exemple de l'opcode planet.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o planet.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a planet oribiting in 3D space.
instr 1
  ; Create a basic tone.
  kamp init 5000
  kcps init 440
  ifn = 1
  asnd oscil kamp, kcps, ifn

  ; Figure out its X, Y, Z coordinates.
  km1 init 0.5
  km2 init 0.35
  ksep init 2.2
  ix = 0
  iy = 0.1
  iz = 0
  ivx = 0.5
  ivy = 0
  ivz = 0
  ih = 0.0003
  ifric = -0.1
  ax1, ay1, az1 planet km1, km2, ksep, ix, iy, iz, \
                       ivx, ivy, ivz, ih, ifric

  ; Place the basic tone within 3D space.
  kx downsamp ax1
  ky downsamp ay1
  kz downsamp az1
  idist = 1
  ift = 0
  imode = 1
  imdel = 1.018853416
  iovr = 2
  aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
                          ift, imode, imdel, iovr

  ; Convert the 3D sound to stereo.
  aleft = aw2 + ay2
  aright = aw2 - ay2

  outs aleft, aright
endin


</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 10 seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Hans Mikelson
Décembre 1998

Nouveau dans la version 3.50 de Csound

pluck

pluck — Produces a naturally decaying plucked string or drum sound.

Description

Audio output is a naturally decaying plucked string or drum sound based on the Karplus-Strong algorithms.

Syntax

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]
```

Initialization

icps -- intended pitch value in Hz, used to set up a buffer of 1 cycle of audio samples which will be smoothed over time by a chosen decay method. *icps* normally anticipates the value of *kcps*, but may be set artificially high or low to influence the size of the sample buffer.

ifn -- table number of a stored function used to initialize the cyclic decay buffer. If *ifn* = 0, a random sequence will be used instead.

imeth -- method of natural decay. There are six, some of which use parameters values that follow.

1. simple averaging. A simple smoothing process, uninfluenced by parameter values.
2. stretched averaging. As above, with smoothing time stretched by a factor of iparm1 (=1).
3. simple drum. The range from pitch to noise is controlled by a 'roughness factor' in iparm1 (0 to 1). Zero gives the plucked string effect, while 1 reverses the polarity of every sample (octave down, odd harmonics). The setting .5 gives an optimum snare drum.
4. stretched drum. Combines both roughness and stretch factors. iparm1 is roughness (0 to 1), and iparm2 the stretch factor (=1).
5. weighted averaging. As method 1, with iparm1 weighting the current sample (the status quo) and iparm2 weighting the previous adjacent one. iparm1 + iparm2must be <= 1.
6. 1st order recursive filter, with coefs .5. Unaffected by parameter values.

iparm1, iparm2 (optional) -- parameter values for use by the smoothing algorithms (above). The default values are both 0.

Performance

kamp -- the output amplitude.

kcps -- the resampling frequency in cycles-per-second.

An internal audio buffer, filled at i-time according to *ifn*, is continually resampled with periodicity *kcps* and the resulting output is multiplied by *kamp*. Parallel with the sampling, the buffer is smoothed to si-

mulate the effect of natural decay.

Plucked strings (1,2,5,6) are best realized by starting with a random noise source, which is rich in initial harmonics. Drum sounds (methods 3,4) work best with a flat source (wide pulse), which produces a deep noise attack and sharp decay.

The original Karplus-Strong algorithm used a fixed number of samples per cycle, which caused serious quantization of the pitches available and their intonation. This implementation resamples a buffer at the exact pitch given by *kcps*, which can be varied for vibrato and glissando effects. For low values of the orch sampling rate (e.g. *sr* = 10000), high frequencies will store only very few samples (*sr / icps*). Since this may cause noticeable noise in the resampling process, the internal buffer has a minimum size of 64 samples. This can be further enlarged by setting *icps* to some artificially lower pitch.

Examples

Here is an example of the pluck opcode. It uses the file *pluck.csd* [examples/pluck.csd].

Exemple 367. Example of the pluck opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pluck.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  icps = 440
  ifn = 0
  imeth = 1

  a1 pluck kamp, kcps, icps, ifn, imeth
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Example written by Kevin Conder.

poisson

poisson — Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares poisson klambda
```

```
ires poisson klambda
```

```
kres poisson klambda
```

Exécution

ares, kres, ires - nombre d'évènements se produisant (toujours un entier).

klambda - le nombre attendu d'évènements par intervalle d'échantillonnage.

Adapté de Wikipédia :

En théorie des probabilités et en statistiques, la distribution de Poisson est une distribution de probabilité discrète. Elle exprime la probabilité d'apparition d'un certain nombre d'évènements pendant une période de temps fixée si ces évènements se produisent avec un taux moyen connu et indépendamment du temps écoulé depuis le dernier évènement.

La distribution de Poisson décrivant la probabilité qu'il y ait exactement *k* évènements (*k* étant un nombre non négatif, k = 0, 1, 2, ...) est :

$$f(k; \lambda) = \frac{e^{-\lambda} \lambda^k}{k!},$$

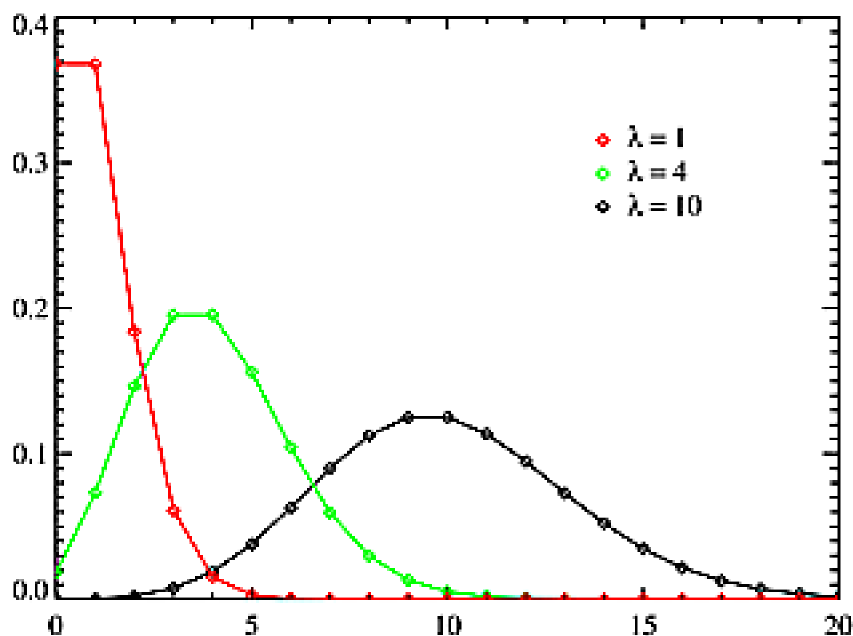
où :

- # est un nombre réel positif, égal au nombre attendu d'évènements se produisant durant l'intervalle donné. Par exemple, si les évènements se produisent en moyenne toutes les 4 minutes, et que l'on est intéressé par le nombre d'évènements se produisant dans un intervalle de 10 minutes, on utilisera comme modèle une distribution de Poisson avec # = 10/4 = 2,5. Ce paramètre se nomme *klambda* dans les opcodes *poisson*.
- *k* fait référence au nombre de i-, k- ou a- périodes écoulées.

La distribution de Poisson apparaît aussi avec les processus de Poisson. Elle s'applique à différents phénomènes de nature discrète (c-à-d, ceux qui peuvent se produire 0, 1, 2, 3, ... fois durant une période de temps donnée ou dans un espace donné) chaque fois que la probabilité du phénomène se produisant est constante dans le temps ou dans l'espace. Parmi les exemples qui peuvent être modélisés par une distri-

bution de Poisson, on trouve :

- Le nombre d'automobiles passant devant un repère sur une route (suffisamment éloigné des feux de circulation) pendant un intervalle de temps donné.
- Le nombre de fautes de frappe que l'on fait lorsque l'on tape une page.
- Le nombre d'appels par minute dans un centre d'appel téléphonique.
- Le nombre d'accès par minute à un serveur web.
- Le nombre d'animaux écrasés par unité de longueur sur une route.
- Le nombre de mutations dans un brin d'ADN après une certaine quantité de radiations.
- Le nombre de noyaux instables qui a diminué pendant une période de temps donnée dans un morceau de substance radioactive. Comme la radioactivité de la substance diminue avec le temps, l'intervalle de temps total utilisé dans le modèle doit être significativement inférieur à la durée de vie moyenne de la substance.
- Le nombre de pins par unité de surface dans une forêt hétérogène.
- Le nombre d'étoiles dans une région donnée de l'espace.
- La distribution des cellules réceptrices de la vision dans la rétine de l'oeil humain.
- Le nombre de virus qui peuvent infecter une cellule dans une culture de cellules.



Un diagramme montrant la distribution de Poisson.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode poisson. Il utilise le fichier *poisson.csd* [examples/poisson.csd]. Il est écrit pour des systèmes *NIX et génèrera des erreurs sur Windows.

Exemple 368. Exemple de l'opcode poisson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poisson.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 441  ;ksmps set deliberately high to have few k-periods per second
nchnls = 1

; Instrument #1.
instr 1
  ; Generates a random number in a poisson distribution.
  ; klambda = 1

  i1 poisson 1

  print i1
endin

instr 2

kres poisson p4
printk (ksmps/sr),kres ;prints every k-period
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
i 2 1 0.2 0.5
i 2 2 0.2 4   ;average 4 events per k-period
i 2 3 0.2 20  ;average 20 events per k-period
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *trirand*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder et Andrés Cabrera

polyaft

polyaft — Retourne la pression d'after-touch polyphonique du numéro de note sélectionné.

Description

polyaft retourne la pression polyphonique du numéro de note choisi, optionnellement mappé dans un intervalle défini par l'utilisateur.

Syntaxe

```
ires polyaft inote [, ilow] [, ihigh]
```

```
kres polyaft inote [, ilow] [, ihigh]
```

Initialisation

inote -- numéro de note. Normalement ajusté à la valeur retournée par *notnum*

ilow (facultatif, par défaut : 0) -- la valeur de sortie la plus basse

ihigh (facultatif, par défaut : 127) -- la valeur de sortie la plus haute

Exécution

kres -- Pression polyphonique (aftertouch).

Exemples

Voici un exemple de l'opcode polyaft. Il utilise le fichier *polyaft.csd* [examples/polyaft.csd].

Ne pas oublier d'inclure l'*option -F* lorque l'on utilise un fichier MIDI externe comme « polyaft.mid ».

Exemple 369. Exemple de l'opcode polyaft.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d         -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o polyaft.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr =  44100
ksmps =  10
nchnls =  1

        massign 1, 1
```

```
itmp ftgen 1, 0, 1024, 10, 1          ; sine wave

        instr 1

kcps cpsmidib 2         ; note frequency
inote notnum                 ; note number
kaft polyaft inote, 0, 127 ; aftertouch
; interpolate aftertouch to eliminate clicks
ktmp phasor 40
ktmp trigger 1 - ktmp, 0.5, 0
kaft tlineto kaft, 0.025, ktmp
; map to sine curve for crossfade
kaft =  sin(kaft * 3.14159 / 254) * 22000

asnd oscili kaft, kcps, 1

        out asnd

        endin


</CsInstruments>
<CsScore>

t 0 120
f 0 9 2 -2 0
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Ajouté grâce à un courriel de Istvan Varga

Nouveau dans la version 4.12

polynomial

polynomial — Efficiently evaluates a polynomial of arbitrary order.

Description

The *polynomial* opcode calculates a polynomial with a single a-rate input variable. The polynomial is a sum of any number of terms in the form *kn*x^n* where *kn* is the nth coefficient of the expression. These coefficients are k-rate values.

Syntax

```
aout polynomial ain, k0 [, k1 [, k2 [...]]]
```

Performance

ain -- the input signal used as the independent variable of the polynomial ("x").

aout -- the output signal ("y").

k0, k1, k2, ... -- the coefficients for each term of the polynomial.

If we consider the input parameter *ain* to be "x" and the output *aout* to be "y", then the *polynomial* opcode calculates the following equation:

$$y = k0 + k1*x + k2*x^2 + k3*x^3 + ...$$

See Also

chebyshevpoly, *mac maca*

Examples

Here is an example of the polynomial opcode. It uses the file *polynomial.csd* [examples/polynomial.csd].

Exemple 370. Example of the polynomial opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
  ; no sound output
  -n
</CsOptions>
<CsInstruments>

sr = 10 ; audio rate is not important
```

```
kr = 10 ; execute the statements in instr 1 ten times per second

instr 1
        ; ax will vary from 1 to 10
        ax          init        1

        ; ay = ax^3 + 2ax^2 + 3ax + 4
        ay          polynomial  ax, 4, 3, 2, 1

        ; convert our a-rate signals to k-rate values so that we can print
        ky          downsamp    ay
        kx          downsamp    ax
                    printf      "%d:  %d\n", kx, kx, ky

        ax          =           ax + 1
endin

</CsInstruments>
<CsScore>

i1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

pop

push — Pops values from the global stack.

Description

Pops values from the global stack.

Syntax

```
xval1, [xval2, ... , xval31] pop
```

```
ival1, [ival2, ... , ival31] pop
```

Initialization

ival1 ... ival31 - values to be popped from the stack.

Performance

xval1 ... xval31 - values to be popped from the stack.

The given values are poped from the stack. The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

push and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Use of any combination of i, k, a, and S types is allowed. Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push_f* only pushes a "reference" to the f-signal, and then the corresponding *pop_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push_f* before *pop_f* is called is not recommended, and if the instrument instance owning the variable that was passed by push_f is deactivated before pop_f is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

See also

stack, *push*, *pop_f* and *push_f*.

Credits

By: Istvan Varga.

2006

pop_f

pop_f — Pops an f-sig frame from the global stack.

Description

Pops an f-sig frame from the global stack.

Syntax

```
fsig pop_f
```

Performance

fsig - f-signal to be popped from the stack.

The values are popped the stack. The global stack must be initialized before used, and its size must be set. The global stack works in LIFO order: after multiple *push_f* calls, *pop_f* should be used in reverse order.

push/pop for a, k, i, and S types copy data by value. By contrast, *push_f* only pushes a "reference" to the f-signal, and then the corresponding *pop_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push_f* before *pop_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push_f* is deactivated before *pop_f* is called, undefined behavior may occur.

push_f and *pop_f* can only take a single argument, and the data is passed both at init and performance time.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

See also

stack, *push*, *pop* and *push_f*.

Credits

By: Istvan Varga.

2006

port

port — Applique un portamento à un signal de contrôle en escalier.

Description

Applique un portamento à un signal de contrôle en escalier.

Syntaxe

```
kres port ksig, ihtim [, isig]
```

Initialisation

ihtim -- durée à mi-parcours de la fonction, en secondes.

isig (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0. Avec une valeur négative l'initialisation sera ignorée et la dernière valeur de l'instance précédente sera la valeur initiale de la note.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

port applique un portamento à un signal de contrôle en escalier. A chaque nouveau palier, *ksig* est filtré par un filtre passe-bas pour que la transition vers cette valeur se fasse au taux déterminé par *ihtim*. *ihtim* est la durée à « mi-parcours » de la fonction (en secondes), au cours de laquelle la courbe parcourera la moitié de la distance la séparant de la nouvelle valeur, puis la moitié de la moitié, etc., n'atteignant théoriquement jamais son asymptote. Avec *portk*, la durée à mi-parcours peut être variée au taux de contrôle.

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *portk*, *reson*, *resonk*, *tone*, *tonek*

portk

portk — Applique un portamento à un signal de contrôle en escalier.

Description

Applique un portamento à un signal de contrôle en escalier.

Syntaxe

```
kres portk ksig, khtim [, isig]
```

Initialisation

isig (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

khtim -- durée à mi-parcours de la fonction, en secondes.

portk est semblable à *port* à part le fait que la durée à mi-parcours peut-être variée au taux de contrôle.

Exemples

Voici un exemple de l'opcode portk. Il utilise le fichier *portk.csd* [examples/portk.csd].

Exemple 371. Exemple de l'opcode portk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          ;  -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o portk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

FLpanel "Slider", 650, 140, 50, 50
    gkval1, gislider1 FLslider "Watch me", 0, 127, 0, 5, -1, 580, 30, 25, 20
    gkval2, gislider2 FLslider "Move me", 0, 127, 0, 5, -1, 580, 30, 25, 80
```

```
    gkhtim, gislider3 FLslider "khtim", 0.1, 1, 0, 6, -1, 30, 100, 610, 10
FLpanelEnd
FLrun

FLsetVal_i 0.1, gislider3 ;set initial time to 0.1

instr 1
kval portk gkval2, gkhtim  ; take the value of slider 2 and apply portamento
FLsetVal 1, kval, gislider1  ;set the value of slider 1 to kval
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *port*, *reson*, *resonk*, *tone*, *tonek*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

poscil

poscil — Oscillateur haute précision.

Description

Oscillateur haute précision.

Syntaxe

```
ares poscil aamp, acps, ifn [, iphs]

ares poscil aamp, kcps, ifn [, iphs]

ares poscil kamp, acps, ifn [, iphs]

ares poscil kamp, kcps, ifn [, iphs]

ires poscil kamp, kcps, ifn [, iphs]

kres poscil kamp, kcps, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction

iphs (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1)

Exécution

ares -- signal de sortie

kamp, *aamp* -- l'amplitude du signal de sortie.

kcps, *acps* -- la fréquence du signal de sortie en cycles par seconde.

poscil (oscillateur de précision) est identique à *oscili*, mais il permet un contrôle de la fréquence plus précis, en particulier lorsque l'on utilise de grandes tables avec de faibles valeurs de fréquence. Il utilise une indexation de la table en virgule flottante, au lieu de l'arithmétique entière utilisée par *oscil* et *oscili*. Il est à peine plus lent que *oscili*.

Depuis Csound 4.22, *poscil* accepte aussi des valeurs de fréquence négatives et il peut utiliser des valeurs de taux-a aussi bien pour l'amplitude que pour la fréquece. Ainsi, cet opcode permet la modulation d'amplitude (MA) et la modulation de fréquence (MF).

L'opcode *poscil3* est le même que *poscil*, mais il utilise une interpolation cubique.

Exemples

Voici un exemple de l'opcode poscil. Il utilise le fichier *poscil.csd* [examples/poscil.csd].

Exemple 372. Exemple de l'opcode poscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 poscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

poscil3

Crédits

Auteur : Gabriel Maldonado
Italie
1998

Exemple écrit par Kevin Conder.

Novembre 2002. Ajout d'une note sur les changements dans la version 4.22 de Csound, merci à Rasmus Ekman.

Nouveau dans la version 3.52 de Csound

poscil3

poscil3 — Oscillateur haute précision avec interpolation cubique.

Description

Oscillateur haute précision avec interpolation cubique.

Syntax

```
ares poscil3 kamp, kcps, ifn [, iphs]
```

```
kres poscil3 kamp, kcps, ifn [, iphs]
```

Initialisation

ifn -- numéro de la table de fonction

iphs (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1)

Exécution

ares -- signal de sortie

kamp -- amplitude du signal de sortie.

kcps -- fréquence du signal de sortie en cycles par seconde.

poscil3 fonctionne comme *poscil*, mais il utilise l'interpolation cubique.

Exemples

Voici un exemple de l'opcode poscil3. Il utilise le fichier *poscil3.csd* [examples/poscil3.csd].

Exemple 373. Exemple de l'opcode poscil3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil3.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 poscil3 kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode poscil3, qui utilise une table remplie à partir d'un fichier son. Il utilise le fichier *poscil3-file.csd* [examples/poscil3-file.csd].

Exemple 374. Un autre exemple de l'opcode poscil3.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil3.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 poscil3 kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

poscil

Crédits

Auteurs : John ffitch, Gabriel Maldonado
Italie

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.52 de Csound

pow

pow — Calcule l'élévation à la puissance d'un argument par l'autre argument.

Description

Calcule *xarg* élevé à la puissance *kpow* (ou *ipow*) et pondère le résultat par *inorm*.

Syntaxe

```
ares pow aarg, kpow [, inorm]
```

```
ires pow iarg, ipow [, inorm]
```

```
kres pow karg, kpow [, inorm]
```

Initialisation

inorm (facultatif, par défault=1) -- Le nombre qui divisera le résultat (1 par défaut). Particulèrement utile si l'on calcule des puissances de signaux de taux -a ou de taux -k, ce qui produit très souvent des échantillons hors intervalle.

Exécution

aarg, *iarg*, *karg* -- la base.

ipow, *kpow* -- l'exposant.

Note

Utiliser ^ avec précaution dans les instructions arithmétiques, car les régles de précédence peuvent ne pas être correctes. Nouveau dans la version 3.493 de Csound.

Exemples

Voici un exemple de l'opcode pow. Il utilise le fichier *pow.csd* [examples/pow.csd].

Exemple 375. Exemple de l'opcode pow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pow.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; This could also be expressed as: i1 = 2 ^ 12
  i1 pow 2, 12

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 4096.000
```

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

powershape

powershape — Waveshapes a signal by raising it to a variable exponent.

Description

The *powershape* opcode raises an input signal to a power with pre- and post-scaling of the signal so that the output will be in a predictable range. It also processes negative inputs in a symmetrical way to positive inputs, calculating a dynamic transfer function that is useful for waveshaping.

Syntax

```
aout powershape ain, kShapeAmount [, ifullscale]
```

Initialization

ifullscale -- optional parameter specifying the range of input values from -*ifullscale* to *ifullscale*. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

Performance

ain -- the input signal to be shaped.

aout -- the output signal.

kShapeAmount -- the amount of the shaping effect applied to the input; equal to the power that the input signal is raised.

The *powershape* opcode is very similar to the *pow* unit generators for calculating the mathematical "power of" operation. However, it introduces a couple of twists that can make it much more useful for waveshaping audio-rate signals. The *kShapeAmount* parameter is the exponent to which the input signal is raised.

To avoid discontinuities, the *powershape* opcode treats all input values as positive (by taking their absolute value) but preserves their original sign in the output signal. This allows for smooth shaping of any input signal while varying the exponent over any range. (*powershape* also (hopefully) deals intelligently with discontinuities that could arise when the exponent and input are both zero. Note though that negative exponents will usually cause the signal to exceed the maximum amplitude specified by the ifullscale parameter and should normally be avoided).

The other adaptation involves the *ifullscale* parameter. The input signal is divided by *ifullscale* before being raised to *kShapeAmount* and then multiplied by *ifullscale* before being output. This normalizes the input signal to the interval [-1,1], guaranteeing that the output (before final scaling) will also be within this range for positive shaping amounts and providing a smoothly evolving transfer function while varying the amount of shaping. Values of *kShapeAmount* between (0,1) will make the signal more "convex" while values greater than 1 will make it more "concave". A value of exactly 1.0 will produce no change in the input signal.

See Also

pow, *powoftwo*

Examples

Here is an example of the powershape opcode. It uses the file *powershape.csd* [examples/powershape.csd].

Exemple 376. Example of the powershape opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
        imaxamp    =           10000
        kshapeamt  line        p5, p3, p6
        aosc       oscili      1.0, cpspch(p4), 1
        aout       powershape  aosc, kshapeamt
        adeclick   linseg      0.0, 0.01, 1.0, p3 - 0.06, 1.0, 0.05, 0.0

                     out         aout * adeclick * imaxamp
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 1    7.00  0.000001 0.8
i1 + 0.5  7.02  0.01     1.0
i1 + .    7.05  0.5      1.0
i1 + .    7.07  4.0      1.0
i1 + .    7.09  1.0      10.0
i1 + 2    7.06  1.0      25.0

e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Anthony Kozar
January 2008

New in Csound version 5.08

powoftwo

powoftwo — Calcule une puissance de deux.

Description

Calcule une puissance de deux.

Syntaxe

```
powoftwo(x)  (argument au taux d'initialisation ou de contrôle seulement)
```

Exécution

La fonction *powoftwo*() retourne 2^x et accepte comme argument des nombres positifs et négatifs. L'intervalle des valeurs autorisées dans *powoftwo*() va de -5 à +5 permettant une précision plus fine qu'un cent dans un intervalle de dix octaves. Pour un intervalle de valeurs plus grand, utiliser l'opcode plus lent *pow*.

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

Exemples

Voici un exemple de l'opcode powoftwo. Il utilise le fichier *powoftwo.csd* [examples/powoftwo.csd].

Exemple 377. Exemple de l'opcode powoftwo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o powoftwo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = powoftwo(12)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
```

```
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 4096.000
```

Voir Aussi

logbtwo, *pow*

Crédits

Auteur : Gabriel Maldonado
Italie
Juin 1998

Auteur : John ffitch
Université de Bath, Codemist, Ltd.
Bath, UK
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

prealloc

prealloc — Creates space for instruments but does not run them.

Description

Creates space for instruments but does not run them.

Syntax

```
prealloc insnum, icount
```

```
prealloc "insname", icount
```

Initialization

insnum -- instrument number

icount -- number of instrument allocations

« insname » -- A string (in double-quotes) representing a named instrument.

Performance

All instances of *prealloc* must be defined in the header section, not in the instrument body.

Examples

Here is an example of the prealloc opcode. It uses the file *prealloc.csd* [examples/prealloc.csd].

Exemple 378. Example of the prealloc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prealloc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Pre-allocate memory for five instances of Instrument #1.
prealloc 1, 5

; Instrument #1
```

```
instr 1
  ; Generate a waveform, get the cycles per second from the 4th p-field.
  a1 oscil 6500, p4, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play five instances of Instrument #1 for one second.
; Note that 4th p-field contains cycles per second.
i 1 0 1 220
i 1 0 1 440
i 1 0 1 880
i 1 0 1 1320
i 1 0 1 1760
e


</CsScore>
</CsoundSynthesizer>
```

See Also

cpuprc, *maxalloc*

Credits

Author: Gabriel Maldonado
Italy
July 1999

Example written by Kevin Conder.

New in Csound version 3.57

prepiano

prepiano — Crée un son similaire à celui d'une corde de piano préparé à la manière Cage.

Description

La sortie audio est un son similaire à celui d'une corde de piano préparé avec des gommes et des pièces de monnaie. La méthode utilise un modèle physique développé pour la résolution des équations différentielles partielles.

Syntaxe

```
ares prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]


al,ar prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]
```

Initialisation

ifreq -- la fréquence de base de la corde.

iNS -- le nombre de cordes impliquées. Dans un vrai piano on trouve 1, 2 ou 3 cordes dans les différentes plages de fréquence.

iD -- l'importance du désaccord de chaque corde, hormis la première, par rapport à la fréquence principale ; mesuré en cents.

iK -- paramètre de raideur, sans dimension.

iT30 -- durée de chute de 30 db en secondes.

ib -- paramètre de perte en haute-fréquence (à garder petit).

imass -- la masse du marteau.

ifreq -- la fréquence de vibration naturelle du marteau.

iinit -- la position initiale du marteau.

ipos -- position de la frappe sur la corde.

ivel -- vélocité normalisée de la frappe.

isfreq -- fréquence de balayage du point de lecture.

isspread -- dispersion de la fréquence de balayage.

irattles -- numéro de la table donnant les positions de la ou des pièces de monnaie.

irubbers -- numéro de la table donnant les positions de la ou des gommes.

Les tables des pièces de monnaie et des gommes sont des collections de quatre valeurs précédées par un

compte. Dans le cas d'une pièce de monnaie, les quatre valeurs sont la position, le rapport de densité entre la pièce de monnaie et la corde, la fréquence de la pièce de monnaie et sa longueur verticale. Pour la gomme, les quatre valeurs sont la position, le rapport de densité entre la gomme et la corde, la fréquence de la gomme et le paramètre de perte.

Exécution

Une note est jouée sur une corde de piano avec les arguments suivants.

kbcL -- Condition aux limites à l'extrémité gauche de la corde (1 fixée, 2 pivotante, 3 libre).

kbcR -- Condition aux limites à l'extrémité droite de la corde (1 fixée, 2 pivotante, 3 libre).

Il faut noter que le changement des conditions au limites durant l'exécution peut produire des bruits parasites et que cette possibilité n'est fournie qu'à titre expérimental.

Exemples

Voici en exemple de l'opcode prepiano. Il utilise le fichier *prepiano.csd* [examples/prepiano.csd].

Exemple 379. Exemple de l'opcode prepiano.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prepiano.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2;

; Instrument #1.
instr 1
;;          fund NS detune stiffness decay loss (bndry) (hammer) scan prep
aa,ab prepiano 60, 3, 10, p4, 3, 0.002, 2, 2, 1, 5000, -0.01, p5, p6, 0, 0.1, 1, 2
      outs aa*.75, ab*.75
endin
</CsInstruments>
<CsScore>
f1 0 8 2 1 0.6 10 100 0.001 ;; 1 rattle
f2 0 8 2 1 0.7 50 500 1000  ;; 1 rubber
i1 0.0 0.5 1 0.09 20
i1 0.5 .  -1 0.09 40        ;; 1 -> skip initialisation
i1 1.0 .  -1 0.09 60
i1 1.5 .  -1 0.09 80
i1 2.0 1.8  -1 0.09 100
e
</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Stefan Bilbao
Université d'Edimbourg, UK
Auteur : John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 5.05 de Csound

print

print — Displays the values init (i-rate) variables.

Description

These units will print orchestra init-values.

Syntax

```
print iarg [, iarg1] [, iarg2] [...]
```

Initialization

iarg, iarg2, ... -- i-rate arguments.

Performance

print -- print the current value of the i-time arguments (or expressions) *iarg* at every i-pass through the instrument.

Note

The *print* opcode will truncate decimal places and may not show the complete value. Csound's precision depends on whether it is the floats (32-bit) or double (64-bit) *version*, since most internal calculations use one of these formats. If you need more resolution in the console output, you can try *printf*.

Examples

Here is an example of the print opcode. It uses the file *print.csd* [examples/print.csd].

Exemple 380. Example of the print opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o print.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Print the fourth p-field.
  print p4
endin


</CsInstruments>
<CsScore>

; p4 = value to be printed.
; Play Instrument #1 for one second, p4 = 50.375.
i 1 0 1 50.375
; Play Instrument #1 for one second, p4 = 300.
i 1 1 1 300
; Play Instrument #1 for one second, p4 = -999.
i 1 2 1 -999
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
instr 1:  p4 = 50.375
instr 1:  p4 = 300.000
instr 1:  p4 = -999.000
```

See Also

dispfft, *display*, *printk*, *printk2*, *printks* , *printf* and *prints*

Credits

Example written by Kevin Conder.

Comments about the *inprds* parameter contributed by Rasmus Ekman.

printf

printf — printf-style formatted output

Description

printf and **printf_i** write formatted output, similarly to the C function printf(). **printf_i** runs at i-time only, while **printf** runs both at initialization and performance time.

Syntax

```
printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
```

```
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]
```

Initialization

Sfmt -- format string, has the same format as in printf() and other similar C functions, except length modifiers (l, ll, h, etc.) are not supported. The following conversion specifiers are allowed:

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

iarg1, iarg2, ... -- input arguments (max. 30) for format. Integer formats like %d round the input values to the nearest integer.

Performance

itrig -- if greater than zero the opcode performs the printing; otherwise it is an null operation.

ktrig -- if greater than zero and different from the value on the previous control cycle the opcode performs the requested printing. Initially this previous value is taken as zero.

xarg1, xarg2, ... -- input arguments (max. 30) for format. Integer formats like %d round the input values to the nearest integer. Note that only k-rate and i-rate arguments are valid (no a-rate printing)

Example

Credits

Author: Istvan Varga
2005

printk

printk — Prints one k-rate value at specified intervals.

Description

Prints one k-rate value at specified intervals.

Syntax

```
printk itime, kval [, ispace]
```

Initialization

itime -- time in seconds between printings.

ispace (optional, default=0) -- number of spaces to insert before printing. (default: 0, max: 130)

Performance

kval -- The k-rate values to be printed.

printk prints one k-rate value on every k-cycle, every second or at intervals specified. First the instrument number is printed, then the absolute time in seconds, then a specified number of spaces, then the *kval* value. The variable number of spaces enables different values to be spaced out across the screen - so they are easier to view.

This opcode can be run on every k-cycle it is run in the instrument. To every accomplish this, set *itime* to 0.

When *itime* is not 0, the opcode print on the first k-cycle it is called, and subsequently when every *itime* period has elapsed. The time cycles start from the time the opcode is initialized - typically the initialization of the instrument.

Examples

Here is an example of the printk opcode. It uses the file *printk.csd* [examples/printk.csd].

Exemple 381. Example of the printk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
  ; Change a value linearly from 0 to 100,
  ; over the period defined by p3.
  kval line 0, p3, 100

  ; Print the value of kval, once per second.
  printk 1, kval
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
 i   1 time     0.00002:     0.00000
 i   1 time     1.00002:    20.01084
 i   1 time     2.00002:    40.02999
 i   1 time     3.00002:    60.04914
 i   1 time     4.00002:    79.93327
```

See Also

printk2 and *printks*

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

Thanks goes to Luis Jure for pointing out a mistake wit the *itime* parameter.

printk2

printk2 — Prints a new value every time a control variable changes.

Description

Prints a new value every time a control variable changes.

Syntax

```
printk2 kvar [, inumspaces]
```

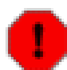
Initialization

inumspaces (optional, default=0) -- number of space characters printed before the value of *kvar*

Performance

kvar -- signal to be printed

Derived from Robin Whittle's *printk*, prints a new value of *kvar* each time *kvar* changes. Useful for monitoring MIDI control changes when using sliders.



Avertissement

WARNING! Don't use this opcode with normal, continuously variant k-signals, because it can hang the computer, as the rate of printing is too fast.

Examples

Here is an example of the printk2 opcode. It uses the file *printk2.csd* [examples/printk2.csd].

Exemple 382. Example of the printk2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printk2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Change a value linearly from 0 to 10,
  ; over the period defined by p3.
  kval1 line 0, p3, 10

  ; If kval1 is greater than or equal to 5,
  ; then kval=2, else kval=1.
  kval2 = (kval1 >= 5 ? 2 : 1)

  ; Print the value of kval2 when it changes.
  printk2 kval2
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include a line like this:

```
 i1     1.00000
 i1     2.00000
```

See Also

printk and *printks*

Credits

Author: Gabriel Maldonado
Italy
1998

Example written by Kevin Conder.

New in Csound version 3.48

printks

printks — Prints at k-rate using a printf() style syntax.

Description

Prints at k-rate using a printf() style syntax.

Syntax

```
printks "string", itime [, kval1] [, kval2] [...]
```

Initialization

"string" -- the text string to be printed. Can be up to 8192 characters and must be in double quotes.

itime -- time in seconds between printings.

Performance

kval1, kval2, ... (optional) -- The k-rate values to be printed. These are specified in *« string »* with the standard C value specifier (%f, %d, etc.) in the order given.

In Csound version 4.23, you can use as many *kval* variables as you like. In versions prior to 4.23, you must specify 4 and only 4 kvals (using 0 for unused kvals).

printks prints numbers and text which can be i-time or k-rate values. *printks* is highly flexible, and if used together with cursor positioning codes, could be used to write specific values to locations in the screen as the Csound processing proceeds.

A special mode of operation allows this *printks* to convert *kval1* input parameter into a 0 to 255 value and to use it as the first character to be printed. This enables a Csound program to send arbitrary characters to the console. To achieve this, make the first character of the string a # and then, if desired continue with normal text and format specifiers.

This opcode can be run on every k-cycle it is run in the instrument. To every accomplish this, set *itime* to 0.

When *itime* is not 0, the opcode print on the first k-cycle it is called, and subsequently when every *itime* period has elapsed. The time cycles start from the time the opcode is initialized - typically the initialization of the instrument.

Print Output Formatting

All standard C language printf() control characters may be used. For example, if *kval1* = 153.26789 then some common formatting options are:

1. %f prints with full precision: 153.26789
2. %5.2f prints: 153.26
3. %d prints integers-only: 153

4. %c treats *kval1* as an ascii character code.

In addition to all the printf() codes, printks supports these useful character codes:

printks Code	**Character Code**
\\r, \\R, %r, or %R	return character (\r)
\\n, \\N, %n, %N	newline character (\n)
\\t, \\T, %t, or %T	tab character (\t)
%!	semicolon character (;) This was needed because a « ; » is interpreted as an comment.
^	escape character (0x1B)
^ ^	caret character (^)
~	ESC[(escape+[is the escape sequence for ANSI consoles)
~~	tilde (~)

For more information about printf() formatting, consult any C language documentation.

Note

Prior to version 4.23, only the %f format code was supported.

Examples

Here is an example of the printks opcode. It uses the file *printks.csd* [examples/printks.csd].

Exemple 383. Example of the printks opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printks.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
  ; Change a value linearly from 0 to 100,
  ; over the period defined by p3.
  kup line 0, p3, 100
  ; Change a value linearly from 30 to 10,
  ; over the period defined by p3.
  kdown line 30, p3, 10
```

```
  ; Print the value of kup and kdown, once per second.
  printks "kup = %f, kdown = %f\\n", 1, kup, kdown
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
kup = 0.000000, kdown = 30.000000
kup = 20.010843, kdown = 25.962524
kup = 40.029991, kdown = 21.925049
kup = 60.049141, kdown = 17.887573
kup = 79.933266, kdown = 13.872493
```

See Also

printk2 and *printk*

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

Thanks goes to Luis Jure for pointing out a mistake with the *itime* parameter.

Thanks to Matt Ingalls, updated the documentation for version 4.23.

prints

prints — Prints at init-time using a printf() style syntax.

Description

Prints at init-time using a printf() style syntax.

Syntax

```
prints "string" [, kval1] [, kval2] [...]
```

Initialization

"string" -- the text string to be printed. Can be up to 8192 characters and must be in double quotes.

Performance

kval1, kval2, ... (optional) -- The k-rate values to be printed. These are specified in *« string »* with the standard C value specifier (%f, %d, etc.) in the order given. Use 0 for those which are not used.

prints is similar to the *printks* opcode except it operates at init-time instead of k-rate. For more information about output formatting, please look at *printks's documentation*.

Examples

Here is an example of the prints opcode. It uses the file *prints.csd* [examples/prints.csd].

Exemple 384. Example of the prints opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prints.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Init-time print.
  prints "%2.3f\\t%!%!%!%!%!%!semicolons!\\n", 1234.56789
endin
```

```
</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play instrument #1.
i 1 0 0.004


</CsScore>
</CsoundSynthesizer>
```

Its output should include a line like this:

```
1234.568        ;;;;;;semicolons!
```

See Also

printks

Credits

Author: Matt Ingalls
January 2003

product

product — Multiplie n'importe quel nombre de signaux de taux-a.

Description

Multiplie n'importe quel nombre de signaux de taux-a.

Syntaxe

```
ares product asig1, asig2 [, asig3] [...]
```

Exécution

asig1, asig2, asig3, ... -- signaux de taux-a à multiplier.

Crédits

Auteur : Gabriel Maldonado
Italie
Avril 1999

Nouveau dans la version 3.54 de Csound

pset

pset — Defines and initializes numeric arrays at orchestra load time.

Description

Defines and initializes numeric arrays at orchestra load time.

Syntax

```
pset icon1 [, icon2] [...]
```

Initialization

icon1, icon2, ... -- preset values for a MIDI instrument

pset (optional) defines and initializes numeric arrays at orchestra load time. It may be used as an orchestra header statement (i.e. instrument 0) or within an instrument. When defined within an instrument, it is not part of its i-time or performance operation, and only one statement is allowed per instrument. These values are available as i-time defaults. When an instrument is triggered from MIDI it only gets p1 and p2 from the event, and p3, p4, etc. will receive the actual preset values.

Examples

The example below illustrates *pset* as used within an instrument.

```
instr 1
  pset 0,0,3,4,5,6  ; pfield substitutes
  a1 oscil 10000, 440, p6
```

See Also

strset

ptrack

ptrack — Tracks the pitch of a signal.

Description

ptrack takes an input signal, splits it into *ihopsize* blocks and using a STFT method, extracts an estimated pitch for its fundamental frequency as well as estimating the total amplitude of the signal in dB, relative to full-scale (0dB). The method implies an analysis window size of 2**ihopsize* samples (overlaping by 1/2 window), which has to be a power-of-two, between 128 and 8192 (hopsizes between 64 and 4096). Smaller windows will give better time precision, but worse frequency accuracy (esp. in low fundamentals).This opcode is based on an original algorithm by M. Puckette.

Syntax

```
kcps, kamp ptrack asig, ihopsize[,ipeaks]
```

Initialization

ihopsize -- size of the analysis 'hop', in samples, required to be power-of-two (min 64, max 4096). This is the period between measurements.

ipeaks, *ihi* -- number of spectral peaks to use in the analysis, defaults to 20 (optional)

Performance

kcps -- estimated pitch in Hz.

kamp -- estimated amplitude in dB relative to full-scale (0dB) (ie. always <= 0).

ptrack analyzes the input signal, *asig*, to give a pitch/amplitude pair of outputs, for the fundamental of a monophonic signal. The output is updated every *sr/ihopsize* seconds.

Examples

Here is an example of the ptrack opcode. This example uses the files *ptrack.csd* [examples/ptrack.csd].

Exemple 385. Example of the ptrack opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No display
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>
sr= 44100
ksmps = 16
nchnls= 1

;Example by Victor Lazzarini 2007
```

```
instr 1
  a1 inch 1               ; take an input signal
  kf,ka ptrack a1, 512    ; pitch track with winsize=1024
  kcps port kf, 0.01      ; smooth freq
  kamp port ka, 0.01      ; smooth amp

  ; drive an oscillator
  aout oscili ampdb(kamp)*0dbfs, kcps, 1

  out  aout
endin

</CsInstruments>
<CsScore>
i1 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Victor Lazzarini;
NUI, Maynooth.
Maynooth, Ireland
March, 2007

New in Csound version 5.05

puts

puts — Print a string constant or variable

Description

puts prints a string with an optional newline at the end whenever the trigger signal is positive and changes.

Syntax

```
puts Sstr, ktrig[, inonl]
```

Initialization

Sstr -- string to be printed

inonl (optional, defaults to 0) -- if non-zero, disables the default printing of a newline character at the end of the string

Performance

ktrig -- trigger signal, should be valid at i-time. The string is printed at initialization time if ktrig is positive, and at performance time whenever ktrig is both positive and different from the previous value. Use a constant value of 1 to print once at note initialization.

Credits

Author: Istvan Varga
2005

push

push — Pushes a value into the global stack.

Description

Pushes a value into the global stack.

Syntax

```
push  xval1, [xval2, ... , xval31]

push  ival1, [ival2, ... , ival31]
```

Initialization

ival1 ... ival31 - values to be pushed into the stack.

Performance

xval1 ... xval31 - values to be pushed into the stack.

The given values are pushed into the global stack as a bundle. The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

push and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Use of any combination of i, k, a, and S types is allowed. Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push_f* only pushes a "reference" to the f-signal, and then the corresponding *pop_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push_f* before *pop_f* is called is not recommended, and if the instrument instance owning the variable that was passed by push_f is deactivated before pop_f is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

See also

stack, *pop*, *pop_f* and *push_f*.

Credits

By: Istvan Varga.

2006

push_f

push_f — Pushes an f-sig frame into the global stack.

Description

Pushes an f-sig frame into the global stack.

Syntax

```
push_f  fsig
```

Performance

fsig - f-signal to be pushed into the stack.

The values are pushed into the global stack. The global stack works in LIFO order: after multiple *push_f* calls, *pop_f* should be used in reverse order.

push/pop for a, k, i, and S types copy data by value. By contrast, *push_f* only pushes a "reference" to the f-signal, and then the corresponding *pop_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push_f* before *pop_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push_f* is deactivated before *pop_f* is called, undefined behavior may occur.

pop_f and *push_f* can only take a single argument, and the data is passed both at init and performance time.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

See also

stack, *push*, *pop* and *pop_f*.

Credits

By: Istvan Varga.

2006

pvadd

pvadd — Reads from a *pvoc* file and uses the data to perform additive synthesis.

Description

pvadd reads from a *pvoc* file and uses the data to perform additive synthesis using an internal array of interpolating oscillators. The user supplies the wave table (usually one period of a sine wave), and can choose which analysis bins will be used in the re-synthesis.

Syntax

```
ares pvadd ktimpnt, kfmod, ifilcod, ifn, ibins [, ibinoffset] \
      [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]
```

Initialization

ifilcod -- integer or character-string denoting a control-file derived from *pvanal* analysis of an audio signal. An integer denotes the suffix of a file *pvoc.m*; a character-string (in double quotes) gives a filename, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in the one given by the environment variable *SADIR* (if defined). *pvoc* control files contain data organized for fft resynthesis. Memory usage depends on the size of the files involved, which are read and held entirely in memory during computation but are shared by multiple calls (see also *lpread*).

ifn -- table number of a stored function containing a sine wave.

ibins -- number of bins that will be used in the resynthesis (each bin counts as one oscillator in the re-synthesis)

ibinoffset (optional) -- is the first bin used (it is optional and defaults to 0).

ibinincr (optional) -- sets an increment by which *pvadd* counts up from *ibinoffset* for *ibins* components in the re-synthesis (see below for a further explanation).

iextractmode (optional) -- determines if spectral extraction will be carried out and if so whether components that have changes in frequency below *ifreqlim* or above *ifreqlim* will be discarded. A value for *iextractmode* of 1 will cause *pvadd* to synthesize only those components where the frequency difference between analysis frames is greater than *ifreqlim*. A value of 2 for *iextractmode* will cause *pvadd* to synthesize only those components where the frequency difference between frames is less than *ifreqlim*. The default values for *iextractmode* and *ifreqlim* are 0, in which case a simple resynthesis will be done. See examples below.

igatefn (optional) -- is the number of a stored function which will be applied to the amplitudes of the analysis bins before resynthesis takes place. If *igatefn* is greater than 0 the amplitudes of each bin will be scaled by *igatefn* through a simple mapping process. First, the amplitudes of all of the bins in all of the frames in the entire analysis file are compared to determine the maximum amplitude value. This value is then used create normalized amplitudes as indeces into the stored function *igatefn*. The maximum amplitude will map to the last point in the function. An amplitude of 0 will map to the first point in the function. Values between 0 and 1 will map accordingly to points along the function table.This will be made clearer in the examples below.

Performance

ktimpnt and *kfmod* are used in the same way as in *pvoc*.

Examples

```
ktime line  0, p3, p3
```

asig *pvadd* ktime, 1, « oboe.pvoc », 1, 100, 2

In the above, *ibins* is 100 and *ibinoffset* is 2. Using these settings the resynthesis will contain 100 components beginning with bin #2 (bins are counted starting with 0). That is, resynthesis will be done using bins 2-101 inclusive. It is usually a good idea to begin with bin 1 or 2 since the 0th and often 1st bin have data that is neither necessary nor even helpful for creating good clean resynthesis.

```
ktime line  0, p3, p3
asig  pvadd ktime, 1, « oboe.pvoc », 1, 100, 2, 2
```

The above is the same as the previous example with the addition of the value 2 used for the optional *ibinincr* argument. This result will still result in 100 components in the resynthesis, but *pvadd* will count through the bins by 2 instead of by 1. It will use bins 2, 4, 6, 8, 10, and so on. For *ibins*=10, *ibinoffset*=10, and *ibinincr*=10, *pvadd* would use bins 10, 20, 30, 40, up to and including 100.

Below is an example using spectral extraction. In this example *iextractmode* is one and *ifreqlim* is 9. This will cause pvadd to synthesize only those bins where the frequency deviation, averaged over 6 frames, is greater than 9.

```
ktime line  0, p3, p3
asig  pvadd ktime, 1,  « oboe.pvoc », 1, 100, 2, 2, 1, 9
```

If *iextractmode* were 2 in the above, then only those bins with an average frequency deviation of less than 9 would be synthesized. If tuned correctly, this technique can be used to separate the pitched parts of the spectrum from the noisy parts. In practice this depends greatly on the type of sound, the quality of the recording and digitization, and also on the analysis window size and frame increment.

Next is an example using amplitude gating. The last 2 in the argument list stands for f2 in the score.

```
asig  pvadd ktime, 1,  « oboe.pvoc », 1, 100, 2, 2, 0, 0, 2
```

Suppose the score for the above were to contain:

```
f2 0 512 7 0 256 1 256 1
```

Then those bins with amplitudes of 50% of the maximum or greater would be left unchanged, while those with amplitudes less than 50% of the maximum would be scaled down. In this case the lower the amplitude the more severe the scaling down would be. But suppose the score contains:

```
f2 0 512 5 1 512 .001
```

In this case lower amplitudes will be left unchanged and greater ones will be scaled down, turning the sound « upside-down » in terms of the amplitude spectrum! Functions can be arbitrarily complex. Just remember that the normalized amplitude values of the analysis are themselves the indeces into the function.

Finally, both spectral extraction and amplitude gating can be used together. The example below will synthesize only those components that with a frequency deviation of less than 5Hz per frame and it will scale the amplitudes according to F2.

```
asig  pvadd ktime, 1,  « oboe.pvoc », 1, 100, 1, 1, 2, 5, 2
```

USEFUL HINTS

By using several *pvadd* units together, one can gradually fade in different parts of the resynthesis, creating various « filtering » effects. The author uses *pvadd* to synthesis one bin at a time to have control over each separate component of the re-synthesis.

If any combination of *ibins*, *ibinoffset*, and *ibinincr*, creates a situation where *pvadd* is asked to used a bin number greater than the number of bins in the analysis, it will just use all of the available bins, and give no complaint. So to use every bin just make *ibins* a big number (ie. 2000).

Expect to have to scale up the amplitudes by factors of 10-100, by the way.

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48, additional arguments version 3.56

pvbufread

pvbufread — Reads from a phase vocoder analysis file and makes the retrieved data available.

Description

pvbufread reads from a *pvoc* file and makes the retrieved data available to any following *pvinterp* and *pvcross* units that appear in an instrument before a subsequent *pvbufread* (just as *lpread* and *lpreson* work together). The data is passed internally and the unit has no output of its own.

Syntax

```
pvbufread ktimpnt, ifile
```

Initialization

ifile -- the *pvoc* number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

Performance

ktimpnt -- the passage of time, in seconds, through this file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

Examples

The example below shows an example using *pvbufread* with *pvinterp* to interpolate between the sound of an oboe and the sound of a clarinet. The value of *kinterp* returned by a linseg is used to determine the timing of the transitions between the two sounds. The interpolation of frequencies and amplitudes are controlled by the same factor in this example, but for other effects it might be interesting to not have them synchronized in this way. In this example the sound will begin as a clarinet, transform into the oboe and then return again to the clarinet sound. The value of *kfreqscale2* is 1.065 because the oboe in this case is a semitone higher in pitch than the clarinet and this brings them approximately to the same pitch. The value of *kampscale2* is .75 because the analyzed clarinet was somewhat louder than the analyzed oboe. The setting of these two parameters make the transition quite smooth in this case, but such adjustments are by no means necessary or even advocated.

```
ktime1  line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2  line      0, p3, 4.5 ; used as index in the  "clar.pvoc" file
kinterp linseg    1, p3*.15, 1, p3*.35, 0, p3*.25, 0, p3*.15, 1, p3*.1, 1
        pvbufread ktime1, "oboe.pvoc"
apv     pvinterp  ktime2,1,"clar.pvoc",1,1.065,1,.75,1-kinterp,1-kinterp
```

Below is an example using *pvbufread* with *pvcross*. In this example the amplitudes used in the resynthesis gradually change from those of the oboe to those of the clarinet. The frequencies, of course, remain those of the clarinet throughout the process since *pvcross* does not use the frequency data from the file read by *pvbufread*.

```
ktime1  line     0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2  line     0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross  expon      .001, p3, 1
        pvbufread ktime1, "oboe.pvoc"
apv     pvcross   ktime2, 1, "clar.pvoc", 1-kcross, kcross
```

See Also

pvcross, *pvinterp*, *pvread*, *tableseg*, *tablexseg*

Credits

Author: Richard Karpen
Seattle, WA USA
1997

pvcross

pvcross — Applies the amplitudes from one phase vocoder analysis file to the data from a second file.

Description

pvcross applies the amplitudes from one phase vocoder analysis file to the data from a second file and then performs the resynthesis. The data is passed, as described above, from a previously called *pvbufread* unit. The two k-rate amplitude arguments are used to scale the amplitudes of each files separately before they are added together and used in the resynthesis (see below for further explanation). The frequencies of the first file are not used at all in this process. This unit simply allows for cross-synthesis through the application of the amplitudes of the spectra of one signal to the frequencies of a second signal. Unlike *pvinterp*, *pvcross* does allow for the use of the *ispecwp* as in *pvoc* and *vpvoc*.

Syntax

```
ares pvcross ktimpnt, kfmod, ifile, kampscale1, kampscale2 [, ispecwp]
```

Initialization

ifile -- the *pvoc* number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

ispecwp (optional, default=0) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

Performance

ktimpnt -- the passage of time, in seconds, through this file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

kfmod -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

kampscale1, kampscale2 -- used to scale the amplitudes stored in each frame of the phase vocoder analysis file. *kampscale1* scale the amplitudes of the data from the file read by the previously called *pvbufread*. *kampscale2* scale the amplitudes of the file named by *ifile*.

By using these arguments, it is possible to adjust these values before applying the interpolation. For example, if file1 is much louder than file2, it might be desirable to scale down the amplitudes of file1 or scale up those of file2 before interpolating. Likewise one can adjust the frequencies of each to bring them more in accord with one another (or just the opposite, of course!) before the interpolation is performed.

Examples

Below is an example using *pvbufread* with *pvcross*. In this example the amplitudes used in the resynthesis gradually change from those of the oboe to those of the clarinet. The frequencies, of course, remain those of the clarinet throughout the process since *pvcross* does not use the frequency data from the file read by *pvbufread*.

```
ktime1  line     0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2  line     0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross  expon     .001, p3, 1
        pvbufread ktime1, "oboe.pvoc"
apv     pvcross   ktime2, 1, "clar.pvoc", 1-kcross, kcross
```

See Also

pvbufread, *pvinterp*, *pvread*, *tableseg*, *tablexseg*

Credits

Author: Richard Karpen
Seattle, Wash
1997

New in version 3.44

pvinterp

pvinterp — Interpolates between the amplitudes and frequencies of two phase vocoder analysis files.

Description

pvinterp interpolates between the amplitudes and frequencies, on a bin by bin basis, of two phase vocoder analysis files (one from a previously called *pvbufread* unit and the other from within its own argument list), allowing for user defined transitions between analyzed sounds. It also allows for general scaling of the amplitudes and frequencies of each file separately before the interpolated values are calculated and sent to the resynthesis routines. The *kfmod* argument in *pvinterp* performs its frequency scaling on the frequency values after their derivation from the separate scaling and subsequent interpolation is performed so that this acts as an overall scaling value of the new frequency components.

Syntax

```
ares pvinterp ktimpnt, kfmod, ifile, kfreqscale1, kfreqscale2, \
      kampscale1, kampscale2, kfreqinterp, kampinterp
```

Initialization

ifile -- the *pvoc* number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

Performance

ktimpnt -- the passage of time, in seconds, through this file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

kfmod -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

kfreqscale1, *kfreqscale2*, *kampscale1*, *kampscale2* -- used in *pvinterp* to scale the frequencies and amplitudes stored in each frame of the phase vocoder analysis file. *kfreqscale1* and *kampscale1* scale the frequencies and amplitudes of the data from the file read by the previously called *pvbufread* (this data is passed internally to the pvinterp unit). *kfreqscale2* and *kampscale2* scale the frequencies and amplitudes of the file named by *ifile* in the *pvinterp* argument list and read within the *pvinterp* unit.

By using these arguments, it is possible to adjust these values before applying the interpolation. For example, if file1 is much louder than file2, it might be desirable to scale down the amplitudes of file1 or scale up those of file2 before interpolating. Likewise one can adjust the frequencies of each to bring them more in accord with one another (or just the opposite, of course!) before the interpolation is performed.

kfreqinterp, *kampinterp* -- used in *pvinterp*, determine the interpolation distance between the values of one phase vocoder file and the values of a second file. When the value of *kfreqinterp* is 1, the frequency values will be entirely those from the first file (read by the *pvbufread*), post scaling by the *kfreqscale1* argument. When the value of *kfreqinterp* is 0 the frequency values will be those of the second file (read by the *pvinterp* unit itself), post scaling by kfreqscale2. When *kfreqinterp* is between 0 and 1 the frequency values will be calculated, on a bin, by bin basis, as the percentage between each pair of frequencies (in other words, *kfreqinterp*=.5 will cause the frequencies values to be half way between the values in the set of data from the first file and the set of data from the second file).

kampinterp works in the same way upon the amplitudes of the two files. Since these are k-rate arguments, the percentages can change over time making it possible to create many kinds of transitions between sounds.

Examples

The example below shows an example using *pvbufread* with *pvinterp* to interpolate between the sound of an oboe and the sound of a clarinet. The value of *kinterp* returned by a linseg is used to determine the timing of the transitions between the two sounds. The interpolation of frequencies and amplitudes are controlled by the same factor in this example, but for other effects it might be interesting to not have them synchronized in this way. In this example the sound will begin as a clarinet, transform into the oboe and then return again to the clarinet sound. The value of *kfreqscale2* is 1.065 because the oboe in this case is a semitone higher in pitch than the clarinet and this brings them approximately to the same pitch. The value of *kampscale2* is .75 because the analyzed clarinet was somewhat louder than the analyzed oboe. The setting of these two parameters make the transition quite smooth in this case, but such adjustments are by no means necessary or even advocated.

```
ktime1  line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2  line      0, p3, 4.5 ; used as index in the  "clar.pvoc" file
kinterp linseg    1, p3*.15, 1, p3*.35, 0, p3*.25, 0, p3*.15, 1, p3*.1, 1
        pvbufread ktime1, "oboe.pvoc"
apv     pvinterp  ktime2,1,"clar.pvoc",1,1.065,1,.75,1-kinterp,1-kinterp
```

See Also

pvbufread, *pvcross*, *pvread*, *tableseg*, *tablexseg*

Credits

Author: Richard Karpen
Seattle, Wash
1997

pvoc

pvoc — Implements signal reconstruction using an fft-based phase vocoder.

Description

Implements signal reconstruction using an fft-based phase vocoder.

Syntax

```
ares pvoc ktimpnt, kfmod, ifilcod [, ispecwp] [, iextractmode] \
      [, ifreqlim] [, igatefn]
```

Initialization

ifilcod -- integer or character-string denoting a control-file derived from analysis of an audio signal. An integer denotes the suffix of a file *pvoc.m*; a character-string (in double quotes) gives a filename, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in the one given by the environment variable *SADIR* (if defined). *pvoc* control contains breakpoint amplitude and frequency envelope values organized for fft resynthesis. Memory usage depends on the size of the files involved, which are read and held entirely in memory during computation but are shared by multiple calls (see also *lpread*).

ispecwp (optional) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

iextractmode (optional) -- determines if spectral extraction will be carried out and if so whether components that have changes in frequency below *ifreqlim* or above *ifreqlim* will be discarded. A value for *iextractmode* of 1 will cause *pvadd* to synthesize only those components where the frequency difference between analysis frames is greater than *ifreqlim*. A value of 2 for *iextractmode* will cause *pvadd* to synthesize only those components where the frequency difference between frames is less than *ifreqlim*. The default values for *iextractmode* and *ifreqlim* are 0, in which case a simple resynthesis will be done. See examples under *pvadd* for how to use spectral extraction.

igatefn (optional) -- the number of a stored function which will be applied to the amplitudes of the analysis bins before resynthesis takes place. If *igatefn* is greater than 0 the amplitudes of each bin will be scaled by *igatefn* through a simple mapping process. First, the amplitudes of all of the bins in all of the frames in the entire analysis file are compared to determine the maximum amplitude value. This value is then used create normalized amplitudes as indeces into the stored function *igatefn*. The maximum amplitude will map to the last point in the function. An amplitude of 0 will map to the first point in the function. Values between 0 and 1 will map accordingly to points along the function table. See examples under *pvadd* for how to use amplitude gating.

Performance

ktimpnt -- The passage of time, in seconds, through the analysis file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

kfmod -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

pvoc implements signal reconstruction using an fft-based phase vocoder. The control data stems from a

precomputed analysis file with a known frame rate.

This implementation of *pvoc* was orignally written by Dan Ellis. It is based in part on the system of Mark Dolson, but the pre-analysis concept is new. The spectral extraction and amplitude gating (new in Csound version 3.56) were added by Richard Karpen based on functions in SoundHack by Tom Erbe.

See Also

vpvoc, *PVANAL*.

Credits

Authors: Dan Ellis and Richard Karpen
Seattle, Wash
1997

pvread

pvread — Reads from a phase vocoder analysis file and returns the frequency and amplitude from a single analysis channel or bin.

Description

pvread reads from a *pvoc* file and returns the frequency and amplitude from a single analysis channel or bin. The returned values can be used anywhere else in the Csound instrument. For example, one can use them as arguments to an oscillator to synthesize a single component from an analyzed signal or a bank of pvreads can be used to resynthesize the analyzed sound using additive synthesis by passing the frequency and magnitude values to a bank of oscillators.

Syntax

```
kfreq, kamp pvread ktimpnt, ifile, ibin
```

Initialization

ifile -- the *pvoc* number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

ibin -- the number of the analysis channel from which to return frequency in Hz and magnitude.

Performance

kfreq, kamp -- outputs of the *pvread* unit. These values, retrieved from a phase vocoder analysis file, represent the values of frequency and amplitude from a single analysis channel specified in the ibin argument. Interpolation between analysis frames is performed at k-rate resolution and dependent of course upon the rate and direction of ktimpnt.

ktimpnt -- the passage of time, in seconds, through this file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

Examples

The example below shows the use *pvread* to synthesize a single component from a phase vocoder analysis file. It should be noted that the *kfreq* and *kamp* outputs can be used for any kind of synthesis, filtering, processing, and so on.

```
ktime         line    0, p3, 3
kfreq, kamp   pvread  ktime, "pvoc.file", 7 ; read
                                        ;data from 7th analysis bin.
asig          oscili  kamp, kfreq, 1  ; function 1
                                        ;is a stored sine
```

See Also

pvbufread, *pvcross*, *pvinterp*, *tableseg*, *tablexseg*

Credits

Author: Richard Karpen
Seattle, Wash
1997

New in version 3.44

pvsadsyn

pvsadsyn — Resynthesize using a fast oscillator-bank.

Description

Resynthesize using a fast oscillator-bank.

Syntax

```
ares pvsadsyn fsrc, inoscs, kfmod [, ibinoffset] [, ibinincr] [, iinit]
```

Initialization

inoscs -- The number of analysis bins to synthesise. Cannot be larger than the size of fsrc (see *pvsinfo*), e.g. as created by *pvsanal*. Processing time is directly proportional to inoscs.

ibinoffset (optional, default=0) -- The first (lowest) bin to resynthesise, counting from 0 (default = 0).

ibinincr (optional) -- Starting from bin ibinoffset, resynthesize bins ibinincr apart.

iinit (optional) -- Skip reinitialization. This is not currently implemented for any of these opcodes, and it remains to be seen if it is even practical.

Performance

kfmod -- Scale all frequencies by factor kfmod. 1.0 = no change, 2 = up one octave.

pvsadsyn is experimental, and implements the oscillator bank using a fast direct calculation method, rather than a lookup table. This takes advantage of the fact, empirically arrived at, that for the analysis rates generally used, (and presuming analysis using *pvsanal*, where frequencies in a bin change only slightly between frames) it is not necessary to interpolate frequencies between frames, only amplitudes. Accurate resynthesis is often contingent on the use of pvsanal with iwinsize = ifftsize*2.

This opcode is the most likely to change, or be much extended, according to feedback and advice from users. It is likely that a full interpolating table-based method will be added, via a further optional iarg. The parameter list to pvsadsyn mimics that for *pvadd*, but excludes spectral extraction.

Examples

```
; resynth the first 100 odd-numbered bins, with pitch scaling envelope.
kpch  linseg    1,p3/3,1,p3/3,1.5,p3/3,1
aout  pvsadsyn  fsrc, 100,kpch,1,2
```

See Also

pvsynth

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsanal

pvsanal — Generate an fsig from a mono audio source ain, using phase vocoder overlap-add analysis.

Description

Generate an fsig from a mono audio source ain, using phase vocoder overlap-add analysis.

Syntax

```
fsig pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
```

Initialization

ifftsize -- The FFT size in samples. Need not be a power of two (though these are especially efficient), but must be even. Odd numbers are rounded up internally. ifftsize determines the number of analysis bins in fsig, as ifftsize/2 + 1. For example, where ifftsize = 1024, fsig will contain 513 analysis bins, ordered linearly from the fundamental to Nyquist. The fundamental of analysis (which in principle gives the lowest resolvable frequency) is determined as sr/ifftsize. Thus, for the example just given and assuming sr = 44100, the fundamental of analysis is 43.07Hz. In practice, due to the phase-preserving nature of the phase vocoder, the frequency of any bin can deviate bilaterally, so that DC components are recorded. Given a strongly pitched signal, frequencies in adjacent bins can bunch very closely together, around partials in the source, and the lowest bins may even have negative frequencies.

As a rule, the only reason to use a non power-of-two value for ifftsize would be to match the known fundamental frequency of a strongly pitched source. Values with many small factors can be almost as efficient as power-of-two sizes; for example: 384, for a source pitched at around low A=110Hz.

ioverlap -- The distance in samples (« hop size ») between overlapping analysis frames. As a rule, this needs to be at least ifftsize/4, e.g. 256 for the example above. ioverlap determines the underlying analysis rate, as sr/ioverlap. ioverlap does not require to be a simple factor of ifftsize; for example a value of 160 would be legal. The choice of ioverlap may be dictated by the degree of pitch modification applied to the fsig, if any. As a rule of thumb, the more extreme the pitch shift, the higher the analysis rate needs to be, and hence the smaller the value for ioverlap. A higher analysis rate can also be advantageous with broadband transient sounds, such as drums (where a small analysis window gives less smearing, but more frequency-related errors).

Note that it is possible, and reasonable, to have distinct fsigs in an orchestra (even in the same instrument), running at different analysis rates. Interactions between such fsigs is currently unsupported, and the fsig assignment opcode does not allow copying between fsigs with different properties, even if the only difference is in ioverlap. However, this is not a closed issue, as it is possible in theory to achieve crude rate conversion (especially with regard to in-memory analysis files) in ways analogous to time-domain techniques.

iwinsize -- The size in samples of the analysis window filter (as set by iwintype). This must be at least ifftsize, and can usefully be larger. Though other proportions are permitted, it is recommended that iwinsize always be an integral multiple of ifftsize, e.g. 2048 for the example above. Internally, the analysis window (Hamming, von Hann) is multiplied by a sinc function, so that amplitudes are zero at the boundaries between frames. The larger analysis window size has been found to be especially important for oscillator bank resynthesis (e.g. using pvsadsyn), as it has the effect of increasing the frequency resolution of the analysis, and hence the accuracy of the resynthesis. As noted above, iwinsize determines the overall latency of the analysis/resynthesis system. In many cases, and especially in the absence of pitch modifications, it will be found that setting iwinsize=ifftsize works very well, and offers the lowest laten-

cy.

iwintype -- The shape of the analysis window. Currently only two choices are implemented:

- 0 = Hamming window
- 1 = von Hann window

Both are also supported by the PVOC-EX file format. The window type is stored as an internal attribute of the fsig, together with the other parameters (see *pvsinfo*). Other types may be implemented later on (e.g. the Kaiser window, also supported by PVOC-EX), though an obvious alternative is to enable windows to be defined via a function table. The main issue here is the constraint of f-tables to power-of-two sizes, so this method does not offer a complete solution. Most users will find the Hamming window meets all normal needs, and can be regarded as the default choice.

iformat -- (optional) The analysis format. Currently only one format is implemented by this opcode:

- 0 = amplitude + frequency

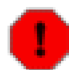
This is the classic phase vocoder format; easy to process, and a natural format for oscillator-bank resynthesis. It would be very easy (tempting, one might say) to treat an fsig frame not purely as a phase vocoder frame but as a generic additive synthesis frame. It is indeed possible to use an fsig this way, but it is important to bear in mind that the two are not, strictly speaking, directly equivalent.

Other important formats (supported by PVOC-EX) are:

- 1 = amplitude + phase
- 2 = complex (real + imaginary)

iformat is provided in case it proves useful later to add support for these other formats. Formats 0 and 1 are very closely related (as the phase is « wrapped » in both cases - it is a trivial matter to convert from one to the other), but the complex format might warrant a second explicit signal type (a « csig ») specifically for convolution-based processes, and other processes where the full complement of arithmetic operators may be useful.

iinit -- (optional) Skip reinitialization. This is not currently implemented for any of these opcodes, and it remains to be seen if it is even practical.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

```
ain   in                             ; live source
ffin   pvsanal  ain,1024,256,2048,0  ; analyze, using Hamming
ffout  pvsmaska ffin,1,0.75          ; apply eq from f-table
aout  pvsynth   ffout                ; and resynthesize
```

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsarp

pvsarp — Arpeggiate the spectral components of a streaming pv signal.

Description

This opcode arpeggiates spectral components, by amplifying one bin and attenuating all the others around it. Used with an LFO it will provide a spectral arpeggiator similar to Trevor Wishart's CDP program specarp.

Syntax

```
fsig pvsarp fsigin, kbin, kdepth, kgain
```

Performance

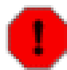
fsig -- output pv stream

fsigin -- input pv stream

kbin -- target bin, normalised 0 - 1 (0Hz - Nyquist).

kdepth -- depth of attenuation of surrounding bins

kgain -- gain boost applied to target bin



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Here is an example of the butterbp opcode. It uses the file *pvsarp.csd* [examples/pvsarp.csd]. The example shows a spectral arpeggiator working in the 220.5 - 2425.5 range (sr=44100). The LFO outputs a positive-only signal, so its ftable will be defined in the 0 - 1 range (a hanning window can be used, for instance).

Exemple 386. Example of the pvsarp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     ;;;-d      RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsarp.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 300
nchnls = 1

instr 1
asig  in                                ; get the signal in
idepth = p4

fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
kbin  oscili   0.1, 0.5, 1              ; ftable 1 in the 0-1 range
ftps  pvsarp   fsig, kbin+0.01, idepth, 2   ; arpeggiate it (range 220.5 - 2425.5)
atps  pvsynth  ftps                     ; synthesise it

       out atps
endin


</CsInstruments>
<CsScore>
; Table #1, a sine wave.
f 1 0 4096 10 1
i 1 0 10 0.5
i 1 + 10 0.9
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Victor Lazzarini
April 2005

New plugin in version 5

April 2005.

pvsbandp

pvsbandp — A band pass filter working in the spectral domain.

Description

Filter the pvoc frames, passing bins whose frequency is within a band, and with linear interpolation for transitional bands.

Syntax

```
fsig pvsbandp fsigin, xlowcut,
      xlowfull, xhighfull, xhighcut[, ktype]
```

Performance

fsig -- output pv stream

fsigin -- input pv stream.

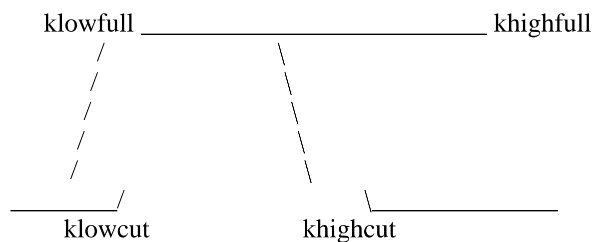
xlowcut, xlowfull, xhighfull, xhighcut -- define a trapezium shape for the band that is passed. The a-rate versions only apply to the sliding case.

ktype -- specifies the shape of the transitional band. If at the default value of zero the shape is as below, with linear transition in amplitude. Other values yield and exponential shape:

(1 - exp(r*type)) / (1 - exp(type))

This includes a linear dB shape when ktype is log(10) or about 2.30.

The opcode performs a band-pass filter with a spectral envelope shaped like



Examples

Exemple 387. Example

```
asig  in                                      ; get the signal in
```

```
fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvsbandp  fsig, 200, 400, 1000, 2000 ; filter 300-1500Hz
atps  pvsynth   ftps                     ; synthesise it
```

Credits

Author: John ffitch
December 2007

pvsbandr

pvsbandr — A band reject filter working in the spectral domain.

Description

Filter the pvoc frames, rejecting bins whose frequency is within a band, and with linear interpolation for transitional bands.

Syntax

```
fsig pvsbandr fsigin, xlowcut,
    xlowfull, xhighfull, xhighcut[, ktype]
```

Performance

fsig -- output pv stream

fsigin -- input pv stream.

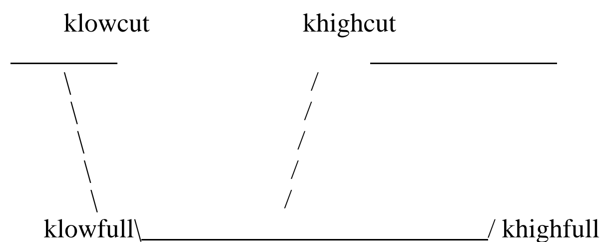
xlowcut, xlowfull, xhighfull, xhighcut -- define a trapezium shape for the band that is rejected. The a-rate versions only apply to the sliding case.

ktype -- specifies the shape of the transitional band. If at the default value of zero the shape is as below, with linear transition in amplitude. Other values give an exponential curve

(1 - exp(r*type)) / (1 - exp(type))

This includes a linear dB shape when ktype is log(10) or about 2.30.

The opcode performs a band-reject filter with a spectral envelope shaped like



Examples

Exemple 388. Example

```
asig  in                                    ; get the signal in
```

```
fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvsbandr  fsig, 200, 400, 1000, 2000 ; filter 300-1500Hz
atps  pvsynth   ftps                     ; synthesise it
```

Credits

Author: John ffitch
December 2007

pvsbin

pvsbin — Obtain the amp and freq values off a PVS signal bin.

Description

Obtain the amp and freq values off a PVS signal bin as k-rate variables.

Syntax

```
kamp, kfr pvsbin fsig, kbin
```

Performance

kamp -- bin amplitude

kfr -- bin frequency

fsig -- an input pv stream

kbin -- bin number

Examples

Here is an example of the pvsbin opcode. It uses the file *pvsbin.csd* [examples/pvsbin.csd]. This example uses realtime input, but you can also use it for soundfile input.

Exemple 389. Example of the pvsbin opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
ifftsize = 1024
iwtype = 1    /* cleaner with hanning window */

;a1   soundin "input.wav"  ;select a soundifle
a1 inch 1   ;Use realtime input

fsig pvsanal   a1, ifftsize, ifftsize/4, ifftsize, iwtype
kamp, kfr pvsbin   fsig, 10
adm  oscil     kamp, kfr, 1
```

```
       out     adm
endin

</CsInstruments>
<CsScore>

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Victor Lazzarini
August 2006

pvsblur

pvsblur — Average the amp/freq time functions of each analysis channel for a specified time.

Description

Average the amp/freq time functions of each analysis channel for a specified time (truncated to number of frames). As a side-effect the input pvoc stream will be delayed by that amount.

Syntax

```
fsig pvsblur fsigin, kblurtime, imaxdel
```

Performance

fsig -- output pv stream

fsigin -- input pv stream.

kblurtime -- time in secs during which windows will be averaged .

imaxdel -- maximum delay time, used for allocating memory used in the averaging operation.

This opcode will blur a pvstream by smoothing the amplitude and frequency time functions (a type of low-pass filtering); the amount of blur will depend on the length of the averaging period, larger blurtimes will result in a more pronounced effect.

Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 390. Example

```
asig  in                                  ; get the signal in

fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvsblur   fsig, 0.2, 0.2           ; blur it for 200 ms
atps  pvsynth   ftps                     ; synthesise it
```

Credits

Author: Victor Lazzarini;

November 2004

New plugin in version 5

November 2004.

pvsbuffer

pvsbuffer — This opcode creates and writes to a circular buffer for streaming PV signals.

Description

This opcode sets up and writes to a circular buffer of length ilen (secs), giving a handle for the buffer and a time pointer, which holds the current write position (also in seconds). It can be used with one or more pvsbufread opcodes. Writing is circular, wrapping around at the end of the buffer.

Syntax

```
ihandle, ktime  pvsbuffer fsig, ilen
```

Initialisation

Initialisation

ihandle -- handle identifying this particular buffer, which should be passed to a reader opcode.

ilen -- buffer length in seconds.

fsig -- an input pv stream

ktime -- the current time of writing in the buffer

Examples

Here is an example of the pvsbuffer opcode.

Exemple 391. Example of the pvsbuffer opcode

With this opcode and pvsbufread, it is possible to, among other things: 1) time-stretch/compress a fsig stream, by reading it at different rates 2) delay a fsig or portions of it. 3) 'brassage' two or more fsigs by switching buffers, since the reading handles are k-rate. Note that, when using k-rate handles, it is important to initialise the k-rate variable to a given handle (so that the fsig initialisation can take place) and it is only possible to switch handles between compatible fsig buffers (with the same fftsize and overlap), eg.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
fsig1   pvsanal   asig1,1024,256,1024,1
fsig2   pvsanal   asig2,1024,256,1024,1


ibuf1,kt1   pvsbuffer   fsig1, 10  ; 10-sec buf with fsig1
ibuf2,kt2   pvsbuffer  fsig2, 7      ; 7-sec buf with fsig2


khan init ibuf1    ; initialise handle to buf1
```

```
if  ktrig  > 0 then   ; switch buffers according to trigger
khan = ibuf2
else
khan = ibuf1
endif

fsb  pvsbufread  kt1, khan   ; read buffer
```

Credits

Author: Victor Lazzarini
July 2007

pvsbufread

pvsbufread — This opcode creates and writes to a circular buffer for streaming PV signals.

Description

This opcode sets up and writes to a circular buffer of length ilen (secs), giving a handle for the buffer and a time pointer, which holds the current write position (also in seconds). It can be used with one or more pvsbufread opcodes. Writing is circular, wrapping around at the end of the buffer.

Syntax

```
fsig pvsbufread  ktime, khandle[, ilo, ihi]
```

Initialisation

Initialisation

ilo, ihi -- set the lowest and highest freqs to be read from the buffer (defaults to 0, Nyquist).

fsig -- output pv stream

ktime -- time position of reading pointer (in secs).

khandle -- handle identifying the buffer to be read. When using k-rate handles, it is important to initialise the k-rate variable to a given existing handle. When changing buffers, fsig buffers need to be compatible (same fsig format).

Examples

Here is an example of the pvsbufread opcode.

Exemple 392. Example of the pvsbufread opcode

With this opcode and pvsbuffer, it is possible to, among other things: 1) time-stretch/compress a fsig stream, by reading it at different rates 2) delay a fsig or portions of it. 3) 'brassage' two or more fsigs by switching buffers, since the reading handles are k-rate. Note that, when using k-rate handles, it is important to initialise the k-rate variable to a given handle (so that the fsig initialisation can take place) and it is only possible to switch handles between compatible fsig buffers (with the same fftsize and overlap), eg.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
fsig1   pvsanal   asig1,1024,256,1024,1
fsig2   pvsanal   asig2,1024,256,1024,1


ibuf1,kt1   pvsbuffer   fsig1, 10  ; 10-sec buf with fsig1
ibuf2,kt2   pvsbuffer  fsig2, 7      ; 7-sec buf with fsig2
```

```
khan init ibuf1    ; initialise handle to buf1


if  ktrig  > 0 then   ; switch buffers according to trigger
khan = ibuf2
else
khan = ibuf1
endif


fsb  pvsbufread  kt1, khan   ; read buffer
```

Credits

Author: Victor Lazzarini
July 2007

pvscale

pvscale — Scale the frequency components of a pv stream.

Description

Scale the frequency components of a pv stream, resulting in pitch shift. Output amplitudes can be optionally modified in order to attempt formant preservation.

Syntax

```
fsig pvscale fsigin, kscal[, ikeepform, igain]]
```

Performance

fsig -- output pv stream

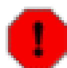
fsigin -- input pv stream

kscal -- scaling ratio.

ikeepform -- attempt to keep input signal -- -- formants; 0: do not keep formants; 1: keep formants by imposing original amps; 2: keep formants by filtering using the original spec envelope (defaults to 0).

igain -- amplitude scaling (defaults to 1).

The quality of the pitch shift will be improved with the use of a Hanning window in the pvoc analysis. Formant preservation is only successful with strong-formant sounds, such as voices and certain instrumental sounds, but also can be used for intersting transformation effects.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 393. Example

```
asig  in                                 ; get the signal in

fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvscale   fsig, 1.5, 1, 2          ; transpose it keeping formants
atps  pvsynth  ftps                      ; synthesise it

adp   delayr .1                          ; delay original signal
adel  deltapn 1024                  ; by 1024 samples
      delayw  asig

       out atps+adel                     ; add tranposed and original
```

The example above shows a vocal harmoniser. The delay is necessary to time-align the signals, as the analysis-synthesis process will imply a delay of 1024 samples between the analysis input and the synthesis output.

Credits

Author: Victor Lazzarini;
November 2004

New plugin in version 5

November 2004.

pvscent

pvscent — Calculate the spectral centroid of a signal.

Description

Canculate the spectral centrols of a signal from its discrete Fourier transform.

Syntax

```
kcent pvscent fsig
```

Performance

kcent -- the spectral centroid

fsig -- an input pv stream

Examples

Exemple 394. Example

```
ifftsize = 1024
iwtype = 1    /* cleaner with hanning window */

a1   soundin "input.wav"

fsig pvsanal   a1, ifftsize, ifftsize/4, ifftsize, iwtype
kcen pvscent   fsig
adm  oscil     32000, kcen, 1

       out    adm
```

Credits

Author: John ffitch;
March 2005

New plugin in version 5

March 2005.

pvscross

pvscross — Performs cross-synthesis between two source fsigs.

Description

Performs cross-synthesis between two source fsigs.

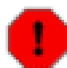
Syntax

```
fsig pvscross fsrc, fdest, kamp1, kamp2
```

Performance

The operation of this opcode is identical to that of *pvcross* (q.v.), except in using *fsig*s rather than analysis files, and the absence of spectral envelope preservation. The amplitudes from *fsrc* and *fdest* (using scale factors *kamp1* for *fsrc* and *kamp2* for *fdest*) are applied to the frequencies of *fsrc*. *kamp1* and *kamp2* must not exceed the range 0 to 1.

With this opcode, cross-synthesis can be performed on real-time audio input, by using *pvsanal* to generate *fsrc* and *fdest*. These must have the same format.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

```
kcross  linseg    0,p3/3,0,p3/3,1,p3/3,1 ; progressive cross-synthesis
fcross  pvscross  fsig1,fsig2,1-kcross,kcross
across  pvsynth   fcross
```

Credits

Author: Richard Dobson
August 2001

November 2003. Thanks to Kanata Motohashi, fixed the link to the *pvcross* opcode.

New in version 4.13

pvsdemix

pvsdemix — Spectral azimuth-based de-mixing of stereo sources.

Description

Spectral azimuth-based de-mixing of stereo sources, with a reverse-panning result. This opcode implements the Azimuth Discrimination and Resynthesis (ADRess) algorithm, developed by Dan Barry (Barry et Al. "Sound Source Separation Azimuth Discrimination and Resynthesis". DAFx'04, Univ. of Napoli). The source separation, or de-mixing, is controlled by two parameters: an azimuth position (kpos) and a subspace width (kwidth). The first one is used to locate the spectral peaks of individual sources on a stereo mix, whereas the second widens the 'search space', including/exclufing the peaks around kpos. These two parameters can be used interactively to extract source sounds from a stereo mix. The algorithm is particularly successful with studio recordings where individual instruments occupy individual panning positions; it is, in fact, a reverse-panning algorithm.

Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Syntax

```
fsig pvsdemix fleft, fright, kpos, kwidth, ipoints
```

Performance

fsig -- output pv stream

fleft -- left channel input pv stream.

fright -- right channel pv stream.

kpos -- the azimuth target centre position, which will be de-mixed, from left to right (-1 <= kpos <= 1). This is the reverse pan-pot control.

kwidth -- the azimuth subspace width, which will determine the number of points around kpos which will be used in the de-mixing process. (1 <= kwidth <= ipoints)

ipoints -- total number of discrete points, which will divide each pan side of the stereo image. This ultimately affects the resolution of the process.

Examples

The example below takes a stereo input and passes through a de-mixing process revealing a source located at ipos +/- iwidth points. These parameters can be controlled in realtime (e.g. using FLTK widgets or MIDI) for an interactive search of sound sources.

Exemple 395. Example

```
ifftsize = 1024
iwtype = 1    /* cleaner with hanning window */
ipos = -0.8  /* to the left of the stereo image */
iwidth = 20  /* use peaks of 20 points around it */

al,ar  soundin "sinput.wav"

flc  pvsanal   al, ifftsize, ifftsize/4, ifftsize, iwtype
frc  pvsanal   ar, ifftsize, ifftsize/4, ifftsize, iwtype
fdm  pvsdemix  flc, frc, kpos, kwidth, 100
adm  pvsynth   fdm

       outs    adm,adm
```

Credits

Author: Victor Lazzarini;
January 2005

New plugin in version 5

January 2005.

pvsdiskin

pvsdiskin — Read a selected channel from a PVOC-EX analysis file.

Description

Create an fsig stream by reading a selected channel from a PVOC-EX analysis file, with frame interpolation.

Syntax

```
fsig pvsdiskinSFname,ktscal,kgain[,ioffset, ichan]
```

Initialization

Sfname -- Name of the analysis file. This must have the .pvx file extension.

A multi-channel PVOC-EX file can be generated using the extended *pvanal utility*.

ichan -- (optional) The channel to read (counting from 1). Default is 1.

ioff -- start offset from beginning of file (secs) (default: 0) .

Performance

ktscal -- time scale, ie. the read pointer speed (1 is normal speed, negative is backwards, 0 < ktscal < 1 is slower and ktscal > 1 is faster)

kgain -- gain scaling.

Examples

```
fsigr pvsdiskin "test.pvx",1,1  ; read PVOCEX file with tscale and gain = 1
aout pvsynth fsigr  ; resynthesise it
```

Credits

Author: Victor Lazzarini
May 2007
New in Csound 5.06

pvsdisp

pvsdisp — Displays a PVS signal as an amplitude vs. freq graph.

Description

This opcode will display a PVS signal fsig. Uses X11 or FLTK windows if enabled, else (or if *-g* flag is set) displays are approximated in ASCII characters.

Syntax

```
pvsdisp fsig[, ibins, iwtflg]
```

Initialization

iprd -- the period of pvsdisp in seconds.

ibins (optional, default=all bins) -- optionally, display only ibins bins.

iwtflg (optional, default=0) -- wait flag. If non-zero, each pvsdisp is held until released by the user. The default value is 0 (no wait).

Performance

pvsdisp -- displays the PVS signal frame-by-frame.

Examples

Here is an example of the pvsdisp opcode. It uses the file *pvsdisp.csd* [examples/pvsdisp.csd]. This example uses realtime input, but you can also use it for soundfile input.

Exemple 396. Example of the pvsdisp opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsdisp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
asig inch 1
;a1   soundin "input.wav"  ;select a soundifle
fsig pvsanal asig, 1024,256, 1024, 1
```

```
pvsdisp fsig

endin

</CsInstruments>
<CsScore>

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

See Also

dispfft, *print*

Credits

Author: V Lazzarini, 2006

pvsfilter

pvsfilter — Multiply amplitudes of a pvoc stream by those of a second pvoc stream, with dynamic scaling.

Description

Multiply amplitudes of a pvoc stream by those of a second pvoc stream, with dynamic scaling.

Syntax

```
fsig pvsfilter fsigin, fsigfil, kdepth[, igain]
```

Performance

fsig -- output pv stream

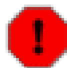
fsigin -- input pv stream.

fsigfil -- filtering pvoc stream.

kdepth -- controls the depth of filtering of fsigin by fsigfil .

igain -- amplitude scaling (optional, defaults to 1).

Here the input pvoc stream amplitudes are modified by the filtering stream, keeping its frequencies intact. As usual, both signals have to be in the same format.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 397. Example

```
kfreq  expon 500, p3, 4000           ; 3-octave sweep
kdepth linseg 1, p3/2, 0.5, p3/2, 1  ; varying filter depth

asig  in                             ; input
afil  oscili  1, kfreq, 1            ; filter t-domain signal

fim   pvsanal  asig,1024,256,1024,0  ; pvoc analysis
fil   pvsanal  afil,1024,256,1024,0
fou   pvsfilter fim, fil, kdepth     ; filter signal
aout  pvsynth  fou                   ; pvoc synthesis
```

In the example above the filter curve will depend on the spectral envelope of afil; in the simple case of a sinusoid, it will be equivalent to a narrowband band-pass filter.

Credits

Author: Victor Lazzarini;
November 2004

New plugin in version 5

November 2004.

pvsfread

pvsfread — Read a selected channel from a PVOC-EX analysis file.

Description

Create an fsig stream by reading a selected channel from a PVOC-EX analysis file loaded into memory, with frame interpolation. Only format 0 files (amplitude+frequency) are currently supported. The operation of this opcode mirrors that of pvoc, but outputs an fsig instead of a resynthesized signal.

Syntax

```
fsig pvsfread ktimpt, ifn [, ichan]
```

Initialization

ifn -- Name of the analysis file. This must have the .pvx file extension.

A multi-channel PVOC-EX file can be generated using the extended *pvanal utility*.

ichan -- (optional) The channel to read (counting from 0). Default is 0.

Performance

ktimpt -- Time pointer into analysis file, in seconds. See the description of the same parameter of *pvoc* for usage.

Note that analysis files can be very large, especially if multi-channel. Reading such files into memory will very likely incur breaks in the audio during real-time performance. As the file is read only once, and is then available to all other interested opcodes, it can be expedient to arrange for a dedicated instrument to preload all such analysis files at startup.

Examples

```
idur  filelen   "test.pvx"         ; find dur of (stereo) analysis file
kpos  line      0,p3,idur          ; to ensure we process whole file
fsigr pvsfread  kpos,"test.pvx",1  ; create fsig from R channel
```

(NB: as this example shows, the filelen opcode has been extended to accept both old and new analysis file formats).

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsfreeze

pvsfreeze — Freeze the amplitude and frequency time functions of a pv stream according to a control-rate trigger.

Description

This opcodes 'freezes' the evolution of pvs stream by locking into steady amplitude and/or frequency values for each bin. The freezing is controlled, independently for amplitudes and frequencies, by a control-rate trigger, which switches the freezing 'on' if equal to or above 1 and 'off' if below 1.

Syntax

```
fsig pvsfreeze fsigin, kfreeza, kfreezf
```

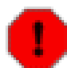
Performance

fsig -- output pv stream

fsigin -- input pv stream.

kfreeza -- freezing switch for amplitudes. Freezing is on if above or equal to 1 and off if below 1.

kfcf -- freezing switch for frequencies. Freezing is on if above or equal to 1 and off if below 1.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 398. Example

```
asig  in                             ; input
ktrig oscil  1.5, 0.25, 1             ; trigger
fim   pvsanal  asig1,1024,256,1024,0  ; pvoc analysis
fou   pvsfreeze fim, abs(ktrig), abs(ktrig)    ; regular 'freeze' of spectra
aout pvsynth fou               ; pvoc synthesis
```

In the example above the input signal will be regularly 'frozen' for a short while, as the trigger rises above 1 about every two seconds.

Credits

Author: Victor Lazzarini;
May 2006

New plugin in version 5

May 2006.

pvsftr

pvsftr — Reads amplitude and/or frequency data from function tables.

Description

Reads amplitude and/or frequency data from function tables.

Syntax

```
pvsftr fsrc, ifna [, ifnf]
```

Initialization

ifna -- A table, at least inbins in size, that stores amplitude data. Ignored if ifna = 0

ifnf (optional) -- A table, at least inbins in size, that stores frequency data. Ignored if ifnf = 0

Performance

fsrc -- a PVOC-EX formatted source.

Enables the contents of *fsrc* to be exchanged with function tables for custom processing. Except when the frame overlap equals *ksmps* (which will generally not be the case), the frame data is not updated each control period. The data in *ifna*, *ifnf* should only be processed when *kflag* is set to 1. To process only frequency data, set *ifna* to zero.

As the function tables are required only to store data from *fsrc*, there is no advantage in defining then in the score, and they should generally be created in the instrument, using *ftgen*.

By exporting amplitude data, say, from one fsig and importing it into another, basic cross-synthesis (as in *pvscross*) can be performed, with the option to modify the data beforehand using the table manipulation opodes.

Note that the format data in the source fsig is not written to the tables. This therefore offers a means of transferring amplitude and frequency data between non-identical fsigs. Used this way, these opcodes become potentially pathological, and can be relied upon to produce unexpected results. In such cases, resynthesis using *pvsadsyn* would almost certainly be required.

To perform a straight copy from one fsig to another one of identical format, the conventional assignment syntax can be used:

```
fsig1 = fsig2
```

It is not necessary to use function tables in this case.

Examples

```
ifn     ftgen      0,0,inbins,10,1        ; make ftable
kflag   pvsftw     fsrc,ifn          ; export  amps to table,
kamp    init       0
if      kflag==0   kgoto contin   ; only proc when frame is ready
; kill lowest bins, for obvious effect
        tablew      kamp,1,ifn
        tablew      kamp,2,ifn
        tablew      kamp,3,ifn
        tablew      kamp,4,ifn
; read modified data back to fsrc
        pvsftr      fsrc,ifn
contin:
; and resynth
aout    pvsynth    fsrc
```

See Also

pvsftw

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsftw

pvsftw — Writes amplitude and/or frequency data to function tables.

Description

Writes amplitude and/or frequency data to function tables.

Syntax

```
kflag pvsftw fsrc, ifna [, ifnf]
```

Initialization

ifna -- A table, at least inbins in size, that stores amplitude data. Ignored if ifna = 0

ifnf -- A table, at least inbins in size, that stores frequency data. Ignored if ifnf = 0

Performance

kflag -- A flag that has the value of 1 when new data is available, 0 otherwise.

fsrc -- a PVOC-EX formatted source.

Enables the contents of *fsrc* to be exchanged with function tables, for custom processing. Except when the frame overlap equals *ksmps* (which will generally not be the case), the frame data is not updated each control period. The data in *ifna*, *ifnf* should only be processed when *kflag* is set to 1. To process only frequency data, set *ifna* to zero.

As the functions tables are required only to store data from *fsrc*, there is no advantage in defining then in the score. They should generally be created in the instrument using *ftgen*.

By exporting amplitude data, say, from one fsig and importing it into another, basic cross-synthesis (as in *pvscross*) can be performed, with the option to modify the data beforehand using the table manipulation opodes.

Note that the format data in the source fsig is not written to the tables. This therefore offers a means of transferring amplitude and frequency data between non-identical fsigs. Used this way, these opcodes become potentially pathological, and can be relied upon to produce unexpected results. In such cases, resynthesis using *pvsadsyn* would almost certainly be required.

To perform a straight copy from one fsig to another one of identical format, the conventional assignment syntax can be used:

fsig1 = fsig2

It is not necessary to use function tables in this case.

Examples

```
ifn     ftgen      0,0,inbins,10,1        ; make ftable
kflag   pvsftw     fsrc,ifn          ; export  amps to table,
kamp    init       0
if      kflag==0   kgoto contin   ; only proc when frame is ready
; kill lowest bins, for obvious effect
        tablew      kamp,1,ifn
        tablew      kamp,2,ifn
        tablew      kamp,3,ifn
        tablew      kamp,4,ifn
; read modified data back to fsrc
        pvsftr      fsrc,ifn
contin:
; and resynth
aout    pvsynth    fsrc
```

See Also

pvsftr

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsfwrite

pvsfwrite — Ecrit un signal fsig dans un fichier PVOCEX.

Description

Cet opcode écrit un signal fsig dans un fichier PVOCEX (qui peut être lu à son tour par *pvsfread* ou par d'autres programmes qui supportent les fichiers PVOCEX en entrée).

Syntaxe

```
pvsfwrite fsig, ifile
```

Initialisation

fsig -- données du fsig en entrée. *ifile* -- nom du fichier (une chaîne de caractères entre guillemets) .

Exemples

Voici un exemple de l'opcode pvsfwrite. Il utilise le fichier *pvsfwrite.csd* [examples/pvsfwrite.csd]. This example uses realtime audio input.

Exemple 399. Exemple de l'opcode pvsfwrite

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsfwrite.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

;By Victor Lazzarini 2008

instr 1
asig oscili 10000, 440, 1
fss pvsanal  asig, 1024,256,1024,0
pvsfwrite fss, "mypvs.pvx"
ase pvsynth fss
           out ase
endin

instr 2 ; must be called after instr 1 finishes
ktim timeinsts
fss  pvsfread ktim, "mypvs.pvx"
asig pvsynth fss
  out asig
endin
```

```
</CsInstruments>
<CsScore>
f1 0 16384 10 1
i1 0   1
i2 1   1
e
</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Victor Lazzarini;
Novembre 2004

Nouveau plugin dans la version 5

Novembre 2004.

pvshift

pvshift — Shift the frequency components of a pv stream, stretching/compressing its spectrum.

Description

Shift the frequency components of a pv stream, stretching/compressing its spectrum.

Syntax

```
fsig pvshift fsigin, kshift, klowest[, ikeepform, igain]
```

Performance

fsig -- output pv stream

fsigin -- input pv stream

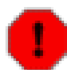
kshift -- shift amount (in Hz, positive or negative).

klowest -- lowest frequency to be shifted.

ikeepform -- attempt to keep input signal formants; 0: do not keep formants; 1: keep formants by imposing original amps; 2: keep formants by filtering using the original spec envelope (defaults to 0).

igain -- amplitude scaling (defaults to 1).

This opcode will shift the components of a pv stream, from a certain frequency upwards, up or down a fixed amount (in Hz). It can be used to transform a harmonic spectrum into an inharmonic one. The ikeepform flag can be used to try and preserve formants for possibly interesting and unusual spectral modifications.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 400. Example

```
asig  in                                 ; get the signal in

fsig  pvsanal   asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvshift   fsig, 100, 0             ; add 100 Hz to each component
atps  pvsynth   ftps                     ; synthesise it
```

Depending on the input, this will transform a pitched sound into an inharmonic, bell-like sound.

Credits

Author: Victor Lazzarini
November 2004

New plugin in version 5

Nivember 2004.

pvsifd

pvsifd — Instantaneous Frequency Distribution, magnitude and phase analysis.

Description

The pvsifd opcode takes an input a-rate signal and performs an Instantaneous Frequency, magnitude and phase analysis, using the STFT and pvsifd (Instantaneous Frequency Distribution), as described in Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. It generates two PV streaming signals, one containing the amplitudes and frequencies (a similar output to pvsanal) and another containing amplitudes and unwrapped phases.

Syntax

```
ffr,fphs pvsifd ain, ifftsize, ihopsize, iwintype[,iscal]
```

Performance

ffr -- output pv stream in AMP_FREQ format

fphs -- output pv stream in AMP_PHASE format

ifftsize -- FFT analysis size, must be power-of-two and integer multiple of the hopsize.

ihopsize -- hopsize in samples

iwintype -- window type (O: Hamming, 1: Hanning)

iscal -- amplitude scaling (defaults to 1).

Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 401. Example

```
ain inch 1                  ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; pvsifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
      aout  resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows the pvsifd analysis feeding into partial tracking and cubic-phase additive re-

synthesis with pitch shifting.

Credits

Author: Victor Lazzarini;
June 2005

New plugin in version 5

November 2004.

pvsinfo

pvsinfo — Get information from a PVOC-EX formatted source.

Description

Get format information about fsrc, whether created by an opcode such as pvsanal, or obtained from a PVOCEX file by pvsfread. This information is available at init time, and can be used to set parameters for other pvs opcodes, and in particular for creating function tables (e.g. for pvsftw), or setting the number of oscillators for pvsadsyn.

Syntax

```
ioverlap, inumbins, iwinsize, iformat pvsinfo fsrc
```

Initialization

ioverlap -- The stream overlap size.

inumbins -- The number of analysis bins (amplitude+frequency) in fsrc. The underlying FFT size is calculated as (inumbins -1) * 2.

iwinsize -- The analysis window size. May be larger than the FFT size.

iformat -- The analysis frame format. If fsrc is created by an opcode, iformat will always be 0, signifying amplitude+frequency. If fsrc is defined from a PVOC-EX file, iformat may also have the value 1 or 2 (amplitude+phase, complex).

Examples

```
fim                 pvsfread  "test.pvx"    ; import pvocex file
iovl,inb,iws,ifmt  pvsinfo   fim           ; get inumbins info
ifn                 ftgen     0,0,inb,10,1  ; and create f-table
```

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsinit

pvsinit — Initialise a spectral (f) variable to zero.

Description

Fermorms the equavent to an init operation on an f-variable.

Syntax

```
fsig pvsinit isize[,iolap,iwinsize,iwintype, iformat]
```

Performance

fsig -- output pv stream set to zero.

isize -- size of the DFT frame.

iolap -- size of the analysis overlap, defaults to isize/4.

iwinsize -- size of the analysis window, defaults to isize.

iwintype -- type of analysis window, defaults to 1, Hanning.

iformat -- pvsdata format, defaults to 0:PVS_AMP_FREQ.

Examples

Exemple 402. Example

```
fsig  pvsinit   1024
```

Credits

Author: Victor Lazzarini;
November 2004

New plugin in version 5

November 2004.

pvsin

pvsin — Retrieve an fsig from the input software bus; a pvs equivalent to chani.

Description

This opcode retrieves an f-sig from the pvs in software bus, which can be used to get data from an external source, using the Csound 5 API. A channel is created if not already existing. The fsig channel is in that case initialised with the given parameters. It is important to note that the pvs input and output (pvsout opcode) busses are independent and data is not shared between them.

Syntax

```
fsig pvsin kchan[,isize,iolap,iwinsize,iwintype,iformat]
```

Initialisation

isize -- initial DFT size,defaults to 1024.

iolap -- size of overlap, defaults to isize/4.

isize -- size of analysis window, defaults to isize.

isize -- type of window, defaults to Hanning (1) (see pvsanal)

isize -- data format, defaults 0 (PVS_AMP_FREQ). Other possible values are 1 (PVS_AMP_PHASE), 2 (PVS_COMPLEX) or 3 (PVS_TRACKS).

Performance

fsig -- output fsig.

kchan -- channel number. If non-existent, a channel will be created.

Examples

Exemple 403. Example

```
fsig  pvsin   0 ; get data from pvs in bus channel 0
```

Credits

Author: Victor Lazzarini;
Auust 2006

pvsmaska

pvsmaska — Modify amplitudes using a function table, with dynamic scaling.

Description

Modify amplitudes of fsrc using function table, with dynamic scaling.

Syntax

```
fsig pvsmaska fsrc, ifn, kdepth
```

Initialization

ifn -- The f-table to use. Given fsrc has N analysis bins, table ifn must be of size N or larger. The table need not be normalized, but values should lie within the range 0 to 1. It can be supplied from the score in the usual way, or from within the orchestra by using *pvsinfo* to find the size of fsrc, (returned by pvsinfo in inbins), which can then be passed to ftgen to create the f-table.

Performance

kdepth -- Controls the degree of modification applied to fsrc, using simple linear scaling. 0 leaves amplitudes unchanged, 1 applies the full profile of ifn.

Note that power-of-two FFT sizes are particularly convenient when using table-based processing, as the number of analysis bins (inbins) is then a power-of-two plus one, for which an exactly matching f-table can be created. In this case it is important that the f-table be created with a size of inbins, rather than as a power of two, as the latter will copy the first table value to the guard point, which is inappropriate for this opcode.

Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 404. Example (using score-supplied f-table, assuming fsig fftsize = 1024)

```
; score f-table using cubic spline to define shaped peaks
f1 0 513 8 0 2 1 3 0 4 1 6 0 10 1 12 0 16 1 32 0 1 0 436 0

asig  buzz     20000,199,50,1         ; pulsewave source
fsig  pvsanal  asig,1024,256,1024,0   ; create fsig
kmod  linseg   0,p3/2,1,p3/2,0        ; simple control sig

fsig2  pvsmaska fsig,2,kmod            ; apply weird eq to fsig
aout  pvsynth  fsig2                   ; resynthesize,
      dispfft  aout,0.1,1024          ; and view the effect
```

Credits

Author: Richard Dobson
August 2001

New in version 4.13

pvsmix

pvsmix — Mix 'seamlessly' two pv signals.

Description

Mix 'seamlessly' two pv signals. This opcode combines the most prominent components of two pvoc streams into a single mixed stream.

Syntax

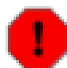
```
fsig pvsmix fsigin1, fsigin2
```

Performance

fsig -- output pv stream

fsigin1 -- input pv stream.

fsigin2 -- input pv stream, which must have same format as fsigin1.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 405. Example

```
fsig1 pvsanal  asig1,1024,256,1024,0  ; pvoc analysis
fsig2   pvsanal  asig2,1024,256,1024,0
fsigout pvsmix  fsig1, fsig2         ; mix signals
aout pvsynth fsigout              ; pvoc synthesis
```

Depending on the input, this will transform a pitched sound into an inharmonic, bell-like sound.

Credits

Author: Victor Lazzarini
November 2004

New plugin in version 5

Nivember 2004.

pvsmorph

pvsmorph — Performs morphing (or interpolation) between two source fsigs.

Description

Performs morphing (or interpolation) between two source fsigs.

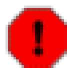
Syntax

```
fsig pvsmorph fsig1, fsig2, kampint, kfrqint
```

Performance

The operation of this opcode is similar to that of *pvinterp* (q.v.), except in using *fsig*s rather than analysis files, and the absence of spectral envelope preservation. The amplitudes and frequencies of *fsig1* are interpolated witht those of *fsig2*, depeding on the values of *kampint* and *kfrqint*, respectively. These range between 0 and 1, where 0 means *fsig1* and 1, *fsig2*. Anything in between will interpolate amps and/or freqs of the two fsigs.

With this opcode, morphing can be performed on real-time audio input, by using *pvsanal* to generate *fsig1* and *fsig2*. These must have the same format.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

```
km  linseg    0,p3/3,0,p3/3,1,p3/3,1 ; progressive morphing
fmo  pvsmorph  fsig1,fsig2,km,km
asig  pvsynth   fmo
```

Credits

Author: Victor Lazzarini
April 2007
New in Csound 5.06

pvsmooth

pvsmooth — Smooth the amplitude and frequency time functions of a pv stream using parallel 1st order lowpass IIR filters with time-varying cutoff frequency.

Description

Smooth the amplitude and frequency time functions of a pv stream using a 1st order lowpass IIR with time-varying cutoff frequency. This opcode uses the same filter as the 'tone' opcode, but this time acting separately on the amplitude and frequency time functions that make up a pv stream. The cutoff frequency parameter runs at the control-rate, but unlike tone and tonek, it is not specified in Hz, but as fractions of 1/2 frame-rate (actually the pv stream sampling rate), which is easier to understand. This means that the highest cutoff frequency is 1 and the lowest 0; the lower the frequency the smoother the functions and more pronounced the effect will be. This opcode produces effects that are more or less similar to pvsblur, but with two important differences: 1.smoothing of amplitudes and frequencies use separate sets of filters; and 2. there is no increase in computational cost when higher amounts of 'blurring' (smoothing) are desired.

Syntax

```
fsig pvsmooth fsigin, kacf, kfcf
```

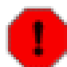
Performance

fsig -- output pv stream

fsigin -- input pv stream.

kacf -- amount of cutoff frequency for amplitude function filtering, between 0 and 1, in fractions of 1/2 frame-rate.

kfcf -- amount of cutoff frequency for frequency function filtering, between 0 and 1, in fractions of 1/2 frame-rate.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 406. Example

```
asig1  in                              ; input
fim   pvsanal  asig1,1024,256,1024,0  ; pvoc analysis
fou   pvsmooth fim, 0.01, 0.01        ; smooth with cf at 1% of 1/2 frame-rate (ca 8.6 Hz)
aout  pvsynth fou                     ; pvoc synthesis
```

In the example above the input signal will be smoothed/blurred by pvsmooth with a cutoff frequency of 1% of 1/2 frame-rate (which is about 172Hz, so the cf is about 8.6Hz) .

Credits

Author: Victor Lazzarini;
May 2006

New plugin in version 5

May 2006.

pvsout

pvsout — Write a fsig to the pvs output bus.

Description

This opcode writes a fsig to a channel of the pvs output bus. Note that the pvs out bus and the pvs in bus are separate and independent. A new channel is created if non-existent.

Syntax

```
pvsout fsig, kchan
```

Performance

fsig -- fsig input data.

kchan -- pvs out bus channel number.

Examples

Exemple 407. Example

```
asig in ; input
fsig pvsanal asig, 1024, 256, 1024, 1 ; analysis
pvsout   fsig,0 ; write signal to pvs out bus channel 0
```

Credits

Author: Victor Lazzarini;
August 2006

pvsosc

pvsosc — PVS-based oscillator simulator.

Description

Generates periodic signal spectra in AMP-FREQ format, with the option of four wave types:

1. sawtooth-like (harmonic weight 1/n, where n is partial number)
2. square-like (similar to 1., but only odd partials)
3. pulse (all harmonics with same weight)
4. cosine

Complex waveforms (ie. all types except cosine) contain all harmonics up to the Nyquist. This makes pvsosc an option for generation of band-limited periodic waves. In addition, types can be changed using a k-rate variable.

Syntax

```
fsig pvsosc kamp, kfreq, ktype, isize [,ioverlap] [, iwinsize] [, iwintype] [, iformat]
```

Initialisation

fsig -- output pv stream set to zero.

isize -- size of analysis frame and window.

ioverlap -- (Optional) size of overlap, defaults to isize/4.

iwinsize -- (Optional) window size, defaults to isize.

iwintype -- (Optional) window type, defaults to Hanning. The choices are currently:

- 0 = Hamming window
- 1 = von Hann window

iformat -- (Optional) data format, defaults to 0 which produces AMP:FREQ data. That is currently the only option.

Performance

kamp -- signal amplitude. Note that the actual signal amplitude can, depending on wave type and frequency, vary slightly above or below this value. Generally the amplitude will tend to exceed kamp on higher frequencies (> 1000 Hz) and be reduced on lower ones. Also due to the overlap-add process, when resynthesing with pvsynth, frequency glides will cause the output amplitude to fluctuate above and below kamp.

kfreq -- fundamental frequency in Hz.

ktype -- wave type: 1. sawtooh-like, 2.square-like, 3.pulse and any other value for cosine.

Examples

Here is an example of the pvsosc opcode. It uses the file *pvsosc.csd* [examples/pvsosc.csd].

Exemple 408. Example of the pvsosc opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsosc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; a band-limited sawtooth-wave oscillator
fsig  pvsosc   10000, 440, 1, 1024 ; generate wave spectral signal
asig pvsynth fsig                       ; resynthesise it
out asig
endin

instr 2
; a band-limited square-wave oscillator
fsig  pvsosc   10000, 440, 2, 1024 ; generate wave spectral signal
asig pvsynth fsig                       ; resynthesise it
out asig
endin


instr 3
; a pulse oscillator
fsig  pvsosc   10000, 440, 3, 1024 ; generate wave spectral signal
asig pvsynth fsig                       ; resynthesise it
out asig
endin

instr 4
; a cosine-wave oscillator
fsig  pvsosc   10000, 440, 4, 1024 ; generate wave spectral signal
asig pvsynth fsig                       ; resynthesise it
out asig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 2 1
i 3 4 1
i 4 6 1

e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Victor Lazzarini;
August 2006

pvspitch

pvspitch — Track the pitch and amplitude of a PVS signal.

Description

Track the pitch and amplitude of a PVS signal as k-rate variables.

Syntax

```
kfr, kamp pvspitch fsig, kthresh
```

Performance

kamp -- Amplitude of fundamental frequency

kfr -- Fundamental frequency

fsig -- an input pv stream

kthresh -- analysis threshold (between 0 and 1). Higher values will eliminate low-amplitude components from the analysis.

Performance

The pitch detection algorithm implemented by *pvspitch* is based upon J. F. Schouten's hypothesis of the neural processes of the brain used to determine the pitch of a sound after the frequency analysis of the basilar membrane. Except for some further considerations, *pvspitch* essentially seeks out the highest common factor of an incoming sound's spectral peaks to find the pitch that may be attributed to it.

In general, input sounds that exhibit pitch will also exhibit peaks in their spectrum according to where their harmonics lie. There are some the exceptions, however. Some sounds whose spectral representation is continuous can impart a sensation of pitch. Such sounds are explained by the centroid or center of gravity of the spectrum and are beyond the scope of the method of pitch detection implemented by *pvspitch* (Using opcodes like *pvscent* might be more appriopriate in these cases).

pvspitch is able (using a previous analysis *fsig* generated by *pvsanal*) to locate the spectral peaks of a signal. The threshold parameter (*kthresh*) is of utmost importance, as adjusting it can introduce weak yet significant harmonics into the calculation of the fundamental. However, bringing *kthresh* too low would allow harmonically unrelated partials into the analysis algorithm and this will compromise the method's accuracy. These initial steps emulate the response of the basilar membrane by identifying physical characteristics of the input sound. The choice of *kthresh* depends on the actual level of the input signal, since its range (from 0 to 1) spans the whole dynamic range of an analysis bin (from -inf to 0dBFS).

It is important to remember that the input to the *pvspitch* opcode is assumed to be characterised by strong partials within its spectrum. If this is not the case, the results outputted by the opcode may not bear any relation to the pitch of the input signal. If a spectral frame with many unrelated partials was analysed, the greatest common factor of these frequency values that allows for adjacent “harmonics” would be chosen. Thus, noisy frames can be characterised by low frequency outputs of *pvspitch*. This fact allows for a primitive type of instrumental transient detection, as the attack portion of some instrumental tones contain inharmonic components. Should the lowest frequency of the analysed melody be known, then all frequencies detected below this threshold are inaccurate readings, due to the presence of unrelated partials.

In order to facilitate efficient testing of the *pvspitch* algorithm, an amplitude value proportional to the one in the observed in the signal frame is also outputted (*kamp*). The results of *pvspitch* can then be employed to drive an oscillator whose pitch can be audibly compared with that of the original signal (In the example below, an oscillator generates a signal which appears a fifth above the detected pitch).

Examples

Here is an example of the pvspitch opcode. It uses the file *pvspitch.csd* [examples/pvspitch.csd]. This example uses realtime audio input but can be used for audiofile input as well.

Exemple 409. Example of the pvspitch opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvspitch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1


giwave ftgen 0, 0, 4096, 10, 1, 0.5, 0.333, 0.25, 0.2, 0.1666

instr 1

ifftsize = 1024
iwtype = 1    /* cleaner with hanning window */

a1 inch 1 ;Realtime audio input
;a1   soundin "input.wav" ;Use this line for file input

fsig pvsanal   a1, ifftsize, ifftsize/4, ifftsize, iwtype
kfr, kamp pvspitch   fsig, 0.01

adm  oscil     kamp, kfr * 1.5, giwave  ;Generate note a fifth above detected pitch

       out    adm
endin


</CsInstruments>
<CsScore>

i 1 0 30

e

</CsScore>
</CsoundSynthesizer>
```

See also

pvsanal, *pvscent*

Credits

Author: Alan OCinneide
August 2005, added by V Lazzarini, August 2006
Part of the text has been adapted from the Csound Journal winter 2006 issue's article "Introducing PVS-PITCH: A pitch tracking opcode for Csound" by Alan OCinneide. The article is available at: *www.csounds.com/journal/2006winter/pvspitch.html*
[http://www.csounds.com/journal/2006winter/pvspitch.html]

pvstencil

pvstencil — Transforms a pvoc stream according to a masking function table.

Description

Transforms a pvoc stream according to a masking function table; if the pvoc stream amplitude falls below the value of the function for a specific pvoc channel, it applies a gain to that channel.

The pvoc stream amplitudes are compared to a masking table, if the fall below the table values, they are scaled by kgain. Prior to the operation, table values are scaled by klevel, which can be used as masking depth control.

Tables have to be at least fftsize/2 in size; for most GENS it is important to use an extended-guard point (size power-of-two plus one), however this is not necessary with GEN43.

One of the typical uses of pvstencil would be in noise reduction. A noise print can be analysed with pvanal into a PVOCEX file and loaded in a table with GEN43. This then can be used as the masking table for pvstencil and the amount of reduction would be controlled by kgain. Skipping post-normalisation will keep the original noise print average amplitudes. This would provide a good starting point for a successful noise reduction (so that klevel can be generally set to close to 1).

Other possible transformation effects are possible, such as filtering and `inverse-masking'.

Syntax

```
fsig pvstencil fsigin, kgain, klevel, iftable
```

Performance

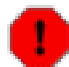
fsig -- output pv stream

fsigin -- input pv stream.

kgain -- `stencil' gain.

klevel -- masking function level (scales the ftable prior to `stenciling') .

iftable -- masking function table.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 410. Example

```
fsig    pvsanal   asig, 1024, 256, 1024, 1
fclean  pvstencil fsig, 0, 1, 1
aclean  pvsynth  fclean
```

Credits

Author: Victor Lazzarini;
November 2004

New plugin in version 5

Nivember 2004.

pvsvoc

pvsvoc — Combine the spectral envelope of one fsig with the excitation (frequencies) of another.

Description

This opcode provides support for cross-synthesis of amplitudes and frequencies. It takes the amplitudes of one input fsig and combines with frequencies from another. It is a spectral version of the well-known channel vocoder.

Syntax

```
fsig pvsvoc famp, fexc, kdepth, kgain
```

Performance

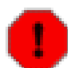
fsig -- output pv stream

famp -- input pv stream from which the amplitudes will be extracted

fexc -- input pv stream from which the frequencies will be taken

kdepth -- depth of effect, affecting how much of the frequencies will be taken from the second fsig: 0, the output is the famp signal, 1 the output is the famp amplitudes and fexc frequencies.

kgain -- gain boost/attenuation applied to the output.



Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

Examples

Exemple 411. Example

```
asig  in                                ; get the signal in
asyn  oscili 16000, 150, 1              ; excitation signal

famp  pvsanal   asig, 1024, 256, 1024, 1 ; analyse in signal
fexc  pvsanal   asyn, 1024, 256, 1024, 1 ; analyse excitation signal
ftps  pvsvoc   famp, fexc, 1, 1     ; cross it
atps  pvsynth  ftps                     ; synthesise it

       out atps
```

The example above shows a typical cross-synthesis operation. The input signal (say a vocal sound) is

used for its amplitude spectrum. An oscillator with an arbitrary complex waveform produces the excitation signal, giving the vocal sound its pitch.

Credits

Author: Victor Lazzarini;
April 2005

New plugin in version 5

April 2005.

pvsynth

pvsynth — Resynthesise using a FFT overlap-add.

Description

Resynthesise using a FFT overlap-add.

Syntax

```
ares pvsynth fsrc, [iinit]
```

Performance

ares -- output audio signal

fsrc -- input signal

iinit -- not yet implemented.

Examples

Exemple 412. Example (using score-supplied f-table, assuming fsig fftsize = 1024)

```
; score f-table using cubic spline to define shaped peaks
f1 0 513 8 0 2 1 3 0 4 1 6 0 10 1 12 0 16 1 32 0 1 0 436 0

asig  buzz     20000,199,50,1        ; pulsewave source
fsig  pvsanal  asig,1024,256,1024,0  ; create fsig
kmod  linseg   0,p3/2,1,p3/2,0       ; simple control sig

fsig  pvsmaska fsig,2,kmod           ; apply weird eq to fsig
aout  pvsynth  fsig                  ; resynthesize,
      dispfft  aout,0.1,1024         ; and view the effect
```

This also illustrates that the usual Csound behaviour applies to fsigs; the same name can be used for both input and output.

See Also

pvsadsyn

Credits

Author: Richard Dobson
August 2001

New in version 4.13

February 2004. Thanks to a note from Francisco Vila, updated the example.

pyassign Opcodes

pyassign — Assign the value of the given Csound variable to a Python variable possibly destroying its previous content.

Syntax

```
pyassign "variable", kvalue
```

```
pyassigni "variable", ivalue
```

```
pylassign "variable", kvalue
```

```
pylassigni "variable", ivalue
```

```
pyassignt ktrigger, "variable", kvalue
```

```
pylassignt ktrigger, "variable", kvalue
```

Description

Assign the value of the given Csound variable to a Python variable possibly destroying its previous content. The resulting Python object will be a float.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

pycall Opcodes

pycall — Invoke the specified Python callable at k-time and i-time (i suffix), passing the given arguments. The call is perfomed in the global environment, and the result (the returning value) is copied into the Csound output variables specified.

Syntax

```
                                          pycall   "callable", karg1, ...
kresult                                   pycall1  "callable", karg1, ...
kresult1, kresult2                        pycall2  "callable", karg1, ...
kr1, kr2, kr3                             pycall3  "callable", karg1, ...
kr1, kr2, kr3, kr4                        pycall4  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5                   pycall5  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6              pycall6  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7         pycall7  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8    pycall8  "callable", karg1, ...

                                          pycallt   ktrigger, "callable", karg1, ...
kresult                                   pycall1t  ktrigger, "callable", karg1, ...
kresult1, kresult2                        pycall2t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3                             pycall3t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4                        pycall4t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5                   pycall5t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6              pycall6t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7         pycall7t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8    pycall8t  ktrigger, "callable", karg1, ...

                                          pycalli   "callable", karg1, ...
iresult                                   pycall1i  "callable", iarg1, ...
iresult1, iresult2                        pycall2i  "callable", iarg1, ...
ir1, ir2, ir3                             pycall3i  "callable", iarg1, ...
ir1, ir2, ir3, ir4                        pycall4i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5                   pycall5i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6              pycall6i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7         pycall7i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8    pycall8i  "callable", iarg1, ...

pycalln   "callable", nresults, kresult1, ..., kresultn, karg1, ...
pycallni  "callable", nresults, iresult1, ..., iresultn, iarg1,  ...

                                          pylcall   "callable", karg1, ...
kresult                                   pylcall1  "callable", karg1, ...
kresult1, kresult2                        pylcall2  "callable", karg1, ...
kr1, kr2, kr3                             pylcall3  "callable", karg1, ...
kr1, kr2, kr3, kr4                        pylcall4  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5                   pylcall5  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6              pylcall6  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7         pylcall7  "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8    pylcall8   "callable", karg1, ...

                                          pylcallt   ktrigger, "callable", karg1, ...
kresult                                   pylcall1t  ktrigger, "callable", karg1, ...
kresult1, kresult2                        pylcall2t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3                             pylcall3t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4                        pylcall4t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5                   pylcall5t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6              pylcall6t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7         pylcall7t  ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8    pylcall8t  ktrigger, "callable", karg1, ...

                                          pylcalli   "callable", karg1, ...
iresult                                   pylcall1i  "callable", iarg1, ...
iresult1, iresult2                        pylcall2i  "callable", iarg1, ...
ir1, ir2, ir3                             pylcall3i  "callable", iarg1, ...
ir1, ir2, ir3, ir4                        pylcall4i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5                   pylcall5i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6              pylcall6i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7         pylcall7i  "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8    pylcall8i  "callable", iarg1, ...
```

```
pylcalln   "callable", nresults, kresult1, ..., kresultn, karg1, ...
pylcallni  "callable", nresults, iresult1, ..., iresultn, iarg1, ...
```

Description

This family of opcodes call the specified Python callable at k-time and i-time (i suffix), passing the given arguments. The call is perfomed in the global environment and the result (the returning value) is copied into the Csound output variables specified.

They pass any number of parameters which are cast to float inside the Python interpreter.

The *pycall*/*pycalli*, *pycall1*/*pycall1i* ... *pycall8*/*pycall8i* opcodes can accomodate for a number of results ranging from 0 to 8 according to their numerical prefix (0 is omitted).

The *pycalln*/*pycallni* opcodes can accomodate for any number of results: the callable name is followed by the number of output arguments, then come the list of Csound output variable and the list of parameters to be passed.

The returning value of the callable must be `None` for *pycall* or *pycalli*, a float for *pycall1i* or *pycall1i* and a tuple (with proper size) of floats for the *pycall2*/*pycall2i* ... *pycall8*/*pycall8i* and *pycalln*/*pycallni* opcodes.

Examples

Exemple 413. Calling a C or Python function

Supposing we have previously defined or imported a function named `get_number_from_pool` as:

```
from random import random, choice

# a pool of 100 numbers
pool = [i ** 1.3 for i in range(100)]

def get_number_from_pool(n, p):
    # substitute an old number with the new number?
    if random() < p:
        i = choice(range(len(pool)))
        pool[i] = n

    # return a random number from the pool
    return choice(pool)
```

then the following orchestra code

```
k2   pycall1 "get_number_from_pool", k1, p6
```

would set k2 randomly from a pool of numbers changing in time. You can pass new pools elements and control the change rate from the orchestra.

Exemple 414. Calling a Function Object

A more generic implementation of the previous example makes use of a simple function object:

```
from random import random, choice
```

```
class GetNumberFromPool:
    def __init__(self, e, begin=0, end=100, step=1):
        self.pool = [i ** e for i in range(begin, end, step)]

    def __call__(self, n, p):
        # substitute an old number with the new number?
        if random() < p:
            i = choice(range(len(pool)))
            pool[i] = n

        # return a random number from the pool
        return choice(pool)

get_number_from_pool1 = GetNumberFromPool(1.3)
get_number_from_pool2 = GetNumberFromPool(1.5, 50, 250, 2)
```

Then the following orchestra code:

```
k2   pycall1 "get_number_from_pool1", k1, p6
k4   pycall1 "get_number_from_pool2", k3, p7
```

would set k2 and k3 randomly from a pool of numbers changing in time. You can pass new pools elements (here k1 and k3) and control the change rate (here p6 and p7) from the orchestra.

As you can see in the first snippet, you can customize the initialization of the pool as well as create several pools.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

pyeval Opcodes

pyeval — Evaluate a generic Python expression and store the result in a Csound variable at k-time or i-time (i suffix).

Syntax

```
kresult pyeval "expression"

iresult pyevali "expression"

kresult pyleval "expression"

iresult pylevali "expression"

kresult pyevalt ktrigger, "expression"

kresult pylevalt ktrigger, "expression"
```

Description

These opcodes evaluate a generic Python expression and store the result in a Csound variable at k-time or i-time (i suffix).

The expression must evaluate in a float or an object that can be cast to a float.

They can be used effectively to trasfer data from a Python object into a Csound variable.

Example of the pyeval Opcode Group

The code:

```
k1          pyeval      "v1"
```

will copy the content of the Python variable v1 into the Csound variable k1 at each k-time.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

pyexec Opcodes

pyexec — Execute a script from a file at k-time or i-time (i suffix).

Syntax

pyexec "filename"

pyexeci "filename"

pylexec "filename"

pylexeci "filename"

pyexect ktrigger, "filename"

plyexect ktrigger, "filename"

Description

Execute a script from a file at k-time or i-time (i suffix).

This is not the same as calling the script with the `system()` call, since the code is executed by the embedded interpreter.

The code contained in the specified file is executed in the global environment for opcodes pyexec and pyexeci and in the private environment for the opcodes pylexec and pylexeci.

These opcodes perform no message passing. However, since the statement has access to the main namespace and the private namespace, it can interact with objects previously created in that environment.

The "local" version of the *pyexec* opcodes are useful when the code ran by different instances of an instrument should not interact.

Example of the pyexec Opcode Group

Exemple 415. Orchestra (pyexec.orc)

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;If you're not running CsoundAC you need the following line
;to initialize the python interpreter
;pyinit

        pyruni "import random"

        pyexeci "pyexec1.py"

instr 1

        pyexec          "pyexec2.py"
```

```
        pylexeci        "pyexec3.py"
        pylexec         "pyexec4.py"

endin
```

Exemple 416. Score (pyexec.sco)

```
i1 0 0.01
i1 0 0.01
```

Exemple 417. The pyexec1.py Script

```
import time, os

print
print "Welcome to Csound!"

try:
    s = ', %s?' % os.getenv('USER')
except:
    s = '?'

print 'What sound do you want to hear today%s' % s
answer = raw_input()
```

Exemple 418. The pyexec2.py script

```
print 'your answer is "%s"' % answer
```

Exemple 419. The pyexec3.py script

```
message = 'a private random number: %f' % random.random()
```

Exemple 420. The pyexec4.py script

```
print message
```

If I run this example on my machine I get something like:

```
Using ../../csound.xmg
Csound Version 4.19 (Mar 23 2002)
Embedded Python interpreter version 2.2
```

```
orchname:  pyexec.orc
scorename: pyexec.sco
sorting score ...
        ... done
orch compiler:
11 lines read
        instr   1
Csound Version 4.19 (Mar 23 2002)
displays suppressed

Welcome to Csound!
What sound do you want to hear today, maurizio?
```

then I answer

```
a sound
```

then Csound continues with the normal performance

```
your answer is "a sound"
a private random number: 0.884006
new alloc for instr 1:
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
a private random number: 0.889868
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
a private random number: 0.889868
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
...
```

In the same instrument a message is created in the private namespace and printed, appearing different for each instance.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

pyinit Opcodes

pyinit — Initialize the Python interpreter.

Syntax

```
pyinit
```

Description

In the command-line version of Csound, you must first invoke the *pyinit* opcode in the orchestra header to initialize the Python interpreter, before using any of the other Python opcodes.

But if you use the Python opcodes in the CsoundAC version of Csound, you need not invoke *pyinit*, because CsoundAC automatically initializes the Python interpreter for you. In addition, CsoundAC automatically creates a Python interface to the Csound API, in the form a global instance of the `CsoundAC.CppSound` class named `csound`. Therefore, Python code written in the Csound orchestra has access to the global `csound` object.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

pyrun Opcodes

pyrun — Run a Python statement or block of statements.

Syntax

```
pyrun "statement"

pyruni "statement"

pylrun "statement"

pylruni "statement"

pyrunt ktrigger, "statement"

pylrunt ktrigger, "statement"
```

Description

Execute the specified Python statement at k-time (*pyrun* and *pylrun*) or i-time (*pyruni* and *pylruni*).

The statement is executed in the global environment for pyrun and pyruni or the local environment for pylrun and pylruni.

These opcodes perform no message passing. However, since the statement have access to the main namespace and the private namespace, it can interact with objects previously created in that environment.

The "local" version of the *pyrun* opcodes are useful when the code ran by different instances of an instrument should not interact.

Example of the pyrun Opcode Group

Exemple 421. Orchestra

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;If you're not running CsoundAC you need the following line
;to initialize the python interpreter
;pyinit

pyruni "import random"

instr 1
        ; This message is stored in the main namespace
        ; and is the same for every instance
        pyruni  "message = 'a global random number: %f' % random.random()"
        pyrun   "print message"

        ; This message is stored in the private namespace
        ; and is different for different instances
        pylruni "message = 'a private random number: %f' % random.random()"
        pylrun  "print message"
```

```
endin
```

Exemple 422. Score

```
i1 0 0.1
```

Running this score you should get intermixed pairs of messages from the two instances of instrument 1.

The first message of each pair is stored into the main namespace and so the second instance overwrites the message of the first instance. The result is that first message will be the same for both instances.

The second message is different for the two instances, being stored in the private namespace.

Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

rand

rand — Génère une suite contrôlée de nombres aléatoires.

Description

La sortie est une suite contrôlée de nombres aléatoires entre -*amp* et +*amp*.

Syntaxe

```
ares rand xamp [, iseed] [, isel] [, ioffset]
```

```
kres rand xamp [, iseed] [, isel] [, ioffset]
```

Initialisation

iseed (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et 1 produira une sortie initiale de *kamp* * *iseed*. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système. Avec une valeur négative, la réinitialisation de la graine sera ignorée. La valeur par défaut est 0,5.

isel (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

ioffset (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

Exécution

kamp, xamp -- intervalle sur lequel les nombres aléatoires sont distribués.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de -*kamp* à +*kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp / (racine de 2)*.

Exemples

Voici un exemple de l'opcode rand. Il utilise le fichier *rand.csd* [examples/rand.csd].

Exemple 423. Exemple de l'opcode rand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rand.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Choose a random frequency between 4,100 and 44,100.
  kfreq rand 20000
  kcps = kfreq + 24100

  a1 oscil 30000, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

randh, *randi*

Crédits

Exemple écrit par Kevin Conder.

Grâce à une note de John ffitch, j'ai changé les noms des paramètres.

randh

randh — Génère des nombres aléatoires et les maintient pendant une certaine durée.

Description

Génère des nombres aléatoires et les maintient pendant une certaine durée.

Syntax

```
ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
```

```
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]
```

Initialisation

iseed (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de *kamp* * *iseed*. Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

isize (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

ioffset (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

Exécution

kamp, xamp -- intervalle sur lequel les nombres aléatoires sont distribués.

kcps, xcps -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp / (racine de 2)*.

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randh* maintient chaque nouveau nombre durant le cycle spécifié.

Exemples

Voici un exemple de l'opcode randh. Il utilise le fichier *randh.csd* [examples/randh.csd].

Exemple 424. Exemple de l'opcode randh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Choose a random frequency between 200 and 1000.
  ; Generate new random numbers at 4 Hz.
  ; kamp = 400
  ; kcps = 4
  ; iseed = 0.5
  ; isize = 0
  ; ioffset = 600

  kcps randh 400, 4, 0.5, 0, 600
  printk2 kcps

  a1 oscil 30000, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

rand, *randi*

Crédits

Exemple écrit par Kevin Conder.

randi

rand — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

Syntaxe

```
ares randi xamp, xcps [, iseed] [, isize] [, ioffset]
```

```
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]
```

Initialisation

iseed (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de *kamp* * *iseed*. Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

isize (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

ioffset (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

Exécution

kamp, xamp -- intervalle sur lequel les nombres aléatoires sont distribués.

kcps, xcps -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp / (racine de 2)*.

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randi* produit une interpolation linéaire entre chaque nouveau nombre et le précédent.

Exemples

Voici un exemple de l'opcode randi. Il utilise le fichier *randi.csd* [examples/randi.csd].

Exemple 425. Exemple de l'opcode randi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Choose a random frequency between 4,100 and 44,100.
  ; Generate new random numbers at 10 Hz.
  ; kamp = 40000
  ; kcps = 10
  ; iseed = 0.5
  ; isize = 0
  ; ioffset = 4100

  kcps randi 40000, 10, 0.5, 0, 4100

  a1 oscil 30000, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

rand, *randh*

Crédits

Exemple écrit par Kevin Conder.

random

random — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

Description

Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

Syntaxe

```
ares random kmin, kmax
```

```
ires random imin, imax
```

```
kres random kmin, kmax
```

Initialisation

imin -- limite inférieure de l'intervalle

imax -- limite supérieure de l'intervalle

Exécution

kmin -- limite inférieure de l'intervalle

kmax -- limite supérieure de l'intervalle

L'opcode *random* est semblable à *linrand* et à *trirand* mais parfois je [Gabriel Maldonado] le trouve plus pratique car il permet de fixer arbitrairement les valeurs du minimum et du maximum.

Exemples

Voici un exemple de l'opcode random. Il utilise le fichier *random.csd* [examples/random.csd].

Exemple 426. Exemple de l'opcode random.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o random.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number between 220 and 440.
  kmin init 220
  kmax init 440
  k1 random kmin, kmax

  printks "k1 = %f\\n", 0.1, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie incluera des lignes comme celles-ci :

```
k1 = 414.232056
k1 = 419.393402
k1 = 275.376373
```

Voir Aussi

linrand, *randomh*, *randomi*, *trirand*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

randomh

randomh — Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

Description

Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

Syntaxe

```
ares randomh kmin, kmax, acps
```

```
kres randomh kmin, kmax, kcps
```

Exécution

kmin -- limite inférieure de l'intervalle

kmax -- limite supérieure de l'intervalle

kcps, acps -- taux de génération des points aléatoires

L'opcode *randomh* est semblable à *randh* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

Exemples

Voici un exemple de l'opcode randomh. Il utilise le fichier *randomh.csd* [examples/randomh.csd].

Exemple 427. Exemple de l'opcode randomh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randomh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Choose a random frequency between 220 and 440 Hz.
  ; Generate new random numbers at 10 Hz.
```

```
  kmin = 220
  kmax = 440
  kcps = 10

  k1 randomh kmin, kmax, kcps

  printks "k1 = %f\\n", 0.1, k1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie incluera des lignes comme celles-ci :

```
k1 = 220.000000
k1 = 414.232056
k1 = 284.095184
```

Voir Aussi

randh, *random*, *randomi*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

randomi

randomi — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

Syntaxe

```
ares randomi kmin, kmax, acps
```

```
kres randomi kmin, kmax, kcps
```

Exécution

kmin -- limite inférieure de l'intervalle

kmax -- limite supérieure de l'intervalle

kcps, acps -- taux de génération des points aléatoires

L'opcode *randomi* est semblable à *randi* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

Exemples

Voici un exemple de l'opcode randomi. Il utilise le fichier *randomi.csd* [examples/randomi.csd].

Exemple 428. Exemple de l'opcode randomi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randomi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Choose a random frequency between 220 and 440.
  ; Generate new random numbers at 10 Hz.
  kmin init 220
```

```
  kmax init 440
  kcps init 10

  k1 randomi kmin, kmax, kcps

  printks "k1 = %f\\n", 0.1, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 220.000000
k1 = 414.226196
k1 = 284.101074
```

Voir Aussi

randi, *random*, *randomh*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

rbjeq

rbjeq — Parametric equalizer and filter opcode with 7 filter types, based on algorithm by Robert Bristow-Johnson.

Description

Parametric equalizer and filter opcode with 7 filter types, based on algorithm by Robert Bristow-Johnson.

Syntax

```
ar rbjeq asig, kfco, klvl, kQ, kS[, imode]
```

Initialization

imode (optional, defaults to zero) - sum of:

- 1: skip initialization (should be used in tied, or re-initialized notes only)

and exactly one of the following values to select filter type:

- 0: resonant lowpass filter. kQ controls the resonance: at the cutoff frequency (kfco), the amplitude gain is kQ (e.g. 20 dB for kQ = 10), and higher kQ values result in a narrower resonance peak. If kQ is set to sqrt(0.5) (about 0.7071), there is no resonance, and the filter has a response that is very similar to that of butterlp. If kQ is less than sqrt(0.5), there is no resonance, and the filter has a -6 dB / octave response from about kfco * kQ to kfco. Above kfco, there is always a -12 dB / octave cutoff.

NOTE

The rbjeq lowpass filter is basically the same as ar pareq asig, kfco, 0, kQ, 2 but is faster to calculate.

- 2: resonant highpass filter. The parameters are the same as for the lowpass filter, but the equivalent filter is butterhp if kQ is 0.7071, and "ar pareq asig, kfco, 0, kQ, 1" in other cases.
- 4: bandpass filter. kQ controls the bandwidth, which is kfco / kQ, and must be always less than sr / 2. The bandwidth is measured between -3 dB points (i.e. amplitude gain = 0.7071), beyond which there is a +/- 6 dB / octave slope. This filter type is very similar to ar butterbp asig, kfco, kfco / kQ.
- 6: band-reject filter, with the same parameters as the bandpass filter, and a response similar to that of butterbr.
- 8: peaking EQ. It has an amplitude gain of 1 (0 dB) at 0 Hz and sr / 2, and klvl at the center frequency (kfco). Thus, klvl controls the amount of boost (if it is greater than 1), or cut (if it is less than 1). Setting klvl to 1 results in a flat response. Similarly to the bandpass and band-reject filters, the bandwidth is determined by kfco / kQ (which must be less than sr / 2 again); however, this time it is between sqrt(klvl) points (or, in other words, half the boost or cut in decibels). NOTE: excessively low or high values of klvl should be avoided (especially with 32-bit floats), though the opcode was tested with klvl = 0.01 and klvl = 100. klvl = 0 is always an error, unlike in the case of pareq, which does allow a zero level.

- 10: low shelf EQ, controlled by klvl and kS (kQ is ignored by this filter type). There is an amplitude gain of klvl at zero frequency, while the level of high frequencies (around sr / 2) is not changed. At the corner frequency (kfco), the gain is sqrt(klvl) (half the boost or cut in decibels). The kS parameter controls the steepness of the slope of the frequency response (see below).
- 12: high shelf EQ. Very similar to the low shelf EQ, but affects the high frequency range.

The default value for imode is zero (lowpass filter, initialization not skipped).

Performance

ar -- the output signal.

asig -- the input signal

NOTE

If the input contains silent sections, on Intel CPUs a significant slowdown can occur due to denormals. In such cases, it is recommended to process the input signal with "denorm" opcode before filtering it with rbjeq (and actually many other filters).

kfco -- cutoff, corner, or center frequency, depending on filter type, in Hz. It must be greater than zero, and less than sr / 2 (the range of about sr * 0.0002 to sr * 0.49 should be safe).

klvl -- level (amount of boost or cut), as amplitude gain (e.g. 1: flat response, 4: 12 dB boost, 0.1: 20 dB cut); zero or negative values are not allowed. It is recognized by the peaking and shelving EQ types (8, 10, 12) only, and is ignored by other filters.

kQ -- resonance (also kfco / bandwidth in many filter types). Not used by the shelving EQs (imode = 10 and 12). The exact meaning of this parameter depends on the filter type (see above), but it should be always greater than zero, and usually (kfco / kQ) less than sr / 2.

kS -- shelf slope parameter for shelving filters. Must be greater than zero; a higher value means a steeper slope, with resonance if kS > 1 (however, a too high kS value may make the filter unstable). If kS is set to exactly 1, the shelf slope is as steep as possible without a resonance. Note that the effect of kS - especially if it is greater than 1 - also depends on klvl, and it does not have any well defined unit.

Examples

Here is an example of the *rbjeq* opcode. It uses the file *rbjeq.csd* [examples/rbjeq.csd].

Exemple 429. An example of the rbjeq opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rbjeq.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr      =  44100
```

```
ksmps    =  10
nchnls   =  1

        instr 1

a1      vco2    10000, 155.6            ; sawtooth wave
kfco    expon   8000, p3, 200           ; filter frequency
a1      rbjeq   a1, kfco, 1, kfco * 0.005, 1, 0 ; resonant lowpass
        out a1

        endin

</CsInstruments>
<CsScore>

i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Original algorithm by Robert Bristow-Johnson
Csound orchestra version by Josep M Comajuncosas, Aug 1999
Converted to C (with optimizations and bug fixes) by Istvan Varga, Dec 2002

readclock

readclock — Reads the value of an internal clock.

Description

Reads the value of an internal clock.

Syntax

```
ir readclock inum
```

Initialization

inum -- the number of a clock. There are 32 clocks numbered 0 through 31. All other values are mapped to clock number 32.

ir -- value at i-time, of the clock specified by *inum*

Performance

Between a *clockon* and a *clockoff* opcode, the CPU time used is accumulated in the clock. The precision is machine dependent but is the millisecond range on UNIX and Windows systems. The *readclock* opcde reads the current value of a clock at initialization time.

Examples

Here is an example of the readclock opcode. It uses the file *readclock.csd* [examples/readclock.csd].

Exemple 430. Example of the readclock opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o readclock.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
  ; Start clock #1.
  clockon 1
  ; Do something that keeps Csound busy.
  a1 oscili 10000, 440, 1
```

```
  out a1
  ; Stop clock #1.
  clockoff 1
  ; Print the time accumulated in clock #1.
  i1 readclock 1
  print i1
endin


</CsInstruments>
<CsScore>

; Initialize the function tables.
; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second starting at 0:00.
i 1 0 1
; Play Instrument #1 for one second starting at 0:01.
i 1 1 1
; Play Instrument #1 for one second starting at 0:02.
i 1 2 1
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
instr 1:  i1 = 0.000
instr 1:  i1 = 90.000
instr 1:  i1 = 180.000
```

See Also

clockoff, *clockon*

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
July, 1999

Example written by Kevin Conder.

New in Csound version 3.56

readk

readk — Periodically reads an orchestra control-signal value from an external file.

Description

Periodically reads an orchestra control-signal value from a named external file in a specific format.

Syntax

```
kres readk ifilname, iformat, iprd
```

Initialization

ifilname -- an integer N denoting a file named "readk.N" or a character string (in double quotes, spaces permitted) denoting the external file name. For a string, it may either be a full path name with directory specified or a simple filename. In the later case, the file is sought first in the current directory, then in SSDIR, and finally in SFDIR.

iformat -- specifies the input data format:

- 1 = 8-bit signed integers (char)
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers (plain text)
- 8 = ASCII floats (plain text)

Note that A-law and U-law formats are not available, and that all formats except the last two are binary. The input file should be a "raw", headerless data file.

iprd -- the rate (period) in seconds, rounded to the nearest orchestra control period, at which the signals is read from the input file. A value of 0 implies one control period (the enforced minimum), which will read new values at the orchestra control rate. Longer periods will cause the same values to repeat for more than one control period unless interpolation is used.

Performance

kres -- output of the signal read from ifilname.

This opcode allows a generated control signal value to be read from a named external file. The file should contain no header information but it should contain a regularly sampled time series of control values. For ASCII text formats, the values are assumed to be separated by at least one whitespace character. There may be any number of *readk* opcodes in an instrument or orchestra and they may read from the same or different files.

Examples

Here is an example of the readk opcode. It uses the file *dumpk.csd* [examples/readk.csd].

Exemple 431. Example of the readk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o readk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

0dbfs = 1
; By Andres Cabrera 2008

instr 1
; Read a number from the file every 0.5 seconds
  kfibo readk "fibonacci.txt", 7, 0.5
  kpitchclass = 8 +  ((kfibo % 12)/100)
  printk2 kpitchclass
  kcps = cpspch( kpitchclass )
  printk2 kcps
  a1 oscil 0.5, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See Also

dumpk, *dumpk2*, *dumpk3*, *dumpk4*, *readk2*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

readk2

readk2 — Periodically reads two orchestra control-signal values from an external file.

Description

Periodically reads two orchestra control-signal values from an external file.

Syntax

```
kr1, kr2 readk2 ifilname, iformat, iprd
```

Initialization

ifilname -- an integer N denoting a file named "readk.N" or a character string (in double quotes, spaces permitted) denoting the external file name. For a string, it may either be a full path name with directory specified or a simple filename. In the later case, the file is sought first in the current directory, then in SSDIR, and finally in SFDIR.

iformat -- specifies the input data format:

- 1 = 8-bit signed integers (char)
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers (plain text)
- 8 = ASCII floats (plain text)

Note that A-law and U-law formats are not available, and that all formats except the last two are binary. The input file should be a "raw", headerless data file.

iprd -- the rate (period) in seconds, rounded to the nearest orchestra control period, at which the signals are read from the input file. A value of 0 implies one control period (the enforced minimum), which will read new values at the orchestra control rate. Longer periods will cause the same values to repeat for more than one control period unless interpolation is used.

Performance

kr1, kr2 -- output of the signals read from ifilname.

This opcode allows two generated control signal values to be read from a named external file. The file should contain no header information but it should contain a regularly sampled time series of control values. For binary formats, the individual samples of each signal are interleaved. For ASCII text formats, the values are assumed to be separated by at least one whitespace character. The two "channels" in a sample frame may be on the same line or separated by newline characters, it does not matter. There may be any number of *readk2* opcodes in an instrument or orchestra and they may read from the same or dif-

ferent files.

Examples

See the example for *readk*. The only difference between *readk* and *readk2* is that *readk2* can read two values at a time from the file.

See Also

dumpk, *dumpk2*, *dumpk3*, *dumpk4*, *readk*, *readk3*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

readk3

readk3 — Periodically reads three orchestra control-signal values from an external file.

Description

Periodically reads three orchestra control-signal values from an external file.

Syntax

```
kr1, kr2, kr3 readk3 ifilname, iformat, iprd
```

Initialization

ifilname -- an integer N denoting a file named "readk.N" or a character string (in double quotes, spaces permitted) denoting the external file name. For a string, it may either be a full path name with directory specified or a simple filename. In the later case, the file is sought first in the current directory, then in SSDIR, and finally in SFDIR.

iformat -- specifies the input data format:

- 1 = 8-bit signed integers (char)
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers (plain text)
- 8 = ASCII floats (plain text)

Note that A-law and U-law formats are not available, and that all formats except the last two are binary. The input file should be a "raw", headerless data file.

iprd -- the rate (period) in seconds, rounded to the nearest orchestra control period, at which the signals are read from the input file. A value of 0 implies one control period (the enforced minimum), which will read new values at the orchestra control rate. Longer periods will cause the same values to repeat for more than one control period unless interpolation is used.

Performance

kr1, kr2, kr3 -- output of the signals read from ifilname.

This opcode allows three generated control signal values to be read from a named external file. The file should contain no header information but it should contain a regularly sampled time series of control values. For binary formats, the individual samples of each signal are interleaved. For ASCII text formats, the values are assumed to be separated by at least one whitespace character. The three "channels" in a sample frame may be on the same line or separated by newline characters, it does not matter. There may be any number of *readk3* opcodes in an instrument or orchestra and they may read from the same or dif-

ferent files.

Examples

See the example for *readk*. The only difference between *readk* and *readk3* is that *readk3* can read three values at a time from the file.

See Also

dumpk, *dumpk2*, *dumpk3*, *dumpk4*, *readk*, *readk2*, *readk4*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

readk4

readk4 — Periodically reads four orchestra control-signal values from an external file.

Description

Periodically reads four orchestra control-signal values from an external file.

Syntax

```
kr1, kr2, kr3, kr4 readk4 ifilname, iformat, iprd
```

Initialization

ifilname -- an integer N denoting a file named "readk.N" or a character string (in double quotes, spaces permitted) denoting the external file name. For a string, it may either be a full path name with directory specified or a simple filename. In the later case, the file is sought first in the current directory, then in SSDIR, and finally in SFDIR.

iformat -- specifies the input data format:

- 1 = 8-bit signed integers (char)
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = ASCII long integers (plain text)
- 8 = ASCII floats (plain text)

Note that A-law and U-law formats are not available, and that all formats except the last two are binary. The input file should be a "raw", headerless data file.

iprd -- the rate (period) in seconds, rounded to the nearest orchestra control period, at which the signals are read from the input file. A value of 0 implies one control period (the enforced minimum), which will read new values at the orchestra control rate. Longer periods will cause the same values to repeat for more than one control period unless interpolation is used.

Performance

kr1, kr2, kr3, kr4 -- output of the signals read from ifilname.

This opcode allows four generated control signal values to be read from a named external file. The file should contain no header information but it should contain a regularly sampled time series of control values. For binary formats, the individual samples of each signal are interleaved. For ASCII text formats, the values are assumed to be separated by at least one whitespace character. The four "channels" in a sample frame may be on the same line or separated by newline characters, it does not matter. There may be any number of *readk4* opcodes in an instrument or orchestra and they may read from the same or dif-

ferent files.

Examples

See the example for *readk*. The only difference between *readk* and *readk4* is that *readk4* can read four values at a time from the file.

See Also

dumpk, *dumpk2*, *dumpk3*, *dumpk4*, *readk*, *readk2*, *readk3*

Credits

By: John ffitch and Barry Vercoe

1999 or earlier

reinit

reinit — Suspends a performance while a special initialization pass is executed.

Description

Suspends a performance while a special initialization pass is executed.

Whenever this statement is encountered during a p-time pass, performance is temporarily suspended while a special Initialization pass, beginning at *label* and continuing to *rireturn* or *endin*, is executed. Performance will then be resumed from where it left off.

Syntax

```
reinit label
```

Examples

The following statements will generate an exponential control signal whose value moves from 440 to 880 exactly ten times over the duration p3. They use the file *reinit.csd* [examples/reinit.csd].

Exemple 432. Example of the reinit opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reinit.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
        timout 0, p3/10, contin
        reinit reset

contin:
        kLine expon 440, p3/10, 880
        aSig oscil 10000, kLine, 1
        out aSig
        rireturn

endin


</CsInstruments>
<CsScore>

f1 0 4096 10 1
```

```
i1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See Also

rigoto, *rireturn*

release

release — Indicates whether a note is in its « release » stage.

Description

Provides a way of knowing when a note off message for the current note is received. Only a noteoff message with the same MIDI note number as the one which triggered the note will be reported by *release*.

Syntax

```
kflag release
```

Performance

kflag -- indicates whether the note is in its « release » stage. (1 if a note off is received, otherwise 0)

release outputs current note state. If current note is in the « release » stage (i.e. if its duration has been extended with *xtratim* opcode and if it has only just deactivated), then the *kflag* output argument is set to 1. Otherwise (in sustain stage of current note), *kflag* is set to 0.

This opcode is useful for implementing complex release-oriented envelopes. When used in conjunction with *xtratim* it can provide an alternative to the hard-coded behaviour of the "r" opcodes (*linsegr*, *expsegr* et al), where release time is set to the longest time specified in the active instrument.

Examples

See the examples for *xtratim*.

See Also

xtratim

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.47

remoteport

remoteport — Defines the port for use with the remote system.

Description

Defines the port for use with the *insremot*, *midremot*, *insglobal* and *midglobal* opcodes.

Syntax

```
remoteport iportnum
```

Initialization

iportnum -- number of the port to be used. If zero or negative the default port 40002 is selected.

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Novemer, 2006

New in Csound version 5.05

remove

remove — Supprime la définition d'un instrument.

Description

Supprime la définition d'un instrument tant qu'il n'est pas utilisé.

Syntaxe

```
remove insnum
```

Initialisation

insnum -- numéro ou nom de l'instrument à effacer

Exécution

Tant que l'instrument indiqué n'est pas actif, *remove* efface l'instrument et la mémoire qui lui est associée. A employer avec précaution car son utilisation peut conduire à un plantage dans certains cas.

Crédits

Auteur : John ffitch
Université de Bath/Codemist Ltd.
Bath, UK
Juin, 2006

Nouveau dans la version 5.04 de Csound

repluck

repluck — Physical model of the plucked string.

Description

repluck is an implementation of the physical model of the plucked string. A user can control the pluck point, the pickup point, the filter, and an additional audio signal, *axcite*. *axcite* is used to excite the 'string'. Based on the Karplus-Strong algorithm.

Syntax

```
ares repluck iplk, kamp, icps, kpick, krefl, axcite
```

Initialization

iplk -- The point of pluck is *iplk*, which is a fraction of the way up the string (0 to 1). A pluck point of zero means no initial pluck.

icps -- The string plays at *icps* pitch.

Performance

kamp -- Amplitude of note.

kpick -- Proportion of the way along the string to sample the output.

krefl -- the coefficient of reflection, indicating the lossiness and the rate of decay. It must be strictly between 0 and 1 (it will complain about both 0 and 1).

Performance

axcite -- A signal which excites the string.

Examples

Here is an example of the repluck opcode. It uses the file *repluck.csd* [examples/repluck.csd].

Exemple 433. Example of the repluck opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o repluck.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iplk = 0.75
  kamp = 30000
  icps = 220
  kpick = 0.75
  krefl = 0.5
  axcite oscil 1, 1, 1

  apluck repluck iplk, kamp, icps, kpick, krefl, axcite

  out apluck
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

wgpluck2

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
1997

New in version 3.47

reson

reson — Un filtre à résonance du second ordre.

Description

Un filtre à résonance du second ordre.

Syntaxe

```
ares reson asig, kcf, kbw [, iscl] [, iskip]
```

Initialisation

iscl (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

ares -- le signal de sortie au taux audio.

asig -- le signal d'entrée au taux audio.

kcf -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

kbw -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

reson est un filtre de second ordre dans lequel *kcf* contrôle la fréquence centrale, ou position fréquentielle de la crête de la réponse, et *kbw* contrôle sa largeur de bande (la différence en fréquence entre les points haut et bas à mi-puissance).

Exemples

Voici un exemple de l'opcode reson. Il utilise le fichier *reson.csd* [examples/reson.csd].

Exemple 434. Exemple de l'opcode reson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reson.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a sine waveform.
  asine buzz 15000, 440, 3, 1

  ; Vary the cut-off frequency from 220 to 1280.
  kcf line 220, p3, 1320
  kbw init 20

  ; Run the sine through a resonant filter.
  ares reson asine, kcf, kbw

  ; Give the filtered signal the same amplitude
  ; as the original signal.
  a1 balance ares, asine
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, an ordinary sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 4 seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *port*, *portk*, *resonk*, *tone*, *tonek*

Crédits

Exemple écrit par Kevin Conder.

resonk

resonk — Un filtre à résonance du second ordre.

Description

Un filtre à résonance du second ordre.

Syntaxe

```
kres resonk ksig, kcf, kbw [, iscl] [, iskip]
```

Initialisation

iscl (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

kcf -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

kbw -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

resonk est semblable à *reson* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *tone*, *tonek*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

resonr

resonr — A bandpass filter with variable frequency response.

Description

Implementations of a second-order, two-pole two-zero bandpass filter with variable frequency response.

Syntax

```
ares resonr asig, kcf, kbw [, iscl] [, iskip]
```

Initialization

The optional initialization variables for *resonr* are identical to the i-time variables for *reson*.

iscl (optional, default=0) -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

iskip (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- input signal to be filtered

kcf -- cutoff or resonant frequency of the filter, measured in Hz

kbw -- bandwidth of the filter (the Hz difference between the upper and lower half-power points)

resonr and *resonz* are variations of the classic two-pole bandpass resonator (*reson*). Both filters have two zeroes in their transfer functions, in addition to the two poles. *resonz* has its zeroes located at z = 1 and z = -1. *resonr* has its zeroes located at +sqrt(*R*) and -sqrt(*R*), where *R* is the radius of the poles in the complex z-plane. The addition of zeroes to *resonr* and *resonz* results in the improved selectivity of the magnitude response of these filters at cutoff frequencies close to 0, at the expense of less selectivity of frequencies above the cutoff peak.

resonr and *resonz* are very close to constant-gain as the center frequency is swept, resulting in a more efficient control of the magnitude response than with traditional two-pole resonators such as *reson*.

resonr and *resonz* produce a sound that is considerably different from *reson*, especially for lower center frequencies; trial and error is the best way of determining which resonator is best suited for a particular application.

Examples

Here is an example of the resonr and resonz opcodes. It uses the file *resonr.csd* [examples/resonr.csd].

Exemple 435. Example of the resonr and resonz opcodes.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o resonr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
 ; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
  ; The outputs of reson, resonr, and resonz are scaled by coefficients
  ; specified in the score, so that each filter can be heard on its own
  ; from the same instrument.

  sr = 44100
  kr = 4410
  ksmps = 10
  nchnls = 1

instr 1

  idur     =        p3
  ibegfreq =        p4                              ; beginning of sweep frequency
  iendfreq =        p5                              ; ending of sweep frequency
  ibw      =        p6                              ; bandwidth of filters in Hz
  ifreq    =        p7                              ; frequency of gbuzz that is to be filtered
  iamp     =        p8                              ; amplitude to scale output by
  ires     =        p9                              ; coefficient to scale amount of reson in output
  iresr    =        p10                             ; coefficient to scale amount of resonr in output
  iresz    =        p11                             ; coefficient to scale amount of resonz in output

 ; Frequency envelope for reson cutoff
  kfreq    linseg ibegfreq, idur * .5, iendfreq, idur * .5, ibegfreq

 ; Amplitude envelope to prevent clicking
  kenv     linseg 0, .1, iamp, idur - .2, iamp, .1, 0

 ; Number of harmonics for gbuzz scaled to avoid aliasing
  iharms   =        (sr*.4)/ifreq

  asig     gbuzz 1, ifreq, iharms, 1, .9, 1      ; "Sawtooth" waveform
  ain      =        kenv * asig                     ; output scaled by amp envelope
  ares     reson ain, kfreq, ibw, 1
  aresr    resonr ain, kfreq, ibw, 1
  aresz    resonz ain, kfreq, ibw, 1

           out ares * ires + aresr * iresr + aresz * iresz

endin


</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25                                ; cosine table for gbuzz generator

i1  0 10 1 3000 200 100 4000 1 0 0                 ; reson  output with bw = 200
i1 10 10 1 3000 200 100 4000 0 1 0                 ; resonr output with bw = 200
i1 20 10 1 3000 200 100 4000 0 0 1                 ; resonz output with bw = 200
i1 30 10 1 3000  50 200 8000 1 0 0                 ; reson  output with bw = 50
i1 40 10 1 3000  50 200 8000 0 1 0                 ; resonr output with bw = 50
i1 50 10 1 3000  50 200 8000 0 0 1                 ; resonz output with bw = 50
e


</CsScore>
</CsoundSynthesizer>
```

Technical History

resonr and *resonz* were originally described in an article by Julius O. Smith and James B. Angell.[1] Smith and Angell recommended the *resonz* form (zeros at +1 and -1) when computational efficiency was the main concern, as it has one less multiply per sample, while *resonr* (zeroes at + and - the square root of the pole radius R) was recommended for situations when a perfectly constant-gain center peak was required.

Ken Steiglitz, in a later article [2], demonstrated that *resonz* had constant gain at the true peak of the filter, as opposed to *resonr*, which displayed constant gain at the pole angle. Steiglitz also recommended *resonz* for its sharper notches in the gain curve at zero and Nyquist frequency. Steiglitz's recent book [3] features a thorough technical discussion of *reson* and *resonz*, while Dodge and Jerse's textbook [4] illustrates the differences in the response curves of *reson* and *resonz*.

References

1. Smith, Julius O. and Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Winter 1982.

2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Winter 1994.

3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.

4. Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nd edition, pp. 211-214.

See Also

resonz

Credits

Author: Sean Costello
Seattle, Washington
1999

New in Csound version 3.55

resonx

resonx — Emule une série de filtres utilisant l'opcode *reson*.

Description

resonx est équivalent à un filtre constitué de plusieurs couches de filtres *reson* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de k cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

Syntaxe

```
ares resonx asig, kcf, kbw [, inumlayer] [, iscl] [, iskip]
```

Initialisation

inumlayer (optional) -- (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

iscl (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

asig -- signal d'entrée

kcf -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

kbw -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

Voir Aussi

atonex, *tonex*

Crédits

Auteur : Gabriel Maldonado (adapté par John ffitch)
Italie

Nouveau dans la version 3.49 de Csound

resonxk

resonxk — Control signal resonant filter stack.

Description

resonxk is equivalent to a group of resonk filters, with the same arguments, serially connected. Using a stack of a larger number of filters allows a sharper cutoff.

Syntax

```
kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]
```

Initialization

inumlayer - number of elements of filter stack. Default value is 4. Maximum value is 10

iscl (optional, default=0) - coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than kcf are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. (This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise.) A zero value signifies no scaling of the signal, leaving that to some later adjustment (see balance). The default value is 0.

istor (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

kres - output signal

ksig - input signal

kcf - the center frequency of the filter, or frequency position of the peak response.

kbw - bandwidth of the filter (the Hz difference between the upper and lower half-power points)

resonxk is a lot faster than using individual instances in Csound orchestra of the old opcodes, because only one initialization and 'k' cycle are needed at a time, and the audio loop falls enterely inside the cache memory of processor.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

resony

resony — A bank of second-order bandpass filters, connected in parallel.

Description

A bank of second-order bandpass filters, connected in parallel.

Syntax

```
ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]
```

Initialization

inum -- number of filters

isepmode (optional, default=0) -- if *isepmode* = 0, the separation of center frequencies of each filter is generated logarithmically (using octave as unit of measure). If *isepmode* not equal to 0, the separation of center frequencies of each filter is generated linearly (using Hertz). Default value is 0.

iscl (optional, default=0) -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. (This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise.) A zero value signifies no scaling of the signal, leaving that to some later adjustment (e.g. *balance*). The default value is 0.

iskip (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

asig -- audio input signal

kbf -- base frequency, i.e. center frequency of lowest filter in Hz

kbw -- bandwidth in Hz

ksep -- separation of the center frequency of filters in octaves

resony is a bank of second-order bandpass filters, with k-rate variant frequency separation, base frequency and bandwidth, connected in parallel (i.e. the resulting signal is a mix of the output of each filter). The center frequency of each filter depends of *kbf* and *ksep* variables. The maximum number of filters is set to 100.

Examples

Here is an example of the resony opcode. It uses the file *resony.csd* [examples/resony.csd], and *beats.wav* [examples/beats.wav].

Exemple 436. Example of the resony opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o resony.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 32000, 220, 1

  ; Vary the base frequency from 60 to 600 Hz.
  kbf line 60, p3, 600
  kbw = 50
  inum = 2
  ksep = 1
  isepmode = 0
  iscl = 1

  a1 resony asig, kbf, kbw, inum, ksep, isepmode, iscl

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy
1999

Example written by Kevin Conder.

New in Csound version 3.56

resonz

resonz — A bandpass filter with variable frequency response.

Description

Implementations of a second-order, two-pole two-zero bandpass filter with variable frequency response.

Syntax

```
ares resonz asig, kcf, kbw [, iscl] [, iskip]
```

Initialization

The optional initialization variables for *resonr* and *resonz* are identical to the i-time variables for *reson*.

iskip -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

iscl -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

Performance

resonr and *resonz* are variations of the classic two-pole bandpass resonator (*reson*). Both filters have two zeroes in their transfer functions, in addition to the two poles. *resonz* has its zeroes located at z = 1 and z = -1. *resonr* has its zeroes located at +sqrt(*R*) and -sqrt(*R*), where *R* is the radius of the poles in the complex z-plane. The addition of zeroes to *resonr* and *resonz* results in the improved selectivity of the magnitude response of these filters at cutoff frequencies close to 0, at the expense of less selectivity of frequencies above the cutoff peak.

resonr and *resonz* are very close to constant-gain as the center frequency is swept, resulting in a more efficient control of the magnitude response than with traditional two-pole resonators such as *reson*.

resonr and *resonz* produce a sound that is considerably different from *reson*, especially for lower center frequencies; trial and error is the best way of determining which resonator is best suited for a particular application.

asig -- input signal to be filtered

kcf -- cutoff or resonant frequency of the filter, measured in Hz

kbw -- bandwidth of the filter (the Hz difference between the upper and lower half-power points)

Technical History

resonr and *resonz* were originally described in an article by Julius O. Smith and James B. Angell.[1]

Smith and Angell recommended the *resonz* form (zeros at +1 and -1) when computational efficiency was the main concern, as it has one less multiply per sample, while *resonr* (zeroes at + and - the square root of the pole radius R) was recommended for situations when a perfectly constant-gain center peak was required.

Ken Steiglitz, in a later article [2], demonstrated that *resonz* had constant gain at the true peak of the filter, as opposed to *resonr*, which displayed constant gain at the pole angle. Steiglitz also recommended *resonz* for its sharper notches in the gain curve at zero and Nyquist frequency. Steiglitz's recent book [3] features a thorough technical discussion of *reson* and *resonz*, while Dodge and Jerse's textbook [4] illustrates the differences in the response curves of *reson* and *resonz*.

References

1. Smith, Julius O. and Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Winter 1982.

2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Winter 1994.

3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.

4. Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nd edition, pp. 211-214.

See Also

resonr

Credits

Author: Sean Costello
Seattle, Washington
1999

New in Csound version 3.55

resyn

resyn — Streaming partial track additive synthesis with cubic phase interpolation with pitch control and support for timescale-modified input

Description

The resyn opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials). It resynthesises the signal using linear amplitude and cubic phase interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls. Resyn is a modified version of sinsyn, allowing for the resynthesis of data with pitch and timescale changes.

Syntax

```
asig resyn fin, kscal, kpitch, kmaxtracks, ifn
```

Performance

asig -- output audio rate signal

fin -- input pv stream in TRACKS format

kscal -- amplitude scaling

kpitch -- pitch scaling

kmaxtracks -- max number of tracks in resynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

ifn -- function table containing one cycle of a sinusoid (sine or cosine)

Examples

Exemple 437. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
      aout  resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis with pitch shifting.

Credits

Author: Victor Lazzarini;
June 2005

New plugin in version 5

November 2004.

reverb

reverb — Reverberates an input signal with a « natural room » frequency response.

Description

Reverberates an input signal with a « natural room » frequency response.

Syntax

```
ares reverb asig, krvt [, iskip]
```

Initialization

iskip (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

Performance

krvt -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

A standard *reverb* unit is composed of four *comb* filters in parallel followed by two *alpass* units in series. Loop times are set for optimal « natural room response. » Core storage requirements for this unit are proportional only to the sampling rate, each unit requiring approximately 3K words for every 10KC. The *comb*, *alpass*, *delay*, *tone* and other Csound units provide the means for experimenting with alternate reverberator designs.

Since output from the standard *reverb* will begin to appear only after 1/20 second or so of delay, and often with less than three-fourths of the original power, it is normal to output both the source and the reverberated signal. If *krvt* is inadvertently set to a non-positive number, *krvt* will be reset automatically to 0.01. (New in Csound version 4.07.) Also, since the reverberated sound will persist long after the cessation of source events, it is normal to put *reverb* in a separate instrument to which sound is passed via a *global variable*, and to leave that instrument running throughout the performance.

Examples

Here is an example of the reverb opcode. It uses the file *reverb.csd* [examples/reverb.csd].

Exemple 438. Example of the reverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reverb.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; init an audio receiver/mixer
ga1 init 0

; Instrument #1. (there may be many copies)
instr 1
  ; generate a source signal
  a1 oscili 7000, cpspch(p4), 1
  ; output the direct sound
  out a1
  ; and add to audio receiver
  ga1 = ga1 + a1
endin

; (highest instr number executed last)
instr 99
  ; reverberate whatever is in ga1
  a3 reverb ga1, 1.5
  ; and output the result
  out a3
  ; empty the receiver for the next pass
  ga1 = 0
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=6.00
i 1 0 0.1 6.00
; Play Instrument #1 for a tenth of a second, p4=6.02
i 1 1 0.1 6.02
; Play Instrument #1 for a tenth of a second, p4=6.04
i 1 2 0.1 6.04
; Play Instrument #1 for a tenth of a second, p4=6.06
i 1 3 0.1 6.06

; Make sure the reverb remains active.
i 99 0 6
e


</CsScore>
</CsoundSynthesizer>
```

See Also

alpass, *comb*, *valpass*, *vcomb*

Credits

Author: William « Pete » Moss
University of Texas at Austin
Austin, Texas USA
January 2002

reverb2

reverb2 — Same as the nreverb opcode.

Description

Same as the *nreverb* opcode.

Syntax

```
ares reverb2 asig, ktime, khdif [, iskip] [,inumCombs] \
      [, ifnCombs] [, inumAlpas] [, ifnAlpas]
```

reverbsc

reverbsc — 8 delay line stereo FDN reverb, based on work by Sean Costello

Description

8 delay line stereo FDN reverb, with feedback matrix based upon physical modeling scattering junction of 8 lossless waveguides of equal characteristic impedance. Based on Csound orchestra version by Sean Costello.

Syntax

```
aoutL, aoutR reverbsc ainL, ainR, kfblvl, kfco[, israte[, ipitchm[, iskip]]]
```

Initialization

israte (optional, defaults to the orchestra sample rate) -- assume a sample rate of israte. This is normally set to sr, but a different setting can be useful for special effects.

ipitchm (optional, defaults to 1) -- depth of random variation added to delay times, in the range 0 to 10. The default is 1, but this may be too high and may need to be reduced for held pitches such as piano tones.

iskip (optional, defaults to zero) -- if non-zero, initialization of the opcode is skipped, whenever possible.

Performance

aoutL, aoutR -- output signals for left and right channel

ainL, ainR -- left and right channel input. Note that having an input signal on either the left or right channel only will still result in having reverb output on both channels, making this unit more suitable for reverberating stereo input than the freeverb opcode.

kfblvl -- feedback level, in the range 0 to 1. 0.6 gives a good small "live" room sound, 0.8 a small hall, and 0.9 a large hall. A setting of exactly 1 means infinite length, while higher values will make the opcode unstable.

kfco -- cutoff frequency of simple first order lowpass filters in the feedback loop of delay lines, in Hz. Should be in the range 0 to israte/2 (not sr/2). A lower value means faster decay in the high frequency range.

Examples

Here is an example of the *reverbsc* opcode. It uses the file *reverbsc.csd* [examples/reverbsc.csd].

Exemple 439. An example of the reverbsc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reverbsc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr        =  48000
ksmps     =  32
nchnls    =  2
0dbfs     =  1

        instr 1
a1      vco2 0.85, 440, 10
kfrq    port 100, 0.004, 20000
a1      butterlp a1, kfrq
a2      linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1      =  a1 * a2
a2      =  a1 * p5
a1      =  a1 * p4
        denorm a1, a2
aL, aR  reverbsc a1, a2, 0.85, 12000, sr, 0.5, 1
        outs a1 + aL, a2 + aR
        endin

</CsInstruments>
<CsScore>
i 1 0 1 0.71 0.71
i 1 1 1 0 1
i 1 2 1 -0.71 0.71
i 1 3 1 1 0
i 1 4 4 0.71 0.71
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Istvan Varga
2005

rewindscore

rewindscore — Rewinds the playback position of the current score performance.

Description

Rewinds the playback position of the current score performance..

Syntax

```
rewindscore
```

Examples

Here is an example of the rewindscore opcode.

Exemple 440. Example of the rewindscore opcode.

```
instr 1
    rewindscore
    endin
```

See Also

setscorepos

Credits

Author: Victor Lazzarini
2008

New in Csound version 5.09

rezzy

rezzy — A resonant low-pass filter.

Description

A resonant low-pass filter.

Syntax

```
ares rezzy asig, xfco, xres [, imode, iskip]
```

Initialization

imode (optional, default=0) -- high-pass or low-pass mode. If zero, *rezzy* is low-pass. If not zero, *rezzy* is high-pass. Default value is 0. (New in Csound version 3.50) *iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

Performance

asig -- input signal

xfco -- filter cut-off frequency in Hz. As of version 3.50, may i-,k-, or a-rate.

xres -- amount of resonance. Values of 1 to 100 are typical. Resonance should be one or greater. As of version 3.50, may a-rate, i-rate, or k-rate.

rezzy is a resonant low-pass filter created empirically by Hans Mikelson.

Examples

Here is an example of the rezzy opcode. It uses the file *rezzy.csd* [examples/rezzy.csd].

Exemple 441. Example of the rezzy opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rezzy.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 32000, 220, 1

  ; Vary the filter-cutoff frequency from .2 to 2 KHz.
  kfco line 200, p3, 2000

  ; Set the resonance amount.
  kres init 25

  a1 rezzy asig, kfco, kres

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

See Also

biquad, *moogvcf*

Credits

Author: Hans Mikelson
October 1998

Example written by Kevin Conder.

New in Csound version 3.49

rigoto

rigoto — Transfers control during a reinit pass.

Description

Similar to *igoto*, but effective only during a *reinit* pass (i.e., no-op at standard i-time). This statement is useful for bypassing units that are not to be reinitialized.

Syntax

```
rigoto label
```

See Also

cigoto, *igoto*, *reinit*, *rireturn*

rireturn

rireturn — Terminates a reinit pass.

Description

Terminates a *reinit* pass (i.e., no-op at standard i-time). This statement, or an *endin*, will cause normal performance to be resumed.

Syntax

```
rireturn
```

Examples

The following statements will generate an exponential control signal whose value moves from 440 to 880 exactly ten times over the duration p3. They use the file *reinit.csd* [examples/reinit.csd].

Exemple 442. Example of the rireturn opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reinit.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
        timout 0, p3/10, contin
        reinit reset

contin:
        kLine expon 440, p3/10, 880
        aSig oscil 10000, kLine, 1
        out aSig
        rireturn

endin


</CsInstruments>
<CsScore>

f1 0 4096 10 1

i1 0 10
e
```

```
</CsScore>
</CsoundSynthesizer>
```

See Also

reinit, *rigoto*

rms

rms — Determines the root-mean-square amplitude of an audio signal.

Description

Determines the root-mean-square amplitude of an audio signal. It low-pass filters the actual value, to average in the manner of a VU meter.

Syntax

```
kres rms asig [, ihp] [, iskip]
```

Initialization

ihp (optional, default=10) -- half-power point (in Hz) of a special internal low-pass filter. The default value is 10.

iskip (optional, default=0) -- initial disposition of internal data space (see *reson*). The default value is 0.

Performance

asig -- input audio signal

kres -- low-pass filtered rms value of the input signal

rms output values *kres* will trace the root-mean-square value of the audio input *asig*. This unit is not a signal modifier, but functions rather as a signal power-gauge. It uses an internal low-pass filter to make the response smoother. *ihp* can be used to control this smoothing. The higher the value, the "snappier" the measurement.

This opcode can also be used as an evelope follower.

The *kres* output from this opcode is given in aplitude and depends on *0dbfs*. If you want the output in decibels, you can use *dbamp*

Examples

```
arms rms    asig ; get rms value of signal asig
```

Here is an example of the rms opcode. It uses the file *rms.csd* [examples/rms.csd].

Exemple 443. Example of the rms opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d  -m0   ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rms.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

0dbfs = 1

FLpanel "rms", 400, 100, 50, 50
    gkrmstext, gihrmstext FLtext "Rms", -100, 0, 0.1, 3, 110, 30, 60, 50
    gkihp, gihandle FLtext "ihp", 0, 10, 0.05, 1, 100, 30, 220, 50
    gkrmsslider, gihrmsslider FLslider "", -60, -0.5, -1, 5, -1, 380, 20, 10, 10

FLpanelEnd
FLrun


FLsetVal_i 5, gihandle
; Instrument #1.
instr 1
  a1 inch 1

label:
  kval rms a1, i(gkihp)  ;measures rms of input channel 1
rireturn

  kval = dbamp(kval) ; convert to db full scale
  printk 0.5, kval
  FLsetVal 1, kval, gihrmsslider   ;update the slider and text values
  FLsetVal 1, kval, gihrmstext
  knewihp changed gkihp   ; reinit when ihp text has changed
  if (knewihp == 1) then
    reinit label  ;needed because ihp is an i-rate parameter
  endif
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute
i 1 0 60
e


</CsScore>
</CsoundSynthesizer>
```

See Also

balance, *gain*

rnd

rnd — Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

Description

Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

Syntaxe

```
rnd(x) (taux-i ou -k seulement)
```

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

Exécution

Retourne un nombre aléatoire dans l'intervalle unipolaire allant de 0 à *x*.

Exemples

Voici un exemple de l'opcode rnd. Il utilise le fichier *rnd.csd* [examples/rnd.csd].

Exemple 444. Exemple de l'opcode rnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number from 0 to 1.
  i1 = rnd(1)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
```

```
i 1 1 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
rnd at i-rate: 0.973500   rnd at k-rate: 0.139405
rnd at i-rate: 0.973500   rnd at k-rate: 0.040065
rnd at i-rate: 0.973500   rnd at k-rate: 0.412845
rnd at i-rate: 0.973500   rnd at k-rate: 0.440650
rnd at i-rate: 0.973500   rnd at k-rate: 0.663581
rnd at i-rate: 0.973500   rnd at k-rate: 0.876723
rnd at i-rate: 0.973500   rnd at k-rate: 0.302459
rnd at i-rate: 0.973500   rnd at k-rate: 0.398580
rnd at i-rate: 0.973500   rnd at k-rate: 0.448875
rnd at i-rate: 0.973500   rnd at k-rate: 0.907728
```

Voir Aussi

birnd

Crédits

Auteur: Barry L. Vercoe
MIT
Cambridge, Massachussetts
1997

Exemple original écrit par Kevin Conder. Modifié par John Harrison.

rnd31

rnd31 — Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée.

Description

Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée. Ces unités sont portables, c-à-d qu'avec la même valeur de graine on obtiendra la même séquence aléatoire sur tous les systèmes. La distribution des nombres aléatoires générés peut être changée au taux-k.

Syntaxe

```
ax rnd31 kscl, krpow [, iseed]
```

```
ix rnd31 iscl, irpow [, iseed]
```

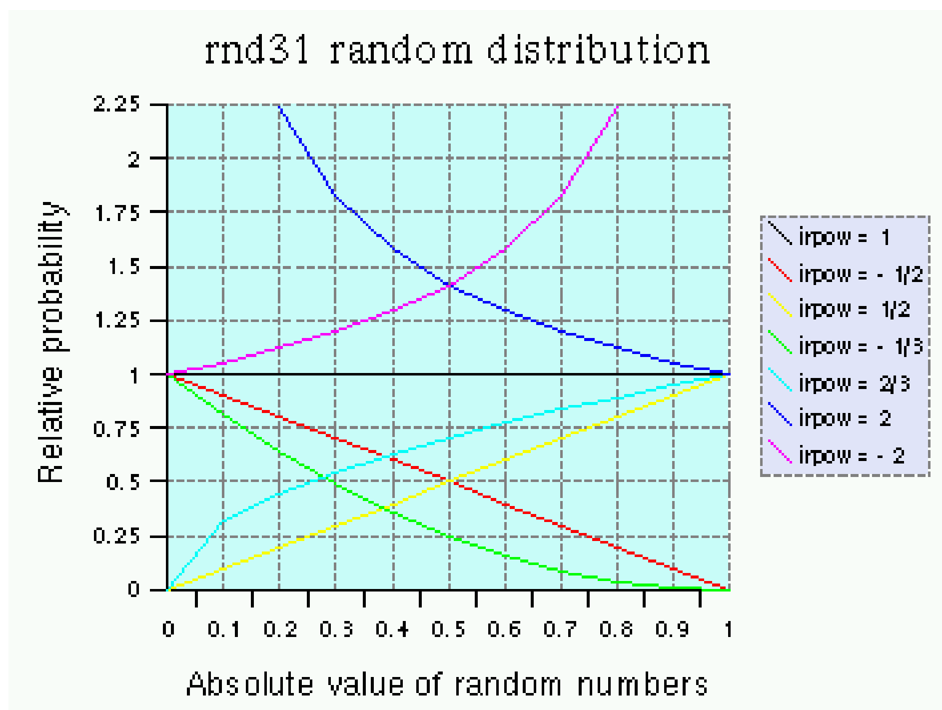
```
kx rnd31 kscl, krpow [, iseed]
```

Initialisation

ix -- valeur de sortie au taux-i.

iscl -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre -*iscl* et *iscl*.

irpow -- contrôle la distribution des nombres aléatoires. Si *irpow* est positif, la distribution aléatoire (x compris entre -1 et 1) est $abs(x)^{((1\ /\ irpow)\ -\ 1)}$; pour des valeurs négatives de *irpow*, elle vaut $(1 - abs(x))^{((-1\ /\ irpow)\ -\ 1)}$. En fixant *irpow* à -1, 0 ou 1 on obtiendra une distribution uniforme (c'est aussi plus rapide à calculer).



Un graphique des distributions pour différentes valeurs de irpow.

iseed (facultatif, par défaut=0) -- valeur de la graine pour le générateur de nombres aléatoires (nombre entier positif compris entre 1 et 2147483646 (2^{31} - 2)). Avec une valeur nulle ou négative la graine est prise à partir de l'horloge du système (c'est le comportement par défaut). Une graine à partir de l'horloge du système nous garantit la génération de séquences aléatoires différentes, même si plusieurs opcodes aléatoires sont appelés dans un temps très court.

Dans les versions de taux-a et de taux-k la graine est fixée à l'initialisation de l'opcode. Avec une sortie de taux-i, si la graine est nulle ou négative, elle sera prise à partir de l'horloge du système lors du premier appel, puis retournera la valeur suivante de la séquence aléatoire lors des appels successifs ; les valeurs positives de la graine sont fixées à tous les appels de taux-i. La graine est locale pour les unités de taux-a et -k, et globale pour les unités de taux-i.

Notes

- bien que des valeurs de graines allant jusqu'à 2147483646 soient permises, il est recommandé d'utiliser des nombres plus petits (< 1000000) pour des raisons de portabilité, car les grands nombres peuvent être arrondis à une valeur différente si l'on utilise des nombres flottants sur 32 bit.
- *rnd31* au taux-i avec une graine positive produira toujours la même valeur en sortie (ce n'est pas un bogue). Pour obtenir des valeurs différentes, fixer la graine à 0 dans les appels successifs, ce qui retournera la valeur suivante de la séquence aléatoire.

Exéution

ax -- valeur de sortie au taux-a.

kx -- valeur de sortie au taux-k.

kscl -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre -*kscl* et *kscl*. Semblable à *iscl*, mais il peut être modifié au taux-k.

krpow -- contrôle la distribution des nombres aléatoires. Semblable à *irpow*, mais il peut être modifié au taux-k.

Exemples

Voici un exemple de l'opcode rnd31. Il utilise le fichier *rnd31.csd* [examples/rnd31.csd].

Exemple 445. Exemple de l'opcode rnd31 au taux-a.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create random numbers at a-rate in the range -2 to 2 with
  ; a triangular distribution, seed from the current time.
  a31 rnd31 2, -0.5

  ; Use the random numbers to choose a frequency.
  afreq = a31 * 500 + 100

  a1 oscil 30000, afreq, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode rnd31 au taux-k. Il utilise le fichier *rnd31_krate.csd* [examples/rnd31_krate.csd].

Exemple 446. Exemple de l'opcode rnd31 au taux-k.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_krate.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create random numbers at k-rate in the range -1 to 1
  ; with a uniform distribution, seed=10.
  k1 rnd31 1, 0, 10

  printks "k1=%f\\n", 0.1, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1= 0.112106
k1=-0.274665
k1= 0.403933
```

Here is an example of the rnd31 opcode that uses the number 7 as a seed value. It uses the file *rnd31_seed7.csd* [examples/rnd31_seed7.csd].

Exemple 447. An example of the rnd31 opcode that uses the number 7 as a seed value.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_seed7.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; i-rate random numbers with linear distribution, seed=7.
  ; (Note that the seed was used only in the first call.)
  i1 rnd31 1, 0.5, 7
  i2 rnd31 1, 0.5
  i3 rnd31 1, 0.5

  print i1
  print i2
  print i3
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  i1 = -0.649
instr 1:  i2 = -0.761
instr 1:  i3 =  0.677
```

Voici un exemple de l'opcode rnd31 qui utilise l'horloge du système comme graine. Il utilise le fichier *rnd31_time.csd* [examples/rnd31_time.csd].

Exemple 448. Eexemple de l'opcode rnd31 qui utilise l'horloge du système comme graine.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_time.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; i-rate random numbers with linear distribution,
  ; seeding from the current time. (Note that the seed
  ; was used only in the first call.)
  i1 rnd31 1, 0.5, 0
  i2 rnd31 1, 0.5
  i3 rnd31 1, 0.5

  print i1
  print i2
  print i3
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  i1 = -0.691
instr 1:  i2 = -0.686
instr 1:  i3 = -0.358
```

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

round

round — Retourne la valeur entière la plus proche de *x* ; si la partie décimale de *x* vaut exactement 0.5, la direction de l'arrondi est indéfinie.

Description

La valeur entière la plus proche de *x* ; si la partie décimale de *x* vaut exactement 0.5, la direction de l'arrondi est indéfinie.

Syntaxe

```
round(x) (des arguments de taux-i, -k ou -a sont permis)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur réalisent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Voir Aussi

abs, *exp*, *int*, *log*, *log10*, *i*, *sqrt*

Crédits

Auteur : Istvan Varga
Nouveau dans Csound 5
2005

rspline

rspline — Génère des courbes splines aléatoires.

Description

Génère des courbes splines aléatoires.

Syntaxe

```
ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
```

```
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax
```

Exécution

kres, ares -- Signal de sortie.

xrangeMin, xrangeMax -- Intervalle des valeurs des points générés aléatoirement.

kcpsMin, kcpsMax -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

rspline (générateur de courbe spline aléatoire) est semblable à *jspline* mais l'intervalle de sortie est défini par deux valeurs limites. De plus, ici, l'intervalle de sortie réel pourra légèrement dépasser les valeurs données à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin-cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

rtclock

rtclock — Read the real time clock from the operating system.

Description

Read the real-time clock from the operating system.

Syntax

```
ires rtclock
```

```
kres rtclock
```

Performance

Read the real-time clock from operating system. Under Windows, this changes only once per second. Under GNU/Linux, it ticks every microsecond. Performance under other systems varies.

Examples

Here is an example of the rtclock opcode. It uses the file *rtclock.csd* [examples/rtclock.csd].

Exemple 449. Example of the rtclock opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rtclock.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1
instr 1
  ; Get the system time.
  k1 rtclock
  ; Print it once per second.
  printk 1, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
 i   1 time     0.00002: 1018236096.00000
 i   1 time     1.00002: 1018236224.00000
```

Credits

Author: John ffitch

Example written by Kevin Conder.

New in version 4.10

s16b14

s16b14 — Creates a bank of 16 different 14-bit MIDI control message numbers.

Description

Creates a bank of 16 different 14-bit MIDI control message numbers.

Syntax

```
i1,...,i16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16
```

```
k1,...,k16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16
```

Initialization

i1 ... i64 -- output values

ichan -- MIDI channel (1-16)

ictlno_msb1 .... ictlno_msb32 -- MIDI control number, most significant byte (0-127)

ictlno_lsb1 .... ictlno_lsb32 -- MIDI control number, least significant byte (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

s16b14 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

s16b14 allows a bank of 16 different MIDI control message numbers. It uses 14-bit values instead of MIDI's normal 7-bit values.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *s16b14*, there is not an initial value input argument. The output is taken directly from the current status of internal controller array of Csound.

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

s32b14

s32b14 — Creates a bank of 32 different 14-bit MIDI control message numbers.

Description

Creates a bank of 32 different 14-bit MIDI control message numbers.

Syntax

```
i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32
```

```
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32
```

Initialization

i1 ... i64 -- output values

ichan -- MIDI channel (1-16)

ictlno_msb1 .... ictlno_msb32 -- MIDI control number, most significant byte (0-127)

ictlno_lsb1 .... ictlno_lsb32 -- MIDI control number, least significant byte (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

s32b14 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

s32b14 allows a bank of 32 different MIDI control message numbers. It uses 14-bit values instead of MIDI's normal 7-bit values.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *s32b14*, there is not an initial value input argument. The output is taken directly from the current status of internal controller array of Csound.

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

scale

scale — Signal de pondération arbitraire.

Description

Met les valeurs entrantes à l'échelle d'un intervalle défini par l'utilisateur. Semblable à l'objet de pondération que l'on trouve dans les languages de flux de données les plus connus.

Syntaxe

```
kscl scale kinput, kmax, kmin
```

Exécution

kin -- Valeur d'entrée. Elle peut provenir de n'importe quelle source au taux-k pourvu que la sortie de cette dernière soit comprise entre 0 et 1.

kmin -- Valeur minimale de l'intervalle de pondération.

kmax -- Valeur maximale de l'intervalle de pondération.

Exemples

Voici un exemple de l'opcode scale. Il utilise le fichier *scale.csd* [examples/scale.csd].

Exemple 450. Exemple de l'opcode scale.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent
-odac           -iadc     -d    ;;;realtime output
</CsOptions>
<CsInstruments>

sr = 22050
ksmps = 10
nchnls = 2

/*--- ---*/

        instr 1 ; scale test

kmod ctrl7 1, 1, 0, 1
        printk2 kmod

kout scale kmod, 0, -127
        printk2 kout

        endin

/*--- ---*/
</CsInstruments>
<CsScore>
```

```
i1 0 8888
e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

gainslider, *logcurve*, *expcurve*

Crédits

Auteur : David Akbari
Octobre
2006

samphold

samphold — Performs a sample-and-hold operation on its input.

Description

Performs a sample-and-hold operation on its input.

Syntax

```
ares samphold asig, agate [, ival] [, ivstor]
```

```
kres samphold ksig, kgate [, ival] [, ivstor]
```

Initialization

ival, ivstor (optional) -- controls initial disposition of internal save space. If *ivstor* is zero the internal « hold » value is set to *ival* ; else it retains its previous value. Defaults are 0,0 (i.e. init to zero)

Performance

kgate, xgate -- controls whether to hold the signal.

samphold performs a sample-and-hold operation on its input according to the value of *gate*. If *gate !- 0*, the input samples are passed to the output; If *gate = 0*, the last output value is repeated. The controlling *gate* can be a constant, a control signal, or an audio signal.

Examples

```
asrc  buzz      10000,440,20, 1     ; band-limited pulse train
adif  diff      asrc                ; emphasize the highs
anew  balance   adif, asrc          ;   but retain the power
agate reson     asrc,0,440          ; use a lowpass of the original
asamp samphold  anew, agate         ;   to gate the new audiosig
aout  tone      asamp,100           ; smooth out the rough edges
```

See Also

diff, *downsamp*, *integ*, *interp*, *upsamp*

sandpaper

sandpaper — Modèle semi-physique d'un son de papier de verre.

Description

sandpaper est un modèle semi-physique d'un son de papier de verre. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]
```

Initialisation

iamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,998 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

Exemples

Voici un exemple de l'opcode sandpaper. Il utilise le fichier *sandpaper.csd* [examples/sandpaper.csd].

Exemple 451. Exemple de l'opcode sandpaper.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o sandpaper.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra ---------------

  sr =           44100
  kr =            4410
  ksmps =              10
  nchnls =               1

instr 01                  ;an example of sandpaper blocks
  a1      line 2, p3, 2                             ;preset amplitude increase
  a2      sandpaper p4, 0.01            ;sandpaper needs a little amp help at these settings
  a3      product a1, a2                               ;increase amplitude
          out a3
          endin


</CsInstruments>
<CsScore>

;score -------------------

   i1 0 1 26000
   e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cabasa, *crunch*, *sekere*, *stix*

Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

scanhammer

scanhammer — Copies from one table to another with a gain control.

Description

This is is a variant of *tablecopy*, copying from one table to another, starting at *ipos*, and with a gain control. The number of points copied is determined by the length of the source. Other points are not changed. This opcode can be used to « hit » a string in the scanned synthesis code.

Syntax

```
scanhammer isrc, idst, ipos, imult
```

Initialization

isrc -- source function table.

idst -- destination function table.

ipos -- starting position (in points).

imult -- gain multiplier. A value of 0 will leave values unchanged.

See Also

scantable

Credits

Author: Matt Gilliard
April 2002

New in version 4.20

scans

scans — Generate audio output using scanned synthesis.

Description

Generate audio output using scanned synthesis.

Syntax

```
ares scans kamp, kfreq, ifn, id [, iorder]
```

Initialization

ifn -- ftable containing the scanning trajectory. This is a series of numbers that contains addresses of masses. The order of these addresses is used as the scan path. It should not contain values greater than the number of masses, or negative numbers. See the *introduction to the scanned synthesis section*.

id -- ID number of the *scanu* opcode's waveform to use

iorder (optional, default=0) -- order of interpolation used internally. It can take any value in the range 1 to 4, and defaults to 4, which is quartic interpolation. The setting of 2 is quadratic and 1 is linear. The higher numbers are slower, but not necessarily better.

Performance

kamp -- output amplitude. Note that the resulting amplitude is also dependent on instantaneous value in the wavetable. This number is effectively the scaling factor of the wavetable.

kfreq -- frequency of the scan rate

Examples

Here is an example of the scanned synthesis. It uses the file *scans.csd* [examples/scans.csd], and *string-128.matrix* [examples/string-128.matrix].

Exemple 452. Example of the scans opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o scans.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

    sr =   44100
    ksmps =   128
```

```
    nchnls =   1

    instr 1
a0  = 0
;   scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, kstif, kcentr, kdamp, ileft,
    scanu 1,     .01,    6,        2,        3,      4,         5,        2,      .1,     .1,     -.01,  .1,
;ar scans kamp,       kfreq,       ifntraj, id
a1  scans ampdb(p4), cpspch(p5), 7,       2
    out a1
    endin


</CsInstruments>
<CsScore>

; Initial condition
f1 0 128 7 0 64 1 64 0

; Masses
f2 0 128 -7 1 128 1

; Spring matrices
f3 0 16384 -23 "string-128.matrix"

; Centering force
f4  0 128 -7 0 128 2

; Damping
f5 0 128 -7 1 128 1

; Initial velocity
f6 0 128 -7 0 128 0

; Trajectories
f7 0 128 -5 .001 128 128

; Note list
i1 0  10  86 6.00
i1 11 14  86 7.00
i1 15 20  86 5.00
e


</CsScore>
</CsoundSynthesizer>
```

The matrix file « string-128.matrix », as well as several other matrices, is also available in a *zipped file* [http://www.csounds.com/scanned/zip/scanmatrices.zip] from the *Scanned Synthesis page* [http://www.csounds.com/scanned/] at cSounds.com.

Credits

Author: Paris Smaragdis
MIT Media Lab
Boston, Massachussetts USA

New in Csound version 4.05

scantable

scantable — A simpler scanned synthesis implementation.

Description

A simpler scanned synthesis implementation. This is an implementation of a circular string scanned using external tables. This opcode will allow direct modification and reading of values with the table opcodes.

Syntax

```
aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel
```

Initialization

ipos -- table containing position array.

imass -- table containing the mass of the string.

istiff -- table containing the stiffness of the string.

idamp -- table containing the damping factors of the string.

ivel -- table containing the velocities.

Performance

kamp -- amplitude (gain) of the string.

kpch -- the string's scanned frequency.

Examples

Here is an example of the scantable opcode. It uses the file *scantable.csd* [examples/scantable.csd].

Exemple 453. Example of the scantable opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o scantable.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```

```
ksmps = 10
nchnls = 1

; Table #1 - initial position
git1 ftgen 1, 0, 128, 7, 0, 64, 1, 64, 0
; Table #2 - masses
git2 ftgen 2, 0, 128, -7, 1, 128, 1
; Table #3 - stiffness
git3 ftgen 3, 0, 128, -7, 0, 64, 100, 64, 0
; Table #4 - damping
git4 ftgen 4, 0, 128, -7, 1, 128, 1
; Table #5 - initial velocity
git5 ftgen 5, 0, 128, -7, 0, 128, 0

; Instrument #1.
instr 1
  kamp init 20000
  kpch init 220
  ipos = 1
  imass = 2
  istiff = 3
  idamp = 4
  ivel = 5

  a1 scantable kamp, kpch, ipos, imass, istiff, idamp, ivel
  a2 dcblock a1

  out a2
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See Also

scanhammer

Credits

Author: Matt Gilliard
April 2002

Example written by Kevin Conder.

New in version 4.20

scanu

scanu — Compute the waveform and the wavetable for use in scanned synthesis.

Description

Compute the waveform and the wavetable for use in scanned synthesis.

Syntax

```
scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, iright, kpos, kstrngth, ain, idisp, id
```

Initialization

init -- the initial position of the masses. If this is a negative number, then the absolute of *init* signifies the table to use as a hammer shape. If *init* > 0, the length of it should be the same as the intended mass number, otherwise it can be anything.

ifnvel -- the ftable that contains the initial velocity for each mass. It should have the same size as the intended mass number.

ifnmass -- ftable that contains the mass of each mass. It should have the same size as the intended mass number.

ifnstif -- ftable that contains the spring stiffness of each connection. It should have the same size as the square of the intended mass number. The data ordering is a row after row dump of the connection matrix of the system.

ifncentr -- ftable that contains the centering force of each mass. It should have the same size as the intended mass number.

ifndamp -- the ftable that contains the damping factor of each mass. It should have the same size as the intended mass number.

ileft -- If *init* < 0, the position of the left hammer (*ileft* = 0 is hit at leftmost, *ileft* = 1 is hit at rightmost).

iright -- If *init* < 0, the position of the right hammer (*iright* = 0 is hit at leftmost, *iright* = 1 is hit at rightmost).

idisp -- If 0, no display of the masses is provided.

id -- If positive, the ID of the opcode. This will be used to point the scanning opcode to the proper waveform maker. If this value is negative, the absolute of this value is the wavetable on which to write the waveshape. That wavetable can be used later from an other opcode to generate sound. The initial contents of this table will be destroyed.

Performance

kmass -- scales the masses

kstif -- scales the spring stiffness

kcentr -- scales the centering force

kdamp -- scales the damping

kpos -- position of an active hammer along the string (*kpos* = 0 is leftmost, *kpos* = 1 is rightmost). The shape of the hammer is determined by *init* and the power it pushes with is *kstrngth*.

kstrngth -- power that the active hammer uses

ain -- audio input that adds to the velocity of the masses. Amplitude should not be too great.

Examples

For an example, see the documentation on *scans*.

Credits

Author: Paris Smaragdis
MIT Media Lab
Boston, Massachussetts USA
March 2000

New in Csound version 4.05

scoreline

scoreline — Issues one or more score line events from an instrument.

Description

Scoreline will issue one or more score events, if ktrig is 1 every k-period. It can handle strings in the same conditions as the standard score. Multi-line strings are accepted, using {{ }} to enclose the string.

Syntax

```
scoreline Sin, ktring
```

Initialization

« Sin » -- a string (in double-quotes or enclosed by {{ }}) containing one or more score events.

Performance

« ktrig » -- event trigger, 1 issues the score event, 0 bypasses it.

Examples

Here is an example of the scoreline opcode.

Exemple 454. Example

```
instr 1
         ktrig init 1
 scoreline {{
               i 2  0  3  "flutec3.wav"
               i 2  1  3  "clarc3.wav"
             }}, ktrig
         ktrig = 0
endin

instr 2
    aout soundin p4
    out aout
endin
```

You can use string opcodes like *sprintfk* to produce strings to be passed to *scoreline* like this:

```
Sfil = "/Volumes/Bla/file.aif"
String  sprintfk {{i 2 0 %f "%s" %f %f %f %f}}, idur, Sfil, p5, p6, knorm, iskip
scoreline String, ktrig
```

Credits

Author: Victor Lazzarini, 2007

scoreline_i

scoreline_i — Issues one or more score line events from an instrument at i-time.

Description

scoreline_i will issue score events at i-time. It can handle strings in the same conditions as the standard score. Multi-line strings are accepted, using {{ }} to enclose the string.

Syntax

```
scoreline_i Sin
```

Initialization

« Sin » -- a string (in double-quotes or enclosed by {{ }}) containing one or more score events.

Examples

Here is an example of the scoreline_i opcode.

Exemple 455. Example

```
instr 1

 scoreline_i {{
               i 2  0  3  "flutec3.wav"
               i 2  1  3  "clarc3.wav"
             }}

endin

instr 2
    aout soundin p4
    out aout
endin
```

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

Credits

Author: Victor Lazzarini, 2007

schedkwhen

schedkwhen — Adds a new score event generated by a k-rate trigger.

Description

Adds a new score event generated by a k-rate trigger.

Syntax

```
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \
      [, ip4] [, ip5] [...]
```

```
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \
      [, ip4] [, ip5] [...]
```

Initialization

« insname » -- A string (in double-quotes) representing a named instrument.

ip4, ip5, ... -- Equivalent to p4, p5, etc., in a score *i statement*

Performance

ktrigger -- triggers a new score event. If *ktrigger* = 0, no new event is triggered.

kmintim -- minimum time between generated events, in seconds. If *kmintim* <= 0, no time limit exists. If the *kinsnum* is negative (to turn off an instrument), this test is bypassed.

kmaxnum -- maximum number of simultaneous instances of instrument *kinsnum* allowed. If the number of extant instances of *kinsnum* is >= *kmaxnum*, no new event is generated. If *kmaxnum* is <= 0, it is not used to limit event generation. If the *kinsnum* is negative (to turn off an instrument), this test is bypassed.

kinsnum -- instrument number. Equivalent to p1 in a score *i statement*.

kwhen -- start time of the new event. Equivalent to p2 in a score *i statement*. Measured from the time of the triggering event. *kwhen* must be >= 0. If *kwhen* > 0, the instrument will not be initialized until the actual time when it should start performing.

kdur -- duration of event. Equivalent to p3 in a score *i statement*. If *kdur* = 0, the instrument will only do an initialization pass, with no performance. If *kdur* is negative, a held note is initiated. (See *ihold* and *i statement*.)

Note: While waiting for events to be triggered by *schedkwhen*, the performance must be kept going, or Csound may quit if no score events are expected. To guarantee continued performance, an *f0 statement* may be used in the score.

Examples

Here is an example of the schedkwhen opcode. It uses the file *schedkwhen.csd* [examples/schedkwhen.csd].

Exemple 456. Example of the schedkwhen opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedkwhen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
  ; Use the fourth p-field as the trigger.
  ktrigger = p4
  kmintim = 0
  kmaxnum = 2
  kinsnum = 2
  kwhen = 0
  kdur = 0.5

  ; Play Instrument #2 at the same time, if the trigger is set.
  schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur

  ; Play a high note.
  a1 oscils 10000, 880, 1
  out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
  ; Play a low note.
  a1 oscils 10000, 220, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = trigger for Instrument #2 (when p4 > 0).
; Play Instrument #1 for half a second, no trigger.
i 1 0 0.5 0
; Play Instrument #1 for half a second, trigger Instrument #2.
i 1 1 0.5 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Rasmus Ekman
EMS, Stockholm, Sweden

Example written by Kevin Conder.

New in Csound version 3.59

schedkwhennamed

schedkwhennamed — Similar to schedkwhen but uses a named instrument at init-time.

Description

Similar to *schedkwhen* but uses a named instrument at init-time.

Syntax

```
schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
      [, ip4] [, ip5] [...]
```

Initialization

ip4, ip5, ... -- Equivalent to p4, p5, etc., in a score *i statement*

Performance

ktrigger -- triggers a new score event. If *ktrigger* is 0, no new event is triggered.

kmintim -- minimum time between generated events, in seconds. If *kmintim* is less than or equal to 0, no time limit exists.

kmaxnum -- maximum number of simultaneous instances of named instrument allowed. If the number of extant instances of the named instrument is greater than or equal to *kmaxnum*, no new event is generated. If *kmaxnum* is less than or equal to 0, it is not used to limit event generation.

"name" -- the named instrument's name.

kwhen -- start time of the new event. Equivalent to p2 in a score *i statement*. Measured from the time of the triggering event. *kwhen* must be greater than or equal to 0. If *kwhen* greater than 0, the instrument will not be initialized until the actual time when it should start performing.

kdur -- duration of event. Equivalent to p3 in a score *i statement*. If *kdur* is 0, the instrument will only do an initialization pass, with no performance. If *kdur* is negative, a held note is initiated. (See *ihold* and *i statement*.)

Note: While waiting for events to be triggered by *schedkwhennamed*, the performance must be kept going, or Csound may quit if no score events are expected. To guarantee continued performance, an *f0 statement* may be used in the score.

Examples

Here is an example of the schedkwhennamed opcode. It uses the file *schedkwhennamed.csd* [examples/schedkwhennamed.csd].

Exemple 457. Example of the schedkwhennamed opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d
; For Non-realtime ouput leave only the line below:
; -o schedkwhennamed.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

  sr      =  48000
  ksmps      =  16
  nchnls     =  2
  0dbfs      =  1

; Example by Jonathan Murphy 2007

  gSinstr2   =   "printer"

    instr 1

  ktrig     metro     1
if (ktrig == 1) then
  ;Call instrument "printer" once per second
            schedkwhennamed   ktrig, 0, 1, gSinstr2, 0, 1

endif

    endin

    instr printer

  ktime     timeinsts
            printk2   ktime

    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

See Also

schedkwhen

Credits

Author: Rasmus Ekman
EMS, Stockholm, Sweden

New in Csound version 4.23

schedule

schedule — Adds a new score event.

Description

Adds a new score event.

Syntax

```
schedule insnum, iwhen, idur [, ip4] [, ip5] [...]

schedule "insname", iwhen, idur [, ip4] [, ip5] [...]
```

Initialization

insnum -- instrument number. Equivalent to p1 in a score *i statement*. *insnum* must be a number greater than the number of the calling instrument.

« insname » -- A string (in double-quotes) representing a named instrument.

iwhen -- start time of the new event. Equivalent to p2 in a score *i statement*. *iwhen* must be nonnegative. If *iwhen* is zero, *insum* must be greater than or equal to the p1 of the current instrument.

idur -- duration of event. Equivalent to p3 in a score *i statement*.

ip4, ip5, ... -- Equivalent to p4, p5, etc., in a score *i statement*.

Performance

ktrigger -- trigger value for new event

schedule adds a new score event. The arguments, including options, are the same as in a score. The *iwhen* time (p2) is measured from the time of this event.

If the duration is zero or negative the new event is of MIDI type, and inherits the release sub-event from the scheduling instruction.

Examples

Here is an example of the schedule opcode. It uses the file *schedule.csd* [examples/schedule.csd].

Exemple 458. Example of the schedule opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
```

```
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedule.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
  ; Play Instrument #2 at the same time.
  schedule 2, 0, p3

  ; Play a high note.
  a1 oscils 10000, 880, 1
  out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
  ; Play a low note.
  a1 oscils 10000, 220, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for half a second.
i 1 0 0.5
; Play Instrument #1 for half a second.
i 1 1 0.5
e


</CsScore>
</CsoundSynthesizer>
```

See Also

schedwhen

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
November 1998

Example written by Kevin Conder.

New in Csound version 3.491

Based on work by Gabriel Maldonado

Thanks goes to David Gladstein, for clarifying the *iwhen* parameter.

schedwhen

schedwhen — Adds a new score event.

Description

Adds a new score event.

Syntax

```
schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
```

```
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]
```

Initialization

ip4, ip5, ... -- Equivalent to p4, p5, etc., in a score *i statement*.

Performance

kinsnum -- instrument number. Equivalent to p1 in a score *i statement*.

« insname » -- A string (in double-quotes) representing a named instrument.

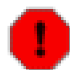
ktrigger -- trigger value for new event

kwhen -- start time of the new event. Equivalent to p2 in a score *i statement*.

kdur -- duration of event. Equivalent to p3 in a score *i statement*.

schedwhen adds a new score event. The event is only scheduled when the k-rate value *ktrigger* is first non-zero. The arguments, including options, are the same as in a score. The *iwhen* time (p2) is measured from the time of this event.

If the duration is zero or negative the new event is of MIDI type, and inherits the release sub-event from the scheduling instruction.



Warning

Support for named instruments is broken in version 4.23

Examples

Here is an example of the schedwhen opcode. It uses the file *schedwhen.csd* [examples/schedwhen.csd].

Exemple 459. Example of the schedwhen opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedwhen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
  ; Use the fourth p-field as the trigger.
  ktrigger = p4
  kinsnum = 2
  kwhen = 0
  kdur = p3

  ; Play Instrument #2 at the same time, if the trigger is set.
  schedwhen ktrigger, kinsnum, kwhen, kdur

  ; Play a high note.
  a1 oscils 10000, 880, 1
  out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
  ; Play a low note.
  a1 oscils 10000, 220, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = trigger for Instrument #2 (when p4 > 0).
; Play Instrument #1 for half a second, trigger Instrument #2.
i 1 0 0.5 1
; Play Instrument #1 for half a second, no trigger.
i 1 1 0.5 0
e


</CsScore>
</CsoundSynthesizer>
```

See Also

schedule

Credits

Author: John ffitch
University of Bath/Codemist Ltd.
Bath, UK
November 1998

Example written by Kevin Conder.

New in Csound version 3.491

Based on work by Gabriel Maldonado

seed

seed — Fixe la valeur globale de la graine.

Description

Fixe la valeur globale de la graine pout tous les *générateurs de bruit de classe x*, ainsi que pour d'autres opcodes qui utilisent un appel de random, tels que *grain*.

Noter que

rand, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

Syntaxe

```
seed ival
```

Exécution

Avec l'utilisation de *seed* on obtiendra des résultats prévisibles d'un orchestre utilisant des générateurs de nombres aléatoires, lors de plusieurs exécutions.

Lors de la spécification d'une valeur de graine, *ival* doit être un entier compris entre 0 et 2^{32}. Si *ival* = 0, la valeur de *ival* sera dérivée de l'horloge du système.

sekere

sekere — Modèle semi-physique d'un son de chekeré.

Description

sekere est un modèle semi-physique d'un son de chekeré. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]
```

Initialisation

iamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 64.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,998 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

Exemples

Voici un exemple de l'opcode sekere. Il utilise le fichier *sekere.csd* [examples/sekere.csd].

Exemple 460. Exemple de l'opcode sekere.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o sekere.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra ---------------

  sr =            44100
  kr =             4410
  ksmps =               10
  nchnls =                1

instr 01                  ;an example of a sekere
a1       sekere p4, 0.01
           out a1
           endin


</CsInstruments>
<CsScore>

;score -------------------

   i1 0 1 26000
   e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cabasa, *crunch*, *sandpaper*, *stix*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

semitone

semitone — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

Syntaxe

```
semitone(x)
```

Cette fonction travaille aux taux-i, -k et -a.

Initialisation

x -- une valeur exprimée en demi-tons.

Exécution

La valeur retournée par la fonction *semitone* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de demi-tons spécifié.

Exemples

Voici un exemple de l'opcode demi-ton. Il utilise le fichier *semitone.csd* [examples/semitone.csd].

Exemple 461. Exemple de l'opcode demi-ton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o semitone.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; The root note is A above middle-C (440 Hz)
  iroot = 440

  ; Raise the root note by three semitones to C.
  isemitone = 3

  ; Calculate the new note.
```

```
  ifactor = semitone(isemitone)
  inew = iroot * ifactor

  ; Print out all of the values.
  print iroot
  print ifactor
  print inew
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1:  iroot = 440.000
instr 1:  ifactor = 1.189
instr 1:  inew = 523.229
```

Voir Aussi

cent, *db*, *octave*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

sense

sense — Same as the sensekey opcode.

Description

Same as the *sensekey* opcode.

sensekey

sensekey — Returns the ASCII code of a key that has been pressed.

Description

Returns the ASCII code of a key that has been pressed, or -1 if no key has been pressed.

Syntax

```
kres[, kkeydown] sensekey
```

Performance

kres - returns the ASCII value of a key which is pressed or released.

kkeydown - returns 1 if the key was pressed, 0 if it was released or if there is no key event.

kres can be used to read keyboard events from stdin and returns the ASCII value of any key that is pressed or released, or it returns -1 when there is no keyboard activity. The value of *kkeydown* is 1 when a key was pressed, or 0 otherwise. This behavior is emulated by default, so a key release is generated immediately after every key press. To have full functionality, FLTK can be used to capture keyboard events. *FLpanel* can be used to capture keyboard events and send them to the sensekey opcode, by adding an additional optional argument. See *FLpanel* for more information.

Note

This opcode can also be written as *sense*.

Examples

Here is an example of the sensekey opcode. It uses the file *sensekey.csd* [examples/sensekey.csd].

Exemple 462. Example of the sensekey opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sensekey.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  k1 sensekey
  printk2 k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 30
e


</CsScore>
</CsoundSynthesizer>
```

Here is what the output should look like when the "q" button is pressed...

```
q i1   113.00000
```

Here is an example of the sensekey opcode in conjucntion with *FLpanel*. It uses the file *FLpanel-sensekey.csd* [examples/FLpanel-sensekey.csd].

Exemple 463. Example of the sensekey opcode using keyboard capture from an FLpanel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel-sensekey.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; Example by Johnathan Murphy

  sr      =  44100
  ksmps      =  128
  nchnls     =  2


        ; ikbdcapture flag set to 1
  ikey     init      1

            FLpanel    "sensekey", 740, 340, 100, 250, 2, ikey
  gkasc, giasc FLbutBank 2, 16, 8, 700, 300, 20, 20, -1
            FLpanelEnd
            FLrun

    instr 1

  kkey     sensekey
  kprint    changed   kkey
            FLsetVal  kprint, kkey, giasc

    endin

</CsInstruments>
<CsScore>
i1 0 60
e
</CsScore>
</CsoundSynthesizer>
```

The lit button in the FLpanel window shows the last key pressed.

Here is a more complex example of the sensekey opcode in conjucntion with *FLpanel*. It uses the file *FLpanel-sensekey2.csd* [examples/FLpanel-sensekey.csd].

Exemple 464. Example of the sensekey opcode using keyboard capture from an FLpanel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac          ; -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel-sensekey2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr =  48000
ksmps =  32
nchnls =  1
; Example by Istvan Varga
; if the FLTK opcodes are commented out, sensekey will read keyboard
; events from stdin
        FLpanel "", 150, 50, 100, 100, 0, 1
        FLlabel 18, 10, 1, 0, 0, 0
        FLgroup "Keyboard Input", 150, 50, 0, 0, 0
        FLgroupEnd
        FLpanelEnd

        FLrun

        instr 1

ktrig1 init 1
ktrig2 init 1
nxtKey1:
k1, k2 sensekey
        if (k1 != -1 || k2 != 0) then
        printf "Key code = %02X, state = %d\n", ktrig1, k1, k2
ktrig1 =  3 - ktrig1
        kgoto nxtKey1
        endif
nxtKey2:
k3 sensekey
        if (k3 != -1) then
        printf "Character = '%c'\n", ktrig2, k3
ktrig2 =  3 - ktrig2
        kgoto nxtKey2
        endif

        endin

</CsInstruments>
<CsScore>
i 1 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

The console output will look something like:

new alloc for instr 1:
Key code = 65, state = 1
Character = 'e'
Key code = 65, state = 0
Key code = 72, state = 1
Character = 'r'
Key code = 72, state = 0
Key code = 61, state = 1
Character = 'a'

Key code = 61, state = 0

See also

FLpanel, *FLkeyIn*

Credits

Author: John ffitch
University of Bath, Codemist. Ltd.
Bath, UK
October 2000

Examples written by Kevin Conder, Johnathan Murphy and Istvan Varga.

New in Csound version 4.09. Renamed in Csound version 4.10.

seqtime

seqtime — Generates a trigger signal according to the values stored in a table.

Description

Generates a trigger signal according to the values stored in a table.

Syntax

```
ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times
```

Performance

ktrig_out -- output trigger signal

ktime_unit -- unit of measure of time, related to seconds.

kstart -- start index of looped section

kloop -- end index of looped section

kinitndx -- initial index

Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

kfn_times -- number of table containing a sequence of times

This opcode handles timed-sequences of groups of values stored into a table.

seqtime generates a trigger signal (a sequence of impulses, see also *trigger* opcode), according to the values stored in the *kfn_times* table. This table should contain a series of delta-times (i.e. times beetween to adjacent events). The time units stored into table are expressed in seconds, but can be rescaled by means of ktime_unit argument. The table can be filled with *GEN02* or by means of an external text-file containing numbers, with *GEN23*.

Note

Note that the *kloop* index marks the loop boundary and is NOT included in the looped elements. If you want to loop the first four elements, you would set *kstart* to 0 and *kloop* to 4.

It is possible to start the sequence from a value different than the first, by assigning to *kinitndx* an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *kinitndx*) correspond to valid table numbers, otherwise Csound will crash (because no range-checking is implementeted).

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* argu-

ments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value. It is possible to trigger two events almost at the same time (actually separated by a k-cycle) by giving a zero value to the corresponding delta-time. First element contained in the table should be zero, if the user intends to send a trigger impulse, it should come immediately after the orchestra instrument containing *seqtime* opcode.

Examples

Here is an example of the seqtime opcode. It uses the file *seqtime.csd* [examples/seqtime.csd].

Exemple 465. Example of the seqtime opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o seqtime.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

 sr = 44100
 ksmps = 64
 nchnls = 1

; By Tim Mortimer and Andres Cabrera 2007

0dbfs = 1

gisine         ftgen    0, 0, 8192, 10,    1
;;; table defining an integer pitch set
gipset    ftgen     0, 0, 4, -2, 8.00, 8.04, 8.07, 8.10
;;;DELTA times for seqtime
gidelta    ftgen    0, 0, 4, -2, .5, 1, .25, 1.25


  instr 1

kndx init 0
ktrigger init 0

ktime_unit init 1
kstart init p4
kloop init p5
kinitndx init 0
kfn_times init gidelta

ktrigger seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

printk2 ktrigger


if (ktrigger > 0) then
   kpitch table kndx, gipset
   event "i", 2, 0, 1, kpitch
   kndx = kndx + 1
   kndx = kndx % kloop
endif

  endin


  instr 2
icps = cpspch (p4)
a1    buzz    1, icps, 7, gisine
aamp expseg    0.00003,.02,1,p3-.02,0.00003

a1 = a1 * aamp * 0.5
```

```
out a1
  endin

</CsInstruments>
<CsScore>
;       start    dur   kstart  kloop
i 1  0  7  0  4
i 1  8  10  0  3
i 1  19  10  4  4

</CsScore>
</CsoundSynthesizer>
```

See Also

GEN02, *GEN23*, *trigseq* *seqtime2*

Credits

Author: Gabriel Maldonado

November 2002. Added a note about the *kinitndx* parameter, thanks to Rasmus Ekman.

New in version 4.06

Example by: Tim Mortimer and Andres Cabrera 2007

seqtime2

seqtime2 — Generates a trigger signal according to the values stored in a table.

Description

Generates a trigger signal according to the values stored in a table.

Syntax

```
ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times
```

Performance

ktrig_out -- output trigger signal

ktime_unit -- unit of measure of time, related to seconds.

ktime_in -- input trigger signal.

kstart -- start index of looped section

kloop -- end index of looped section

kinitndx -- initial index

Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

kfn_times -- number of table containing a sequence of times

This opcode handles timed-sequences of groups of values stored into a table.

seqtime2 generates a trigger signal (a sequence of impulses, see also *trigger* opcode), according to the values stored in the *kfn_times* table. This table should contain a series of delta-times (i.e. times beetween to adjacent events). The time units stored into table are expressed in seconds, but can be rescaled by means of ktime_unit argument. The table can be filled with *GEN02* or by means of an external text-file containing numbers, with *GEN23*.

It is possible to start the sequence from a value different than the first, by assigning to *initndx* an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *initndx*) correspond to valid table numbers, otherwise Csound will crash (because no range-checking is implementeted).

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* arguments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value. It is possible to trigger two events almost at the same time (actually separated by a k-cycle) by giving a zero value to the corresponding delta-time. First element contained in the table should be zero, if the user intends to send a trigger impulse, it should come immediately after the orchestra instrument containing *seqtime2* opcode.

seqtime2 is similar to *seqtime*, the difference is that when ktrig_in contains a non-zero value, current index is reset to kinitndx value. kinitndx can be varied at performance time.

See Also

GEN02, *GEN23*, *seqtime*, *trigseq*, *timedseq*

Credits

Author: Gabriel Maldonado

setctrl

setctrl — Configurable slider controls for realtime user input.

Description

Configurable slider controls for realtime user input. Requires Winsound or TCL/TK. *setctrl* sets a slider to a specific value, or sets a minimum or maximum range.

Syntax

```
setctrl inum, ival, itype
```

Initialization

inum -- number of the slider to set

ival -- value to be sent to the slider

itype -- type of value sent to the slider as follows:

- 1 -- set the current value. Initial value is 0.
- 2 -- set the minimum value. Default is 0.
- 3 -- set the maximum value. Default is 127.
- 4 -- set the label. (New in Csound version 4.09)

Performance

Calling *setctrl* will create a new slider on the screen. There is no theoretical limit to the number of sliders. Windows and TCL/TK use only integers for slider values, so the values may need rescaling. GUIs usually pass values at a fairly slow rate, so it may be advisable to pass the output of control through *port*.

Examples

Here is an example of the setctrl opcode. It uses the file *setctrl.csd* [examples/setctrl.csd].

Exemple 466. Example of the setctrl opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o setctrl.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Display the label "Volume" on Slider #1.
  setctrl 1, "Volume", 4
  ; Set Slider #1's initial value to 20.
  setctrl 1, 20, 1

  ; Capture and display the values for Slider #1.
  k1 control 1
  printk2 k1

  ; Play a simple oscillator.
  ; Use the values from Slider #1 for amplitude.
  kamp = k1 * 128
  a1 oscil kamp, 440, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for thirty seconds.
i 1 0 30
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
 i1    38.00000
 i1    40.00000
 i1    43.00000
```

See Also

control

Credits

Author: John ffitch
University of Bath, Codemist. Ltd.
Bath, UK
May 2000

Example written by Kevin Conder.

New in Csound version 4.06

setksmps

setksmps — Sets the local ksmps value in a user-defined opcode block.

Description

Sets the local ksmps value in a user-defined opcode block.

The *setksmps* statement can be used to set the local *ksmps* value of the user-defined opcode block. It has one i-time parameter specifying the new *ksmps* value (which is left unchanged if zero is used). *setksmps* should be used before any other opcodes (but allowed after *xin*), otherwise unpredictable results may occur.

Syntax

```
setksmps iksmps
```

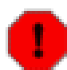
Initialization

iksmps -- sets the local ksmps value.

If *iksmps* is set to zero, the *ksmps* of the caller instrument or opcode is used (this is the default behavior).

Note

The local *ksmps* is implemented by splitting up a control period into smaller sub-kperiods and temporarily modifying internal Csound global variables. This also requires converting the rate of k-rate input and output arguments (input variables receive the same value in all sub-kperiods, while outputs are written only in the last one). It also means that you cannot use a local *ksmps* that is higher than the global *ksmps*.



Warning about local ksmps

When the local *ksmps* is not the same as the orchestra level *ksmps* value (as specified in the orchestra header). Global a-rate operations must not be used in the user-defined opcode block.

These include:

- any access to « ga » variables
- a-rate zak opcodes (*zar*, *zaw*, etc.)
- *tablera* and *tablewa* (these two opcodes may in fact work, but caution is needed)
- The *in* and *out* opcode family (these read from, and write to global a-rate buffers)

In general, the local *ksmps* should be used with care as it is an experimental feature. Though it works correctly in most cases.

The *setksmps* statement can be used to set the local *ksmps* value of the user-defined opcode block. It has one i-time parameter specifying the new *ksmps* value (which is left unchanged if zero is used). *setksmps* should be used before any other opcodes (but allowed after *xin*), otherwise unpredictable results may occur.

Performance

The syntax of a user-defined opcode block is as follows:

```
opcode  name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
[setksmps  iksmps]
... the rest of the instrument's code.
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

The new opcode can then be used with the usual syntax:

```
[xinarg1] [, xinarg2] ... [xinargN]  name  [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

Examples

See the example for the *opcode* opcode.

See Also

endop, *opcode*, *xin*, *xout*

Credits

Author: Istvan Varga, 2002; based on code by Matt J. Ingalls

New in version 4.22

setscorepos

setscorepos — Sets the playback position of the current score performance to a given position.

Description

Sets the playback position of the current score performance to a given position.

Syntax

```
setscorepos  ipos
```

Initialization

ipos -- playback position in seconds.

Examples

Here is an example of the setscorepos opcode.

Exemple 467. Example of the setscorepos opcode.

```
instr 1
    setscorepos 10
    endin
```

See Also

setscorepos,

Credits

Author: Victor Lazzarini;
2008

New in Csound version 5.09

sfilist

sfilist — Prints a list of all instruments of a previously loaded SoundFont2 (SF2) file.

Description

Prints a list of all instruments of a previously loaded SoundFont2 (SF2) sample file. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
sfilist ifilhandle
```

Initialization

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

Performance

sfilist prints a list of all instruments of a previously loaded SF2 file to the console.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfinstr, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfinstr

sfinstr — Plays a SoundFont2 (SF2) sample instrument, generating a stereo sound.

Description

Plays a SoundFont2 (SF2) sample instrument, generating a stereo sound. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

instrnum -- number of an instrument of a SF2 file.

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will probably crash.

sfinstr plays an SF2 instrument instead of a preset (an SF2 instrument is the base of a preset layer). *instr-*

num specifies the instrument number, and the user must be sure that the specified number belongs to an existing instrument of a determinate soundfont bank. Notice that both *xamp* and *xfreq* can operate at k-rate as well as a-rate, but both arguments must work at the same rate.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfinstr3

sfinstr3 — Plays a SoundFont2 (SF2) sample instrument, generating a stereo sound with cubic interpolation.

Description

Plays a SoundFont2 (SF2) sample instrument, generating a stereo sound with cubic interpolation. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

instrnum -- number of an instrument of a SF2 file.

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will probably crash.

sfinstr3 is a cubic-interpolation version of *sfinstr*. Difference of sound-quality is noticeable specially in bass-frequency-transposed samples. In high-freq-transposed samples the difference is less noticeable, and I suggest to use linear-interpolation versions, because they are faster.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr3m*, *sfinstrm*, *sfinstr*, *sfload*, *sfpassign*, *sfplay3*, *sfplay3m*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfinstr3m

sfinstr3m — Plays a SoundFont2 (SF2) sample instrument, generating a mono sound with cubic interpolation.

Description

Plays a SoundFont2 (SF2) sample instrument, generating a mono sound with cubic interpolation. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

instrnum -- number of an instrument of a SF2 file.

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will probably crash.

sfinstr3m is a cubic-interpolation version of *sfinstrm*. Difference of sound-quality is noticeable specially in bass-frequency-transposed samples. In high-freq-transposed samples the difference is less noticeable, and I suggest to use linear-interpolation versions, because they are faster.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr3*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay3*, *sfplay3m*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfinstrm

sfinstrm — Plays a SoundFont2 (SF2) sample instrument, generating a mono sound.

Description

Plays a SoundFont2 (SF2) sample instrument, generating a mono sound. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

instrnum -- number of an instrument of a SF2 file.

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will probably crash.

sfinstrm plays is a mono version of *sfinstr*. This is the fastest opcode of the SF2 family.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfload

sfload — Loads an entire SoundFont2 (SF2) sample file into memory.

Description

Loads an entire SoundFont2 (SF2) sample file into memory. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

sfload should be placed in the header section of a Csound orchestra.

Syntax

```
ir sfload "filename"
```

Initialization

ir -- output to be used by other SF2 opcodes. For *sfload*, *ir* is *ifilhandle*.

« filename » -- name of the SF2 file, with its complete path. It must be a string typed within double-quotes with « / » to separate directories (this applies to DOS and Windows as well, where using a backslash will generate an error), or an integer that has been the subject of a *strset* operation

Performance

sfload loads an entire SF2 file into memory. It returns a file handle to be used by other opcodes. Several instances of *sfload* can placed in the header section of an orchestra, allowing use of more than one SF2 file in a single orchestra.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sflooper

sflooper — Plays a SoundFont2 (SF2) sample preset, generating a stereo sound, with user-defined time-varying crossfade looping.

Description

Plays a SoundFont2 (SF2) sample preset, generating a stereo sound, similarly to sfplay. Unlike that opcode, though, it ignores the looping points set in the SF2 file and substitutes them for a user-defined crossfade loop. It is a cross between sfplay and flooper2.

Syntax

```
ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend, kcrossfade, ifn \
      [, istart, imode, ifenv, iskip]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

ipreindex -- preset index

istart -- playback start pos in seconds

imode -- loop modes: 0 forward, 1 backward, 2 back-and-forth [def: 0]

ifenv -- if non-zero, crossfade envelope shape table number. The default, 0, sets the crossfade to linear.

iskip -- if 1, the opcode initialisation is skipped, for tied notes, performance continues from the position in the loop where the previous note stopped. The default, 0, does not skip initialisation

Performance

kamp -- amplitude scaling

kpitch -- pitch control (transposition ratio); negative values are not allowed.

kloopstart -- loop start point (secs). Note that although k-rate, loop parameters such as this are only updated once per loop cycle. If loop start is set beyond the end of the sample, no looping will result.

kloopend -- loop end point (secs), updated once per loop cycle.

kcrossfade -- crossfade length (secs), updated once per loop cycle and limited to loop length.

sflooper plays a preset, generating a stereo sound.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Victor Lazzarini;
August 2007

New in Csound Version 5.07

sfpassign

sfpassign — Assigns all presets of a SoundFont2 (SF2) sample file to a sequence of progressive index numbers.

Description

Assigns all presets of a previously loaded SoundFont2 (SF2) sample file to a sequence of progressive index numbers. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

sfpassign should be placed in the header section of a Csound orchestra.

Syntax

```
sfpassign istartindex, ifilhandle[, imsgs]
```

Initialization

istartindex -- starting index preset by the user in bulk preset assignments.

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

imsgs -- if non-zero messages are suppressed.

Performance

sfpassign assigns all presets of a previously loaded SF2 file to a sequence of progressive index numbers, to be used later with the opcodes *sfplay* and *sfplaym*. *istartindex* specifies the starting index number. Any number of *sfpassign* instances can be placed in the header section of an orchestra, each one assigning presets belonging to different SF2 files. The user must take care that preset index numbers of different SF2 files do not overlap.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfplay

sfplay — Plays a SoundFont2 (SF2) sample preset, generating a stereo sound.

Description

Plays a SoundFont2 (SF2) sample preset, generating a stereo sound. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

ipreindex -- preset index

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

ienv (optional) -- enables and determines amplitude envelope amplitude envelope. 0 = no envelope, 1 = linear attack and decay, 2 = linear attack, exponential decay (see below). Default = 0.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

Notice that both *xamp* and *xfreq* can use k-rate as well as a-rate signals. Both arguments must use variables of the same rate, or *sfplay* will not work correctly. *ipreindex* must contain the number of a previously assigned preset, or Csound will crash.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will proba-

bly crash.

The *ienv* parameter enables and determines the type of amplitude envelope used. The default value is 0, or no envelope. If *ienv* is set to 1, the attack and decay portions are linear. If set to 2, the attack is linear and the decay is exponential. The release portion of the envelope has not yet been implemented.

sfplay plays a preset, generating a stereo sound. *ivel* does not directly affect the amplitude of the output, but informs *sfplay* about which sample should be chosen in multi-sample, velocity-split presets.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplaym*, *sfplaym*, *sfplaym*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

New optional parameter *ienv* in version 5.09

sfplay3

sfplay3 — Plays a SoundFont2 (SF2) sample preset, generating a stereo sound with cubic interpolation.

Description

Plays a SoundFont2 (SF2) sample preset, generating a stereo sound with cubic interpolation. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

ipreindex -- preset index

iflag -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

ienv (optional) -- enables and determines amplitude envelope amplitude envelope. 0 = no envelope, 1 = linear attack and decay, 2 = linear attack, exponential decay (see below). Default = 0.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

Notice that both *xamp* and *xfreq* can use k-rate as well as a-rate signals. Both arguments must use variables of the same rate, or *sfplay3* will not work correctly. *ipreindex* must contain the number of a previously assigned preset, or Csound will crash.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will proba-

bly crash.

The *ienv* parameter enables and determines the type of amplitude envelope used. The default value is 0, or no envelope. If *ienv* is set to 1, the attack and decay portions are linear. If set to 2, the attack is linear and the decay is exponential. The release portion of the envelope has not yet been implemented.

sfplay3 plays a preset, generating a stereo sound with cubic interpolation. *ivel* does not directly affect the amplitude of the output, but informs *sfplay3* about which sample should be chosen in multi-sample, velocity-split presets.

sfplay3 is a cubic-interpolation version of *sfplay*. Difference of sound-quality is noticeable specially in bass-frequency-transposed samples. In high-freq-transposed samples the difference is less noticeable, and I suggest to use linear-interpolation versions, because they are faster.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr3*, *sfinstr3m*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay3m*, *sfplaym*, *sfplay*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

New optional parameter *ienv* in version 5.09

sfplay3m

sfplay3m — Plays a SoundFont2 (SF2) sample preset, generating a mono sound with cubic interpolation.

Description

Plays a SoundFont2 (SF2) sample preset, generating a mono sound with cubic interpolation. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

ipreindex -- preset index

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

ienv (optional) -- enables and determines amplitude envelope amplitude envelope. 0 = no envelope, 1 = linear attack and decay, 2 = linear attack, exponential decay (see below). Default = 0.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

Notice that both *xamp* and *xfreq* can use k-rate as well as a-rate signals. Both arguments must use variables of the same rate, or *sfplay3m* will not work correctly. *ipreindex* must contain the number of a previously assigned preset, or Csound will crash.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user

should make sure that its value is within the length of the specific sound. Otherwise, Csound will probably crash.

The *ienv* parameter enables and determines the type of amplitude envelope used. The default value is 0, or no envelope. If *ienv* is set to 1, the attack and decay portions are linear. If set to 2, the attack is linear and the decay is exponential. The release portion of the envelope has not yet been implemented.

sfplay3m is a mono version of *sfplay3*. It should be used with mono preset, or with the stereo presets in which stereo output is not required. It is faster than *sfplay3*.

sfplay3m is also a cubic-interpolation version of *sfplaym*. Difference of sound-quality is noticeable specially in bass-frequency-transposed samples. In high-freq-transposed samples the difference is less noticeable, and I suggest to use linear-interpolation versions, because they are faster.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr3*, *sfinstr3m*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay3*, *sfplaym*, *sfplay*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

New optional parameter *ienv* in version 5.09

sfplaym

sfplaym — Plays a SoundFont2 (SF2) sample preset, generating a mono sound.

Description

Plays a SoundFont2 (SF2) sample preset, generating a mono sound. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

Initialization

ivel -- velocity value

inotenum -- MIDI note number value

ipreindex -- preset index

iflag (optional) -- flag regarding the behavior of *xfreq* and *inotenum*

ioffset (optional) -- start playing at offset, in samples.

ienv (optional) -- enables and determines amplitude envelope amplitude envelope. 0 = no envelope, 1 = linear attack and decay, 2 = linear attack, exponential decay (see below). Default = 0.

Performance

xamp -- amplitude correction factor

xfreq -- frequency value or frequency multiplier, depending by *iflag*. When *iflag* = 0, *xfreq* is a multiplier of a the default frequency, assigned by SF2 preset to the *inotenum* value. When *iflag* = 1, *xfreq* is the absolute frequency of the output sound, in Hz. Default is 0.

When *iflag* = 0, *inotenum* sets the frequency of the output according to the MIDI note number used, and *xfreq* is used as a multiplier. When *iflag* = 1, the frequency of the output, is set directly by *xfreq*. This allows the user to use any kind of micro-tuning based scales. However, this method is designed to work correctly only with presets tuned to the default equal temperament. Attempts to use this method with a preset already having non-standard tunings, or with drum-kit-based presets, could give unexpected results.

Adjustment of the amplitude can be done by varying the *xamp* argument, which acts as a multiplier.

Notice that both *xamp* and *xfreq* can use k-rate as well as a-rate signals. Both arguments must use variables of the same rate, or *sfplay* will not work correctly. *ipreindex* must contain the number of a previously assigned preset, or Csound will crash.

The *ioffset* parameter allows the sound to start from a sample different than the first one. The user should make sure that its value is within the length of the specific sound. Otherwise, Csound will proba-

bly crash.

The *ienv* parameter enables and determines the type of amplitude envelope used. The default value is 0, or no envelope. If *ienv* is set to 1, the attack and decay portions are linear. If set to 2, the attack is linear and the decay is exponential. The release portion of the envelope has not yet been implemented.

sfplaym is a mono version of *sfplay*. It should be used with mono preset, or with the stereo presets in which stereo output is not required. It is faster than *sfplay*.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplay*, *sfplay*, *sfplist*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

New optional parameter *ienv* in version 5.09

sfplist

sfplist — Prints a list of all presets of a SoundFont2 (SF2) sample file.

Description

Prints a list of all presets of a previously loaded SoundFont2 (SF2) sample file. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

Syntax

```
sfplist ifilhandle
```

Initialization

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

Performance

sfplist prints a list of all presets of a previously loaded SF2 file to the console.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfpreset*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

sfpreset

sfpreset — Assigns an existing preset of a SoundFont2 (SF2) sample file to an index number.

Description

Assigns an existing preset of a previously loaded SoundFont2 (SF2) sample file to an index number. These opcodes allow management the sample-structure of SF2 files. In order to understand the usage of these opcodes, the user must have some knowledge of the SF2 format, so a brief description of this format can be found in the *SoundFont2 File Format Appendix*.

sfpreset should be placed in the header section of a Csound orchestra.

Syntax

```
ir sfpreset iprog, ibank, ifilhandle, ipreindex
```

Initialization

ir -- output to be used by other SF2 opcodes. For *sfpreset*, *ir* is *ipreindex*.

iprog -- program number of a bank of presets in a SF2 file

ibank -- number of a specific bank of a SF2 file

ifilhandle -- unique number generated by *sfload* opcode to be used as an identifier for a SF2 file. Several SF2 files can be loaded and activated at the same time.

ipreindex -- preset index

Performance

sfpreset assigns an existing preset of a previously loaded SF2 file to an index number, to be used later with the opcodes *sfplay* and *sfplaym*. The user must previously know the program and the bank numbers of the preset in order to fill the corresponding arguments. Any number of *sfpreset* instances can be placed in the header section of an orchestra, each one assigning a different preset belonging to the same (or different) SF2 file to different index numbers.

These opcodes only support the sample structure of SF2 files. The modulator structure of the SoundFont2 format is not supported in Csound. Any modulation or processing to the sample data is left to the Csound user, bypassing all restrictions forced by the SF2 standard.

See Also

sfilist, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*

Credits

Author: Gabriel Maldonado
Italy
May 2000

New in Csound Version 4.07

shaker

shaker — Produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse.

Description

La sortie audio produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse. La méthode est inspirée d'un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

Syntaxe

```
ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]
```

Initialisation

idecay -- S'il est présent, indique la durée d'amortissement du shaker à la fin de la note. La valeur par défaut est zéro.

Exécution

Une note jouée sur un instrument de type maracas, avec les arguments suivants.

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note.

kbeans -- Le nombre de graines dans la calebasse. Une valeur de 8 est convenable.

kdamp -- La valeur d'amortissement du shaker. Des valeurs comprises entre 0,98 et 1 conviennent, avec une valeur raisonnable par défaut de 0,99.

ktimes -- Nombre de secousses.

Note

L'argument *knum* était redondant et a donc été supprimé dans la version 3.49.

Exemples

Voici un exemple de l'opcode shaker. Il utilise le fichier *shaker.csd* [examples/shaker.csd].

Exemple 468. Exemple de l'opcode shaker.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o shaker.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
   a1 shaker 10000, 440, 8, 0.999, 100, 0
   out a1
endin


</CsInstruments>
<CsScore>

i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 3.47 de Csound

Exemple corrigé grâce à un message de Istvan Varga.

sin

sin — Calcule une fonction sinus.

Description

Retourne sinus de *x* (*x* en radians).

Syntaxe

```
sin(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode sin. Il utilise le fichier *sin.csd* [examples/sin.csd].

Exemple 469. Exemple de l'opcode sin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = sin(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = -0.132
```

Voir Aussi

cos, *cosh*, *cosinv*, *sinh*, *sininv*, *tan*, *tanh*, *taninv*

Crédits

Exemple écrit par Kevin Conder.

sinh

sinh — Calcule une fonction sinus hyperbolique.

Description

Retourne sinus hyperboique de *x* (*x* en radians).

Syntaxe

```
sinh(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode sinh. Il utilise le fichier *sinh.csd* [examples/sinh.csd].

Exemple 470. Exemple de l'opcode sinh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sinh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = sinh(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.175
```

Voir Aussi

cos, *cosh*, *cosinv*, *sin*, *sininv*, *tan*, *tanh*, *taninv*

Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47

sininv

sininv — Calcule une fonction arcsinus.

Description

Retourne arcsinus de *x* (*x* en radians).

Syntaxe

```
sininv(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode sininv. Il utilise le fichier *sininv.csd* [examples/sininv.csd].

Exemple 471. Exemple de l'opcode sininv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sininv.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = sininv(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.524
```

Voir Aussi

cos, *cosh*, *cosinv*, *sin*, *sinh*, *tan*, *tanh*, *taninv*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

sinsyn

sinsyn — Streaming partial track additive synthesis with cubic phase interpolation

Description

The sinsyn opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by the partials opcode). It sinsynthesises the signal using linear amplitude and cubic phase interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls. Sinsyn attempts to preserve the phase of the partials in the original signal and in so doing it does not allow for pitch or timescale modifications of the signal.

Syntax

```
asig sinsyn fin, kscal,  kmaxtracks, ifn
```

Performance

asig -- output audio rate signal

fin -- input pv stream in TRACKS format

kscal -- amplitude scaling

kmaxtracks -- max number of tracks in sinsynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

ifn -- function table containing one cycle of a sinusoid (sine or cosine)

Examples

Exemple 472. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
      aout  sinsyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis.

Credits

Author: Victor Lazzarini;
June 2005

New plugin in version 5

November 2004.

sleighbells

sleighbells — Modèle semi-physique d'un son de cloche de traineau.

Description

sleighbells est un modèle semi-physique d'un son de cloche de traineau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]
```

Initialisation

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,9994 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,9994 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,03.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

ifreq (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500.

ifreq1 (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5300.

ifreq2 (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 6500.

Exécution

kamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

Exemples

Voici un exemple de l'opcode sleighbells. Il utilise le fichier *sleighbells.csd* [examples/sleighbells.csd].

Exemple 473. Exemple de l'opcode sleighbells.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sleighbells.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1: An example of sleighbells.
instr 1
  a1 sleighbells 20000, 0.01

  out a1
endin


</CsInstruments>
<CsScore>

i 1 0.00 0.25
i 1 0.30 0.25
i 1 0.60 0.25
i 1 0.90 0.25
i 1 1.20 0.25
i 1 1.50 0.25
i 1 1.80 0.25
i 1 2.10 0.25
i 1 2.40 0.25
i 1 2.70 0.25
i 1 3.00 0.25
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

bamboo, *dripwater*, *guiro*, *tambourine*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

slider16

slider16 — Creates a bank of 16 different MIDI control message numbers.

Description

Creates a bank of 16 different MIDI control message numbers.

Syntax

```
i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum16, imin16, imax16, init16, ifn16
```

```
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum16, imin16, imax16, init16, ifn16
```

Initialization

i1 ... i16 -- output values

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum16 -- MIDI control number (0-127)

imin1 ... imin16 -- minimum values for each controller

imax1 ... imax16 -- maximum values for each controller

init1 ... init16 -- initial value for each controller

ifn1 ... ifn16 -- function table for conversion for each controller

icutoff1 ... icutoff16 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k16 -- output values

slider16 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider16 allows a bank of 16 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider16*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

See Also

s16b14, *s32b14*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider16f

slider16f — Creates a bank of 16 different MIDI control message numbers, filtered before output.

Description

Creates a bank of 16 different MIDI control message numbers, filtered before output.

Syntax

```
k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
      icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

Initialization

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum16 -- MIDI control number (0-127)

imin1 ... imin16 -- minimum values for each controller

imax1 ... imax16 -- maximum values for each controller

init1 ... init16 -- initial value for each controller

ifn1 ... ifn16 -- function table for conversion for each controller

icutoff1 ... icutoff16 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k16 -- output values

slider16f is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider16f allows a bank of 16 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

Avertissement

slider16f does not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

s16b14, *s32b14*, *slider16*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider32

slider32 — Creates a bank of 32 different MIDI control message numbers.

Description

Creates a bank of 32 different MIDI control message numbers.

Syntax

```
i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum32, imin32, imax32, init32, ifn32
```

```
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum32, imin32, imax32, init32, ifn32
```

Initialization

i1 ... i32 -- output values

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum32 -- MIDI control number (0-127)

imin1 ... imin32 -- minimum values for each controller

imax1 ... imax32 -- maximum values for each controller

init1 ... init32 -- initial value for each controller

ifn1 ... ifn32 -- function table for conversion for each controller

icutoff1 ... icutoff32 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k32 -- output values

slider32 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider32 allows a bank of 32 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider32*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider32f

slider32f — Creates a bank of 32 different MIDI control message numbers, filtered before output.

Description

Creates a bank of 32 different MIDI control message numbers, filtered before output.

Syntax

```
k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
      ..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

Initialization

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum32 -- MIDI control number (0-127)

imin1 ... imin32 -- minimum values for each controller

imax1 ... imax32 -- maximum values for each controller

init1 ... init32 -- initial value for each controller

ifn1 ... ifn32 -- function table for conversion for each controller

icutoff1 ... icutoff32 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k32 -- output values

slider32f is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider32f allows a bank of 32 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

Avertissement

slider32f opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider64

slider64 — Creates a bank of 64 different MIDI control message numbers.

Description

Creates a bank of 64 different MIDI control message numbers.

Syntax

```
i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum64, imin64, imax64, init64, ifn64
```

```
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum64, imin64, imax64, init64, ifn64
```

Initialization

i1 ... i64 -- output values

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

slider64 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider64 allows a bank of 64 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider64*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64f* *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider64f

slider64f — Creates a bank of 64 different MIDI control message numbers, filtered before output.

Description

Creates a bank of 64 different MIDI control message numbers, filtered before output.

Syntax

```
k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
      icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

Initialization

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

slider64f is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider64f allows a bank of 64 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

Avertissement

slider64f opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider8

slider8 — Creates a bank of 8 different MIDI control message numbers.

Description

Creates a bank of 8 different MIDI control message numbers.

Syntax

```
i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum8, imin8, imax8, init8, ifn8
```

```
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum8, imin8, imax8, init8, ifn8
```

Initialization

i1 ... i64 -- output values

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

slider8 is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider8 allows a bank of 8 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider8*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8f*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider8f

slider8f — Creates a bank of 8 different MIDI control message numbers, filtered before output.

Description

Creates a bank of 8 different MIDI control message numbers, filtered before output.

Syntax

```
k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
      ..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

Initialization

ichan -- MIDI channel (1-16)

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

k1 ... k64 -- output values

slider8f is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider8f allows a bank of 8 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

Avertissement

slider8f opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

s16b14, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*

Credits

Author: Gabriel Maldonado
Italy
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

slider16table

slider16table — Stores a bank of 16 different MIDI control messages to a table.

Description

Stores a bank of 16 different MIDI control messages to a table.

Syntax

```
kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, .... , ictlnum16, imin16, imax16, init16, ifn16
```

Initialization

i1 ... i16 -- output values

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum16 -- MIDI control number (0-127)

imin1 ... imin16 -- minimum values for each controller

imax1 ... imax16 -- maximum values for each controller

init1 ... init16 -- initial value for each controller

ifn1 ... ifn16 -- function table for conversion for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider16table is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider16table allows a bank of 16 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider16table is very similar to *slider16* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

See Also

slider16tablef, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slidertable8*, *slider8tablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider16tablef

slider16tablef — Stores a bank of 16 different MIDI control messages to a table, filtered before output.

Description

Stores a bank of 16 different MIDI control messages to a table, filtered before output.

Syntax

```
kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

Initialization

ichan -- MIDI channel (1-16)

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ioutTable -- number of the table that will contain the output

ictlnum1 ... ictlnum16 -- MIDI control number (0-127)

imin1 ... imin16 -- minimum values for each controller

imax1 ... imax16 -- maximum values for each controller

init1 ... init16 -- initial value for each controller

ifn1 ... ifn16 -- function table for conversion for each controller

icutoff1 ... icutoff16 -- low-pass filter cutoff frequency for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider16tablef is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

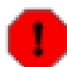
When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider16tablef allows a bank of 16 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider8table is very similar to *slider16tablef* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



Avertissement

slider16tablef does not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

slider16table, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider32table

slider32table — Stores a bank of 32 different MIDI control messages to a table.

Description

Creates a bank of 32 different MIDI control messages to a table.

Syntax

```
kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \
      imax1, init1, ifn1, .... , ictlnum32, imin32, imax32, init32, ifn32
```

Initialization

i1 ... i32 -- output values

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum32 -- MIDI control number (0-127)

imin1 ... imin32 -- minimum values for each controller

imax1 ... imax32 -- maximum values for each controller

init1 ... init32 -- initial value for each controller

ifn1 ... ifn32 -- function table for conversion for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider32table is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider32table allows a bank of 32 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider32table is very similar to *slider32* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

See Also

slider16table, *slider16tablef*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider32tablef

slider32tablef — Creates a bank of 32 different MIDI control message numbers, filtered before output.

Description

Creates a bank of 32 different MIDI control message numbers, filtered before output.

Syntax

```
kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

Initialization

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... *ictlnum32* -- MIDI control number (0-127)

imin1 ... *imin32* -- minimum values for each controller

imax1 ... *imax32* -- maximum values for each controller

init1 ... *init32* -- initial value for each controller

ifn1 ... *ifn32* -- function table for conversion for each controller

icutoff1 ... *icutoff32* -- low-pass filter cutoff frequency for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider32tablef is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

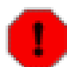
When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider32tablef allows a bank of 32 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider32tablef is very similar to *slider32tablef* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



Avertissement

slider32tablef opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

slider16, *slider16f*, *slider32*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider64table

slider64table — Stores a bank of 64 different MIDI control messages to a table.

Description

Creates a bank of 64 different MIDI control messages to a table.

Syntax

```
kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \
      imax1, init1, ifn1, .... , ictlnum64, imin64, imax64, init64, ifn64
```

Initialization

i1 ... i64 -- output values

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider64table is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider64table allows a bank of 64 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider64table is very similar to *slider64* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

See Also

slider16table, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64tablef* *slider8table*, *slider8tablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider64tablef

slider64tablef — Stores a bank of 64 different MIDI control messages to a table, filtered before output.

Description

Stores a bank of 64 different MIDI MIDI control messages to a table, filtered before output.

Syntax

```
kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

Initialization

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum64 -- MIDI control number (0-127)

imin1 ... imin64 -- minimum values for each controller

imax1 ... imax64 -- maximum values for each controller

init1 ... init64 -- initial value for each controller

ifn1 ... ifn64 -- function table for conversion for each controller

icutoff1 ... icutoff64 -- low-pass filter cutoff frequency for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider64tablef is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

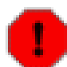
When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider64tablef allows a bank of 64 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider64tablef is very similar to *slider64tablef* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderN-table(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



Avertissement

slider64tablef opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

slider16table, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider8table*, *slider8tablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider8table

slider8table — Stores a bank of 8 different MIDI control messages to a table.

Description

Stores a bank of 8 different MIDI control messages to a table.

Syntax

```
kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8
```

Initialization

i1 ... i8 -- output values

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum8 -- MIDI control number (0-127)

imin1 ... imin8 -- minimum values for each controller

imax1 ... imax8 -- maximum values for each controller

init1 ... init8 -- initial value for each controller

ifn1 ... ifn8 -- function table for conversion for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider8table handles a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider8table allows a bank of 8 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider8table is very similar to *slider8* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderN-table(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

See Also

slider16table, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8tabletablef*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

slider8tablef

slider8tablef — Stores a bank of 8 different MIDI control messages to a table, filtered before output.

Description

Stores a bank of 8 different MIDI control messages to a table, filtered before output.

Syntax

```
kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

Initialization

ichan -- MIDI channel (1-16)

ioutTable -- number of the table that will contain the output

ioffset -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

ictlnum1 ... ictlnum8 -- MIDI control number (0-127)

imin1 ... imin8 -- minimum values for each controller

imax1 ... imax8 -- maximum values for each controller

init1 ... init8 -- initial value for each controller

ifn1 ... ifn8 -- function table for conversion for each controller

icutoff1 ... icutoff8 -- low-pass filter cutoff frequency for each controller

Performance

kflag -- a flag that informs if any control-change message in the bank has been received. In this case kflag is set to 1 is set to 1. Otherwise is set to zero.

slider8tablef is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

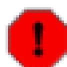
When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

slider8tablef allows a bank of 8 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

slider8tablef is very similar to *slider8f* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in sliderNtable(f), you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



Avertissement

slider8tablef opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

See Also

slider16table, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

sliderKawai

sliderKawai — Creates a bank of 16 different MIDI control message numbers from a KAWAI MM-16 midi mixer.

Description

Creates a bank of 16 different MIDI control message numbers from a KAWAI MM-16 midi mixer.

Syntax

```
k1, k2, ...., k16 sliderKawai imin1, imax1, init1, ifn1, \
      imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16
```

Initialization

ictlnum1 ... ictlnum32 -- MIDI control number (0-127)

imin1 ... imin16 -- minimum values for each controller

imax1 ... imax16 -- maximum values for each controller

init1 ... init16 -- initial value for each controller

ifn1 ... ifn16 -- function table for conversion for each controller

Performance

k1 ... k16 -- output values

The opcode *sliderKawai* is equivalent to *slider16*, but it has the controller and channel numbers (*ichan* and *ictlnum*) hard-coded to make for quick compatiblity with the KAWAI MM-16 midi mixer. This device doesn't allow changing the midi message associated to each slider. It can only output on control 7 for each fader on a separate midi channel. This opcode is a quick way of assigning the mixer's 16 faders to k-rate variables in csound.

See Also

slider16, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

Credits

Author: Gabriel Maldonado

New in Csound version 5.06

sndload

sndload — Loads a sound file into memory for use by *loscilx*

Description

sndload loads a sound file into memory for use by *loscilx*.

Syntax

```
sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istrt   \
      [, ilpmod[, ilps[, ilpe]]]]]]]]]
```

Initialization

Sfname - file name as a string constant or variable, string p-field, or a number that is used either as an index to strings set with strset, or, if that is not available, a fine name in the format soundin.n is used. If the file name does not include a full path, the file is searched in the current directory first, then those specified by SSDIR (if defined), and finally SFDIR. If the same file was already loaded previously, it will not be read again, but the parameters ibas, iamp, istrt, ilpmod, ilps, and ilpe are still updated.

ifmt (optional, defaults to zero) - default sample format for raw (headerless) sound files; if the file has a header, this is ignored. Can be one of the following:

-1: do not allow headerless files (fail with an init error)
0: use the same format as the one specified on the command line
1: 8 bit signed integers
2: a-law
3: u-law
4: 16 bit signed integers
5: 32 bit signed integers
6: 32 bit floats
7: 8 bit unsigned integers
8: 24 bit signed integers
9: 64 bit floats

ichns (optional, defaults to zero) - default number of channels for raw (headerless) sound files; if the file has a header, this is ignored. Zero or negative values are interpreted as 1 channel.

isr (optional, defaults to zero) - default sample rate for raw (headerless) sound files; if the file has a header, this is ignored. Zero or negative values are interpreted as the orchestra sample rate (sr).

ibas (optional, defaults to zero) - base frequency in Hz. If positive, overrides the value specified in the sound file header; otherwise, the value from the header is used if present, and 1.0 if the file does not include such information.

iamp (optional, defaults to zero) - amplitude scale. If non-zero, overrides the value specified in the sound file header (note: negative values are allowed, and will invert the sound output); otherwise, the value from the header is used if present, and 1.0 if the file does not include such information.

istrt (optional, defaults to -1) - starting position in sample frames, can be fractional. If non-negative, overrides the value specified in the sound file header; otherwise, the value from the header is used if present, and 0 if the file does not include such information. Note: even if this parameter is specified, the whole file is still read into memory.

ilpmod (optional, defaults to -1) - loop mode, can be one of the following:

any negative value: use the loop information specified in the sound file header, ignoring ilps and ilpe
0: no looping (ilps and ilpe are ignored)
1: forward looping (wrap around loop end if it is crossed in forward direction, and wrap around loop start if it is crossed in backward direction)
2: backward looping (change direction at loop end if it is crossed in forward direction, and wrap around loop start if it is crossed in backward direction)
3: forward-backward looping (change direction at both loop points if they are crossed as described above)

ilps (optional, defaults to 0) - loop start in sample frames (fractional values are allowed), or loop end if ilps is greater than ilpe. Ignored unless ilpmod is set to 1, 2, or 3. If the loop points are equal, the whole sample is looped.

ilpe (optional, defaults to 0) - loop end in sample frames (fractional values are allowed), or loop start if ilps is greater than ilpe. Ignored unless ilpmod is set to 1, 2, or 3. If the loop points are equal, the whole sample is looped.

Credits

Written by Istvan Varga.

2006

New in Csound 5.03

sndloop

sndloop — A sound looper with pitch control.

Description

This opcode records input audio and plays it back in a loop with user-defined duration and crossfade time. It also allows the pitch of the loop to be controlled, including reversed playback.

Syntax

```
asig, krec sndloop ain, kpitch, ktrig, idur, ifad
```

Initialisation

idur -- loop duration in seconds

ifad -- crossfade duration in seconds

Performance

asig -- output sig

krec -- 'rec on' signal, 1 when recording, 0 otherwise

kpitch -- pitch control (transposition ratio); negative values play the loop back in reverse

ktrig -- trigger signal: when 0, processing is bypassed. When switched on (ktrig >= 1), the opcode starts recording until the loop memory is full. It then plays the looped sound until it is switched off again (ktrig = 0). Another recording can start again with ktrig >= 1.

Examples

Exemple 474. Example

```
asig  in                                 ; get the signal in
ktrig line 0, 1, 1                  ; trigger signal
aout,krec sndloop asig, 1, ktrig, 4, 0.05  ;  rec starts at 1 sec, for 4 secs 0.05 crossfade
printk  1, krec                               ; prints the recording signal
       out aout
```

The example above shows the basic operation of sndloop. Pitch can be controlled at the k-rate, recording is started as soon as the trigger value is >= 1. Recording can be restarted by making the trigger 0 and then 1 again.

Credits

Author: Victor Lazzarini;
April 2005

New in Version 5.00

sndwarp

sndwarp — Lit un son mono échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

Description

sndwarp lit des échantillons sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec *iwsize=sr*/10 et *ioverlap*=15. Essayer *irandw=iwsize**0,2. Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

Syntaxe

```
ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsize, \
      irandw, ioverlap, ifn2, itimemode
```

Initialisation

ifn1 -- le numéro de la table contenant les échantillons qui seront traités par *sndwarp*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

ibeg -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

iwsize -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

irandw -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsize*.

ioverlap -- détermine la densité de fenêtres se chevauchant.

ifn2 -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinusoïde (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

Exécution

ares -- l'unique canal de sortie du générateur unitaire *sndwarp*. *sndwarp* suppose que la table de fonction contenant le signal échantillonné est monophonique. *sndwarp* indexera la table avec un incrément d'un seul échantillon. Il faut ainsi remarquer que si l'on utilise un signal stéréo avec *sndwarp*, la durée et la hauteur seront altérées en conséquence.

ac (facultatif) -- une version mono-couche (pas de superpositions), et non fenêtrée du signal modifié en durée et/ou en hauteur. Elle est fournie afin de permettre de pondérer l'amplitude du signal de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal, avec une version

épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarp* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on l'utilise avec une unité *balance*, *ac* permet d'améliorer grandement la qualité sonore.

xamp -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac*).

xtimewarp -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

xresample -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

Exemples

Voici en exemple de l'opcode sndwarp. Il utilise les fichiers *sndwarp.csd* [examples/sndwarp.csd] et *mary.wav* [examples/mary.wav].

Exemple 475. Exemple de l'opcode sndwarp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sndwarp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
  ; Use the audio file defined in Table #1.
  a1 loscil 30000, 1, 1, 1

  out a1
endin

; Instrument #2 - time-stretch an audio file.
instr 2
  kamp init 6500
  ; Start at 1 second and end at 3.5 seconds.
  ktimewarp line 1, p3, 3.5
  ; Playback at the normal speed.
  kresample init 1
  ; Use the audio file defined in Table #1.
  ifn1 = 1
  ibeg = 0
  iwsize = 4410
  irandw = 882
  ioverlap = 15
  ; Use Table #2 for the windowing function.
  ifn2 = 2
```

```
  ; Use the ktimewarp parameter as a "time" pointer.
  itimemode = 1

  a1 sndwarp kamp, ktimewarp, kresample, ifn1, ibeg, iwsize, irandw, ioverlap, ifn2, itimemode
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0
; Table #2: half of a sine wave.
f 2 0 16384 9 0.5 1 0

; Play Instrument #1 for 3.5 seconds.
i 1 0 3.5
; Play Instrument #2 for 7 seconds (time-stretched).
i 2 3.5 10.5
e


</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```
iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp   line    1, p3, 1
aresamp line    1, p3, 2
kenv    line    1, p3, .1
asig    sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0
```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```
itimemode     =          1
atime         line       0, p3, 10
ar1, ar2      sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
                         iwindfun, itimemode
```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si p3 vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```
asig, acmp  sndwarp 1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itimem
abal        balance asig, acmp

asig1,asig2,acmp1,acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
                                  iwindfun, itimemode
abal1       balance asig1, acmp1
abal2       balance asig2, acmp2
```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.

Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

Voir Aussi

sndwarpst

Crédits

Auteur : Richard Karpen
Seattle, WA USA
1997

Exemple écrit par Kevin Conder.

sndwarpst

sndwarpst — Lit un son stéréo échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

Description

sndwarpst lit des échantillons stéréo sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec *iwsize=sr*/10 et *ioverlap*=15. Essayer *irandw=iwsize**0,2. Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

Syntaxe

```
ar1, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \
      ibeg, iwsize, irandw, ioverlap, ifn2, itimemode
```

Initialisation

ifn1 -- le numéro de la table contenant les échantillons qui seront traités par *sndwarpst*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

ibeg -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

iwsize -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

irandw -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsize*.

ioverlap -- détermine la densité de fenêtres se chevauchant.

ifn2 -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinusoïde (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

Exécution

ar1, ar2 -- *ar1* et *ar2* sont les sorties stéréo (gauche et droite) de *sndwarpst*. *sndwarpst* suppose que la table de fonction contenant le signal échantillonné est stéréophonique. *sndwarpst* indexera la table avec un incrément de deux échantillons. Il faut ainsi remarquer que si l'on utilise un signal mono avec *sndwarpst*, la durée et la hauteur seront altérées en conséquence.

ac1, ac2 -- *ac1* et *ac2* sont des versions mono-couche (pas de superpositions), et non fenêtrées du signal modifié en durée et/ou en hauteur. Elles sont fournies afin de permettre de pondérer l'amplitude du signal de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal,

avec une version épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarpst* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on les utilise avec une unité *balance*, *ac1* et *ac2* permettent d'améliorer grandement la qualité sonore. Ils sont facultatifs mais il faut noter que la syntaxe exige la présence des deux arguments (utiliser les deux ou aucun). Un exemple de leur utilisation est donné ci-dessous.

xamp -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac1* et *ac2*).

xtimewarp -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

xresample -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

Exemples

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```
iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp   line    1, p3, 1
aresamp line    1, p3, 2
kenv    line    1, p3, .1
asig    sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0
```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```
itimemode     =         1
atime         line      0, p3, 10
ar1, ar2      sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
                        iwindfun, itimemode
```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si p3 vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```
asig,acmp     sndwarp  1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itime
abal          balance asig, acmp

asig1,asig2,acmp1,acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
                                  iwindfun, itimemode
abal1         balance asig1, acmp1
```

```
abal2       balance asig2, acmp2
```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.

Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

Voir Aussi

sndwarp

Crédits

Auteur : Richard Karpen
Seattle, WA USA
1997

socksend

socksend — Sends data to other processes using the low-level UDP or TCP protocols

Description

Transmits data directly using the UDP (socksend and socksends) or TCP (stsend) protocol onto a network. The data is not subject to any encoding or special routing. The socksends opcode send a stereo signal interleaved.

Syntax

```
socksend asig, Sipaddr, iport, ilength
```

```
socksends asigl, asigr, Sipaddr, iport,
    ilength
```

```
stsend asig, Sipaddr, iport
```

Initialization

Sipaddr -- a string that is the IP address of the receiver in standard 4-octet dotted form.

iport -- the number of the port that is used for the communication.

ilength -- the length of the individual packets in UDP transmission. This number must be sufficiently small to fit a single MTU, which is set to the save value of 1456. In UDP transmissions the receiver needs to know this value

Performance

asig, asigl, asigr -- audio data to be transmitted.

Example

The example shows a simple sine wave being sent just once to a computer called "172.16.0.255", on port 7777 using UDP. Note that .255 is often used for broadcasting.

```
sr = 44100
ksmps = 100
nchnls = 1

instr   1
a1 oscil        20000,441,1
   socksend     a1, "172.16.0.255",7777, 200
endin
```

Credits

Author: John ffitch
2006

sockrecv

sockrecv — Receives data from other processes using the low-level UDP or TCP protocols

Description

Receives directly using the UDP (sockrecv and sockrecvs) or TCP (strecv) protocol onto a network. The data is not subject to any encoding or special routing. The sockrecvs opcode receives a stereo signal interleaved.

Syntax

```
asig sockrecv iport, ilength
```

```
asigl, asigr sockrecvs iport, ilength
```

```
asig strecv Sipaddr, iport
```

Initialization

Sipaddr -- a string that is the IP address of the sender in standard 4-octet dotted form.

iport -- the number of the port that is used for the communication.

ilength -- the length of the individual packets in UDP transmission. This number must be sufficiently small to fit a single MTU, which is set to the save value of 1456. In UDP transmissions the sender and receiver needs agree on this value

Performance

asig, asigl, asigr -- audio data to be received.

Example

The example shows a mono signal being received on port 7777 using UDP.

```
sr = 44100
ksmps = 100
nchnls = 1


instr   1
a1 sockrecv     7777, 200
   out   a1
endin
```

Credits

Author: John ffitch

2006

soundin

soundin — Reads audio data from an external device or stream.

Description

Reads audio data from an external device or stream. Up to 24 channels may be read.

Syntax

```
ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskptim] [, iformat] \
      [, iskipinit] [, ibufsize]
```

Initialization

ifilcod -- integer or character-string denoting the source soundfile name. An integer denotes the file soundin.filcod; a character-string (in double quotes, spaces permitted) gives the filename itself, optionally a full pathname. If not a full path, the named file is sought first in the current directory, then in that given by the environment variable SSDIR (if defined) then by SFDIR. See also *GEN01*.

iskptim (optional, default=0) -- time in seconds of input sound to be skipped. The default value is 0. In csound 5.00 and later, this may be negative to add a delay instead of skipping time.

iformat (optional, default=0) -- specifies the audio data file format:

- 1 = 8-bit signed char (high-order 8 bits of a 16-bit integer)
- 2 = 8-bit A-law bytes
- 3 = 8-bit U-law bytes
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats
- 7 = 8-bit unsigned int (not available in Csound versions older than 5.00)
- 8 = 24-bit int (not available in Csound versions older than 5.00)
- 9 = 64-bit doubles (not available in Csound versions older than 5.00)

iskipinit -- switches off all initialisation if non zero (default=0). This was introduced in 4_23f13 and csound5.

ibufsize -- buffer size in mono samples (not sample frames). Not available in Csound versions older than 5.00. The default buffer size is 2048.

If *iformat* = 0 it is taken from the soundfile header, and if no header from the Csound *-o* command-line flag. The default value is 0.

Performance

soundin is functionally an audio generator that derives its signal from a pre-existing file. The number of channels read in is controlled by the number of result cells, a1, a2, etc., which must match that of the input file. A *soundin* opcode opens this file whenever the host instrument is initialized, then closes it again each time the instrument is turned off.

There can be any number of *soundin* opcodes within a single instrument or orchestra. Two or more of them can read simultaneously from the same external file.

Note to Windows users

Windows users typically use back-slashes, « \ », when specifying the paths of their files. As an example, a Windows user might use the path « c:\music\samples\loop001.wav ». This is problematic because back-slashes are normally used to specify special characters.

To correctly specify this path in Csound, one may alternately:

- *Use forward slashes*: c:/music/samples/loop001.wav
- *Use back-slash special characters, «* \\ *»*: c:\\music\\samples\\loop001.wav

Examples

Here is an example of the soundin opcode. It uses the file *soundin.csd* [examples/soundin.csd], *beats.wav* [examples/beats.wav].

Exemple 476. Example of the soundin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o soundin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
  asig soundin "beats.wav"
  out asig
endin


</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
```

```
e

</CsScore>
</CsoundSynthesizer>
```

See Also

diskin, *in*, *inh*, *ino*, *inq*, *ins*

Credits

Authors: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

Example written by Kevin Conder.

Warning to Windows users added by Kevin Conder, April 2002

soundout

soundout — Deprecated. Writes audio output to a disk file.

Description

Note

The usage of *soundout* is discouraged. Please use *fout* instead.

Writes audio output to a disk file.

Syntax

```
soundout  asig1, ifilcod [, iformat]
```

Initialization

ifilcod -- integer or character-string denoting the destination soundfile name. An integer denotes the file soundin.filcod; a character-string (in double quotes, spaces permitted) gives the filename itself, optionally a full pathname. If not a full path, the named file is sought first in the current directory, then in that given by the environment variable SSDIR (if defined) then by SFDIR. See also *GEN01*.

iformat (optional, default=0) -- specifies the audio data file format:

- 1 = 8-bit signed char (high-order 8 bits of a 16-bit integer)
- 2 = 8-bit A-law bytes
- 3 = 8-bit U-law bytes
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats

If *iformat* = 0 it is taken from the soundfile header, and if no header from the Csound *-o* command-line flag. The default value is 0.

Performance

soundout writes audio output to a disk file.

Note

Use of *fout* is recommended instead of *soundout*

See Also

fout, *out*, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2* *soundouts*

Credits

Author: Barry L. Vercoe, Matt Ingalls/Mike Berry
MIT, Mills College
1993-1997

soundouts

soundouts — Deprecated. Writes audio output to a disk file.

Description

Note

The usage of *soundouts* is discouraged. Please use *fout* instead.

Writes audio output to a disk file.

Syntax

```
soundouts  asigl, asigr, ifilcod [, iformat]
```

Initialization

ifilcod -- integer or character-string denoting the destination soundfile name. An integer denotes the file soundout.ifilcod; a character-string (in double quotes, spaces permitted) gives the filename itself, optionally a full pathname. If not a full path, the named file is written relative to the directory given by the SFDIR environment variable if defined, or the current directory. See also *GEN01*.

iformat (optional, default=0) -- specifies the audio data file format:

- 1 = 8-bit signed char (high-order 8 bits of a 16-bit integer)
- 4 = 16-bit short integers
- 5 = 32-bit long integers
- 6 = 32-bit floats

If *iformat* = 0 it is taken from the Csound *-o* command-line flag. The default value is 0.

Performance

soundouts writes stereo audio output to a disk file in raw (headerless) format without 0dBFS scaling. The expected range of the audio signals depends on the selected sample format.

Note

Use of *fout* is recommended instead of *soundouts*

See Also

out, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2* *soundout*

Credits

Author: Istvan Varga

space

space — Distributes an input signal among 4 channels using cartesian coordinates.

Description

space takes an input signal and distributes it among 4 channels using Cartesian xy coordinates to calculate the balance of the outputs. The xy coordinates can be defined in a separate text file and accessed through a Function statement in the score using *Gen28*, or they can be specified using the optional *kx, ky* arguments. The advantages to the former are:

1. A graphic user interface can be used to draw and edit the trajectory through the Cartesian plane
2. The file format is in the form time1 X1 Y1 time2 X2 Y2 time3 X3 Y3 allowing the user to define a time-tagged trajectory

space then allows the user to specify a time pointer (much as is used for *pvoc*, *lpread* and some other units) to have detailed control over the final speed of movement.

Syntax

```
a1, a2, a3, a4  space asig, ifn, ktime, kreverbsend, kx, ky
```

Initialization

ifn -- number of the stored function created using *Gen28*. This function generator reads a text file which contains sets of three values representing the xy coordinates and a time-tag for when the signal should be placed at that location. The file should look like:

```
0      -1      1
1       1      1
2       4      4
2.1    -4     -4
3      10    -10
5     -40      0
```

If that file were named « move » then the *Gen28* call in the score would like:

```
f1 0 0 28 "move"
```

Gen28 takes 0 as the size and automatically allocates memory. It creates values to 10 milliseconds of resolution. So in this case there will be 500 values created by interpolating X1 to X2 to X3 and so on, and Y1 to Y2 to Y3 and so on, over the appropriate number of values that are stored in the function table. In the above example, the sound will begin in the left front, over 1 second it will move to the right front, over another second it move further into the distance but still in the left front, then in just 1/10th of a second it moves to the left rear, a bit distant. Finally over the last .9 seconds the sound will move to the

right rear, moderately distant, and it comes to rest between the two left channels (due west!), quite distant. Since the values in the table are accessed through the use of a time-pointer in the *space* unit, the actual timing can be made to follow the file's timing exactly or it can be made to go faster or slower through the same trajectory. If you have access to the GUI that allows one to draw and edit the files, there is no need to create the text files manually. But as long as the file is ASCII and in the format shown above, it doesn't matter how it is made!

Important

If *ifn* is 0, then *space* will take its values for the xy coordinates from *kx* and *ky*.

Performance

The configuration of the xy coordinates in space places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of xy can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. x=0, y=1, will place the signal equally balanced between left and right front channels, x=y=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the xy's are kept so that Y>=1, it should work well to do panning and fixed localization in a stereo field.

asig -- input audio signal.

ktime -- index into the table containing the xy coordinates. If used like:

```
ktime           line  0, 5, 5
a1, a2, a3, a4  space asig, 1, ktime, ...
```

with the file « move » described above, the speed of the signal's movement will be exactly as described in that file. However:

```
ktime           line  0, 10, 5
```

the signal will move at half the speed specified. Or in the case of:

```
ktime           line  5, 15, 0
```

the signal will move in the reverse direction as specified and 3 times slower! Finally:

```
ktime        line  2, 10, 3
```

will cause the signal to move only from the place specified in line 3 of the text file to the place specified in line 5 of the text file, and it will take 10 seconds to do it.

kreverbsend -- the percentage of the direct signal that will be factored along with the distance as derived from the XY coordinates to calculate signal amounts that can be sent to reverb units such as reverb, or reverb2.

kx, ky -- when *ifn* is 0, *space* and *spdist* will use these values as the XY coordinates to localize the signal.

Examples

```
instr 1
  asig    ;some audio signal
  ktime               line  0, p3, p10
  a1, a2, a3, a4      space asig,1, ktime, .1
  ar1, ar2, ar3, ar4 spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                      outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument

  a1 reverb2 ga1, 2.5, .5
  a2 reverb2 ga2, 2.5, .5
  a3 reverb2 ga3, 2.5, .5
  a4 reverb2 ga4, 2.5, .5

     outq a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
endin
```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

space can useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using xy values from the score instead of a function table.

```
instr 1
  ...
  a1, a2, a3, a4      space asig, 0, 0, .1, p4, p5
  ar1, ar2, ar3, ar4 spsend

  ga1=ga1+ar1
```

```
  ga2=ga2+ar2
                   outs  a1, a2
endin

instr 99 ; reverb....
  ....
endin
```

A few notes: p4 and p5 are the X and Y values

```
  ;place the sound in the left speaker and near
    i1 0 1 -1 1
  ;place the sound in the right speaker and far
    i1 1 1 45 45
  ;place the sound equally between left and right and in the middle ground distance
    i1 2 1 0 12
e
```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```
  ktime              line   0, p3, 10
  kdist              spdist 1, ktime
  kfreq = (ifreq * 340) / (340 + kdist)
  asig               oscili iamp, kfreq, 1

  a1, a2, a3, a4     space  asig, 1, ktime, .1
  ar1, ar2, ar3, ar4 spsend
```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

See Also

spdist, *spsend*

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48

spat3d

spat3d — Positions the input sound in a 3D space and allows moving the sound at k-rate.

Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. *spat3d* allows moving the sound at k-rate (this movement is interpolated internally to eliminate "zipper noise" if sr not equal to kr).

Syntax

```
aW, aX, aY, aZ spat3d ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]
```

Initialization

idist -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

amplitude = 1 / (0.1 + distance)

delay = distance / 340 (in seconds)

Distance can be calculated as:

distance = sqrt(iX^2 + iY^2 + iZ^2)

In Mode 4, distance can be calculated as:

distance from left mic = sqrt((iX + idist/2)^2 + iY^2 + iZ^2)
distance from right mic = sqrt((iX - idist/2)^2 + iY^2 + iZ^2)

With *spat3d* the distance between the sound source and any microphone should be at least (340 * 18) / sr meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

ift -- Function table storing room parameters (for free field spatialization, set it to zero or negative).

Table size is 54. The values in the table are:

Room Parameter	Purpose
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: N = (2*R + 2) * R. If all six walls are enabled: N = (((4*R + 6)*R + 8)*R) / 3
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if >= 0.
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of 1 / (wall distance))
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

imode -- Output mode

- 0: B format with W output only (mono)

  aout = aW

- 1: B format with W and Y output (stereo)

  aleft = aW + 0.7071*aY
  aright = aW - 0.7071*aY

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:

```
aWre, aWim      hilbert aW
aXre, aXim      hilbert aX
aYre, aYim      hilbert aY
aWXr    =  0.0928*aXre + 0.4699*aWre
aWXiYr  =  0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft   =  aWXr + aWXiYr
aright  =  aWXr - aWXiYr
```

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

```
aW      butterlp aW, ifreq      ; recommended values for ifreq
aY      butterlp aY, ifreq      ; are around 1000 Hz
aleft   =  aW + aX
aright  =  aY + aZ
```

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here: *http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm*

imdel -- Maximum delay time for spat3d in seconds. This has to be longer than the delay time of the latest reflection (depends on room dimensions, sound source distance, and recursion depth; using this formula gives a safe (although somewhat overestimated) value:

```
imdel = (R + 1) * sqrt(W*W + H*H + D*D) / 340.0
```

where R is the recursion depth, W, H, and D are the width, height, and depth of the room, respectively).

iovr -- Oversample ratio for spat3d (1 to 8). Setting it higher improves quality at the expense of memory and CPU usage. The recommended value is 2.

istor (optional, default=0) -- Skip initialization if non-zero (default: 0).

Performance

aW, aX, aY, aZ -- Output signals

	mode 0	mode 1	mode 2	mode 3	mode 4
aW	W out	W out	W out	W out	left chn / low freq.
aX	0	0	X out	X out	left chn / high frq.
aY	0	Y out	Y out	Y out	right chn / low frq.
aZ	0	0	0	Z out	right chn / high fr.

ain -- Input signal

kX, kY, kZ -- Sound source coordinates (in meters)

If you encounter very slow performance (up to 100 times slower), it may be caused by denormals (this is also true of many other IIR opcodes, including *butterlp*, *pareq*, *hilbert*, and many others). Underflows can be avoided by:

- Using the *denorm* opcode on *ain* before *spat3d*.
- mixing low level DC or noise to the input signal, e.g.

  atmp rnd31 1/1e24, 0, 0

  aW, aX, aY, aZ spa3di ain + atmp, ...

  or

  aW, aX, aY, aZ spa3di ain + 1/1e24, ...

- reducing *irlen* in the case of *spat3dt* (which does not have an input signal). A value of about 0.005 is suitable for most uses, although it also depends on EQ settings. If the equalizer is not used, « irlen » can be set to 0.

Examples

Here is a example of the spat3d opcode that outputs a stereo file. It uses the file *spat3d_stereo.csd* [examples/spat3d_stereo.csd].

Exemple 477. Stereo example of the spat3d opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_stereo.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      =  48000
kr      =  1000
ksmps   =  48
nchnls  =  2

/* room parameters */

idep    =  3    /* early reflection depth       */

itmp    ftgen   1, 0, 64, -2,                                       \
                /* depth1, depth2, max delay, IR length, idist, seed */ \
                idep, 48, -1, 0.01, 0.25, 123,                       \
                1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil  */ \
                1,  1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
                1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
                1,  9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back  */ \
                1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
                1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2  /* left  */

        instr 1

/* some source signal */

a1      phasor 150              ; oscillator
a1      butterbp a1, 500, 200   ; filter
a1      =  taninv(a1 * 100)
a2      phasor 3                ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      =  a1 * a2 * 9000

kazim   line 0, 2.5, 360        ; move sound source around
kdist   line 1, 10, 4           ; distance

; convert polar coordinates
kX      =  sin(kazim * 3.14159 / 180) * kdist
kY      =  cos(kazim * 3.14159 / 180) * kdist
kZ      =  0

a1      =  a1 + 0.000001 * 0.000001     ; avoid underflows

imode   =  1    ; change this to 3 for 8 spk in a cube,
                ; or 1 for simple stereo

aW, aX, aY, aZ  spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      =  aW * 1.4142

; stereo
;
aL      =  aW + aY              /* left                 */
aR      =  aW - aY              /* right                */

; quad (square)
;
;aFL     =  aW + aX + aY         /* front left           */
;aFR     =  aW + aX - aY         /* front right          */
;aRL     =  aW - aX + aY         /* rear left            */
;aRR     =  aW - aX - aY         /* rear right           */

; eight channels (cube)
;
;aUFL    =  aW + aX + aY + aZ    /* upper front left     */
;aUFR    =  aW + aX - aY + aZ    /* upper front right    */
;aURL    =  aW - aX + aY + aZ    /* upper rear left      */
;aURR    =  aW - aX - aY + aZ    /* upper rear right     */
;aLFL    =  aW + aX + aY - aZ    /* lower front left     */
;aLFR    =  aW + aX - aY - aZ    /* lower front right    */
;aLRL    =  aW - aX + aY - aZ    /* lower rear left      */
;aLRR    =  aW - aX - aY - aZ    /* lower rear right     */

        outs aL, aR

        endin

</CsInstruments>
```

```
<CsScore>

/* Written by Istvan Varga */
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

Here is a example of the spat3d opcode that outputs a UHJ file. It uses the file *spat3d_UHJ.csd* [examples/spat3d_UHJ.csd].

Exemple 478. UHJ example of the spat3d opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_UHJ.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr =  48000
kr =  750
ksmps =  64
nchnls =  2

itmp    ftgen   1, 0, 64, -2,                                              \
                /* depth1, depth2, max delay, IR length, idist, seed */ \
                3, 48, -1, 0.01, 0.25, 123,                             \
                1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil  */ \
                1,  1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
                1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
                1,  9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back  */ \
                1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
                1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2  /* left  */

        instr 1

p3 =  p3 + 1.0

kazim line 0.0, 4.0, 360.0          ; azimuth
kelev line 40, p3 - 1.0, -20          ; elevation
kdist =  2.0                          ; distance
; convert coordinates
kX =  kdist * cos(kelev * 0.01745329) * sin(kazim * 0.01745329)
kY =  kdist * cos(kelev * 0.01745329) * cos(kazim * 0.01745329)
kZ =  kdist * sin(kelev * 0.01745329)

; source signal
a1 phasor 160.0
a2 delay1 a1
a1 =  a1 - a2
kffrq1 port 200.0, 0.8, 12000.0
affrq upsamp kffrq1
affrq pareq affrq, 5.0, 0.0, 1.0, 2
kffrq downsamp affrq
aenv4 phasor 3.0
aenv4 limit 2.0 - aenv4 * 8.0, 0.0, 1.0
a1 butterbp a1 * aenv4, kffrq, 160.0
aenv linseg 1.0, p3 - 1.0, 1.0, 0.04, 0.0, 1.0, 0.0
a_ =  4000000 * a1 * aenv + 0.00000001

; spatialize
a_W, a_X, a_Y, a_Z spat3d a_, kX, kY, kZ, 1.0, 1, 2, 2.0, 2

; convert to UHJ format (stereo)
aWre, aWim hilbert a_W
aXre, aXim hilbert a_X
aYre, aYim hilbert a_Y
```

```
aWXre =  0.0928*aXre + 0.4699*aWre
aWXim =  0.2550*aXim - 0.1710*aWim

aL =  aWXre + aWXim + 0.3277*aYre
aR =  aWXre - aWXim - 0.3277*aYre

        outs aL, aR

        endin


</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60

i 1 0.0 8.0
e


</CsScore>
</CsoundSynthesizer>
```

Here is a example of the spat3d opcode that outputs a quadrophonic file. It uses the file *spat3d_quad.csd* [examples/spat3d_quad.csd].

Exemple 479. Quadrophonic example of the spat3d opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac          -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_quad.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      =  48000
kr      =  1000
ksmps   =  48
nchnls  =  4

/* room parameters */

idep    =  3    /* early reflection depth       */

itmp    ftgen   1, 0, 64, -2,                                   \
                /* depth1, depth2, max delay, IR length, idist, seed */ \
                idep, 48, -1, 0.01, 0.25, 123,                   \
                1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil  */ \
                1,  1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
                1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
                1,  9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back  */ \
                1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
                1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2  /* left  */

        instr 1

/* some source signal */

a1      phasor 150              ; oscillator
a1      butterbp a1, 500, 200   ; filter
a1      =  taninv(a1 * 100)
a2      phasor 3                ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      =  a1 * a2 * 9000

kazim   line 0, 2.5, 360        ; move sound source around
kdist   line 1, 10, 4           ; distance
```

```
; convert polar coordinates
kX      =  sin(kazim * 3.14159 / 180) * kdist
kY      =  cos(kazim * 3.14159 / 180) * kdist
kZ      =  0

a1      =  a1 + 0.000001 * 0.000001     ; avoid underflows

imode   =  2    ; change this to 3 for 8 spk in a cube,
                ; or 1 for simple stereo

aW, aX, aY, aZ  spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      =  aW * 1.4142

; stereo
;
;aL     =  aW + aY              /* left                 */
;aR     =  aW - aY              /* right                */

; quad (square)
;
aFL     =  aW + aX + aY         /* front left           */
aFR     =  aW + aX - aY         /* front right          */
aRL     =  aW - aX + aY         /* rear left            */
aRR     =  aW - aX - aY         /* rear right           */

; eight channels (cube)
;
;aUFL   =  aW + aX + aY + aZ    /* upper front left     */
;aUFR   =  aW + aX - aY + aZ    /* upper front right    */
;aURL   =  aW - aX + aY + aZ    /* upper rear left      */
;aURR   =  aW - aX - aY + aZ    /* upper rear right     */
;aLFL   =  aW + aX + aY - aZ    /* lower front left     */
;aLFR   =  aW + aX - aY - aZ    /* lower front right    */
;aLRL   =  aW - aX + aY - aZ    /* lower rear left      */
;aLRR   =  aW - aX - aY - aZ    /* lower rear right     */

        outq aFL, aFR, aRL, aRR

        endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60
i 1 0 10
e


</CsScore>
</CsoundSynthesizer>
```

See Also

spat3di, *spat3dt*

Credits

Author: Istvan Varga
2001

New in version 4.12

Updated April 2002 by Istvan Varga

spat3di

spat3di — Positions the input sound in a 3D space with the sound source position set at i-time.

Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. With *spat3di*, sound source position is set at i-time.

Syntax

```
aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]
```

Initialization

iX -- Sound source X coordinate in meters (positive: right, negative: left)

iY -- Sound source Y coordinate in meters (positive: front, negative: back)

iZ -- Sound source Z coordinate in meters (positive: up, negative: down)

idist -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

amplitude = 1 / (0.1 + distance)

delay = distance / 340 (in seconds)

Distance can be calculated as:

distance = sqrt(iX^2 + iY^2 + iZ^2)

In Mode 4, distance can be calculated as:

distance from left mic = sqrt((iX + idist/2)^2 + iY^2 + iZ^2)
distance from right mic = sqrt((iX - idist/2)^2 + iY^2 + iZ^2)

With *spat3d* the distance between the sound source and any microphone should be at least (340 * 18) / sr meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

ift -- Function table storing room parameters (for free field spatialization, set it to zero or negative). Table size is 54. The values in the table are:

Room Parameter	Purpose
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: N = (2*R + 2) * R. If all six walls are enabled: N = (((4*R + 6)*R + 8)*R) / 3
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if >= 0.
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of 1 / (wall distance))
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

imode -- Output mode

- 0: B format with W output only (mono)

```
aout    =  aW
```

- 1: B format with W and Y output (stereo)

```
aleft  =  aW + 0.7071*aY
aright  =  aW - 0.7071*aY
```

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:

```
aWre, aWim      hilbert aW
aXre, aXim      hilbert aX
aYre, aYim      hilbert aY
aWXr    =  0.0928*aXre + 0.4699*aWre
aWXiYr  =  0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft   =  aWXr + aWXiYr
aright  =  aWXr - aWXiYr
```

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

```
aW      butterlp aW, ifreq      ; recommended values for ifreq
aY      butterlp aY, ifreq      ; are around 1000 Hz
aleft   =  aW + aX
aright  =  aY + aZ
```

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here: *http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm*

istor (optional, default=0) -- Skip initialization if non-zero (default: 0).

Performance

ain -- Input signal

aW, aX, aY, aZ -- Output signals

	mode 0	**mode 1**	**mode 2**	**mode 3**	**mode 4**
aW	W out	W out	W out	W out	left chn / low freq.
aX	0	0	X out	X out	left chn / high frq.

	mode 0	mode 1	mode 2	mode 3	mode 4
aY	0	Y out	Y out	Y out	right chn / low frq.
aZ	0	0	0	Z out	right chn / high fr.

If you encounter very slow performance (up to 100 times slower), it may be caused by denormals (this is also true of many other IIR opcodes, including *butterlp*, *pareq*, *hilbert*, and many others). Underflows can be avoided by:

- Using the *denorm* opcode on *ain* before *spat3di*.
- mixing low level DC or noise to the input signal, e.g.

  atmp rnd31 1/1e24, 0, 0

  aW, aX, aY, aZ spat3di ain + atmp, ...

  or

  aW, aX, aY, aZ spa3di ain + 1/1e24, ...

- reducing *irlen* in the case of *spat3dt* (which does not have an input signal). A value of about 0.005 is suitable for most uses, although it also depends on EQ settings. If the equalizer is not used, « irlen » can be set to 0.

Examples

See the examples for *spat3d*.

See Also

spat3d, *spat3dt*

Credits

Author: Istvan Varga
2001

New in version 4.12

Updated April 2002 by Istvan Varga

spat3dt

spat3dt — Can be used to render an impulse response for a 3D space at i-time.

Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. *spat3dt* can be used to render the impulse response at i-time, storing output in a function table, suitable for convolution.

Syntax

```
spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]
```

Initialization

ioutft -- Output ftable number for spat3dt. W, X, Y, and Z outputs are written interleaved to this table. If the table is too short, output will be truncated.

iX -- Sound source X coordinate in meters (positive: right, negative: left)

iY -- Sound source Y coordinate in meters (positive: front, negative: back)

iZ -- Sound source Z coordinate in meters (positive: up, negative: down)

idist -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

amplitude = 1 / (0.1 + distance)

delay = distance / 340 (in seconds)

Distance can be calculated as:

distance = sqrt(iX^2 + iY^2 + iZ^2)

In Mode 4, distance can be calculated as:

distance from left mic = sqrt((iX + idist/2)^2 + iY^2 + iZ^2)
distance from right mic = sqrt((iX - idist/2)^2 + iY^2 + iZ^2)

With *spat3d* the distance between the sound source and any microphone should be at least (340 * 18) / sr meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

ift -- Function table storing room parameters (for free field spatialization, set it to zero or negative). Table size is 54. The values in the table are:

Room Parameter	**Purpose**
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: N = (2*R + 2) * R. If all six walls are enabled: N = (((4*R + 6)*R + 8)*R) / 3
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if >= 0.
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of 1 / (wall distance))
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

imode -- Output mode

- 0: B format with W output only (mono)

```
aout   =  aW
```

- 1: B format with W and Y output (stereo)

```
aleft  =  aW + 0.7071*aY
aright  =  aW - 0.7071*aY
```

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:

```
aWre, aWim      hilbert aW
aXre, aXim      hilbert aX
aYre, aYim      hilbert aY
aWXr    =  0.0928*aXre + 0.4699*aWre
aWXiYr  =  0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft   =  aWXr + aWXiYr
aright  =  aWXr - aWXiYr
```

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

```
aW      butterlp aW, ifreq      ; recommended values for ifreq
aY      butterlp aY, ifreq      ; are around 1000 Hz
aleft  =  aW + aX
aright  =  aY + aZ
```

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here: *http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm*

irlen -- Impulse response length of echoes (in seconds). Depending on filter parameters, values around 0.005-0.01 are suitable for most uses (higher values result in more accurate output, but slower rendering)

iftnocl (optional, default=0) -- Do not clear output ftable (mix to existing data) if set to 1, clear table before writing if set to 0 (default: 0).

Examples

See the examples for *spat3d*.

See Also

spat3d, *spat3di*

Credits

Author: Istvan Varga
2001

New in version 4.12

Updated April 2002 by Istvan Varga

spdist

spdist — Calculates distance values from xy coordinates.

Description

spdist uses the same xy data as *space*, also either from a text file using *Gen28* or from x and y arguments given to the unit directly. The purpose of this unit is to make available the values for distance that are calculated from the xy coordinates.

In the case of *space*, the xy values are used to determine a distance which is used to attenuate the signal and prepare it for use in *spsend*. But it is also useful to have these values for distance available to scale the frequency of the signal before it is sent to the *space* unit.

Syntax

```
k1 spdist ifn, ktime, kx, ky
```

Initialization

ifn -- number of the stored function created using *Gen28*. This function generator reads a text file which contains sets of three values representing the xy coordinates and a time-tag for when the signal should be placed at that location. The file should look like:

```
0    -1    1
1     1    1
2     4    4
2.1  -4   -4
3    10  -10
5   -40    0
```

If that file were named "move" then the *Gen28* call in the score would like:

```
f1 0 0 28 "move"
```

Gen28 takes 0 as the size and automatically allocates memory. It creates values to 10 milliseconds of resolution. So in this case there will be 500 values created by interpolating X1 to X2 to X3 and so on, and Y1 to Y2 to Y3 and so on, over the appropriate number of values that are stored in the function table. In the above example, the sound will begin in the left front, over 1 second it will move to the right front, over another second it move further into the distance but still in the left front, then in just 1/10th of a second it moves to the left rear, a bit distant. Finally over the last .9 seconds the sound will move to the right rear, moderately distant, and it comes to rest between the two left channels (due west!), quite distant. Since the values in the table are accessed through the use of a time-pointer in the *space* unit, the actual timing can be made to follow the file's timing exactly or it can be made to go faster or slower through the same trajectory. If you have access to the GUI that allows one to draw and edit the files, there is no need to create the text files manually. But as long as the file is ASCII and in the format shown above, it doesn't matter how it is made!

IMPORTANT: If *ifn* is 0 then *space* will take its values for the xy coordinates from *kx* and *ky*.

Performance

The configuration of the xy coordinates in space places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of xy can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. x=0, y=1, will place the signal equally balanced between left and right front channels, x=y=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the xy's are kept so that Y>=1, it should work well to do panning and fixed localization in a stereo field.

ktime -- index into the table containing the xy coordinates. If used like:

```
ktime           line  0, 5, 5
a1, a2, a3, a4  space asig, 1, ktime, ...
```

with the file "move" described above, the speed of the signal's movement will be exactly as described in that file. However:

```
ktime           line  0, 10, 5
```

the signal will move at half the speed specified. Or in the case of:

```
ktime           line  5, 15, 0
```

the signal will move in the reverse direction as specified and 3 times slower! Finally:

```
ktime           line  2, 10, 3
```

will cause the signal to move only from the place specified in line 3 of the text file to the place specified in line 5 of the text file, and it will take 10 seconds to do it.

kx, ky -- when *ifn* is 0, *space* and *spdist* will use these values as the XY coordinates to localize the signal.

Examples

```
instr 1
  asig    ;some audio signal
  ktime              line  0, p3, p10
  a1, a2, a3, a4     space asig,1, ktime, .1
  ar1, ar2, ar3, ar4 spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                     outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument

  a1 reverb2 ga1, 2.5, .5
  a2 reverb2 ga2, 2.5, .5
  a3 reverb2 ga3, 2.5, .5
  a4 reverb2 ga4, 2.5, .5

     outq a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

space can useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using xy values from the score instead of a function table.

```
instr 1
  ...
  a1, a2, a3, a4     space asig, 0, 0, .1, p4, p5
  ar1, ar2, ar3, ar4 spsend

  ga1=ga1+ar1
  ga2=ga2+ar2
                     outs  a1, a2
endin

instr 99 ; reverb....
  ....
endin
```

A few notes: p4 and p5 are the X and Y values

```
  ;place the sound in the left speaker and near
    i1 0 1 -1 1
  ;place the sound in the right speaker and far
    i1 1 1 45 45
  ;place the sound equally between left and right and in the middle ground distance
    i1 2 1 0 12
e
```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```
ktime               line   0, p3, 10
kdist               spdist 1, ktime
kfreq = (ifreq * 340) / (340 + kdist)
asig                oscili iamp, kfreq, 1

a1, a2, a3, a4      space  asig, 1, ktime, .1
ar1, ar2, ar3, ar4 spsend
```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

See Also

space, *spsend*

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48

specaddm

specaddm — Perform a weighted add of two input spectra.

Description

Perform a weighted add of two input spectra.

Syntax

```
wsig specaddm wsig1, wsig2 [, imul2]
```

Initialization

imul2 (optional, default=0) -- if non-zero, scale the *wsig2* magnitudes before adding. The default value is 0.

Performance

wsig1 -- the first input spectra.

wsig2 -- the second input spectra.

Do a weighted add of two input spectra. For each channel of the two input spectra, the two magnitudes are combined and written to the output according to:

magout = mag1in + mag2in * imul2

The operation is performed whenever the input *wsig1* is sensed to be new. This unit will (at Initialization) verify the consistency of the two spectra (equal size, equal period, equal mag types).

Examples

```
wsig2    specdiff         wsig1               ; sense onsets
wsig3    specfilt         wsig2, 2            ; absorb slowly
         specdisp         wsig2, .1           ; & display both spectra
         specdisp         wsig3, .1
```

See Also

specdiff, *specfilt*, *spechist*, *specscal*

specdiff

specdiff — Finds the positive difference values between consecutive spectral frames.

Description

Finds the positive difference values between consecutive spectral frames.

Syntax

```
wsig specdiff wsigin
```

Performance

wsig -- the output spectrum.

wsigin -- the input spectra.

Finds the positive difference values between consecutive spectral frames. At each new frame of *wsigin*, each magnitude value is compared with its predecessor, and the positive changes written to the output spectrum. This unit is useful as an energy onset detector.

Examples

```
wsig2    specdiff         wsig1                 ; sense onsets
wsig3    specfilt         wsig2, 2              ; absorb slowly
         specdisp         wsig2, .1             ; & display both spectra
         specdisp         wsig3, .1
```

See Also

specaddm, *specfilt*, *spechist*, *specscal*

specdisp

specdisp — Displays the magnitude values of the spectrum.

Description

Displays the magnitude values of the spectrum.

Syntax

```
specdisp wsig, iprd [, iwtflg]
```

Initialization

iprd -- the period, in seconds, of each new display.

iwtflg (optional, default=0) -- wait flag. If non-zero, hold each display until released by the user. The default value is 0 (no wait).

Performance

wsig -- the input spectrum.

Displays the magnitude values of spectrum *wsig* every *iprd* seconds (rounded to some integral number of *wsig*'s originating *iprd*).

Examples

```
  ksum     specsum   wsig,  1                     ; sum the spec bins, and ksmooth
           if        ksum < 2000   kgoto  zero    ; if sufficient amplitude
  koct     specptrk  wsig                         ;    pitch-track the signal
           kgoto      contin
zero:
  koct    =     0                                 ; else output zero
contin:
```

See Also

specsum

specfilt

specfilt — Filters each channel of an input spectrum.

Description

Filters each channel of an input spectrum.

Syntax

```
wsig specfilt wsigin, ifhtim
```

Initialization

ifhtim -- half-time constant.

Performance

wsigin -- the input spectrum.

Filters each channel of an input spectrum. At each new frame of *wsigin*, each magnitude value is injected into a 1st-order lowpass recursive filter, whose half-time constant has been initially set by sampling the ftable *ifhtim* across the (logarithmic) frequency space of the input spectrum. This unit effectively applies a *persistence* factor to the data occurring in each spectral channel, and is useful for simulating the *energy integration* that occurs during auditory perception. It may also be used as a time-attenuated running *histogram* of the spectral distribution.

Examples

```
wsig2    specdiff         wsig1                 ; sense onsets
wsig3    specfilt         wsig2, 2              ; absorb slowly
         specdisp         wsig2, .1             ; & display both spectra
         specdisp         wsig3, .1
```

See Also

specaddm, *specdiff*, *spechist*, *specscal*

spechist

spechist — Accumulates the values of successive spectral frames.

Description

Accumulates the values of successive spectral frames.

Syntax

```
wsig spechist wsigin
```

Performance

wsigin -- the input spectra.

Accumulates the values of successive spectral frames. At each new frame of *wsigin*, the accumulations-to-date in each magnitude track are written to the output spectrum. This unit thus provides a running *histogram* of spectral distribution.

Examples

```
wsig2   specdiff        wsig1               ; sense onsets
wsig3   specfilt        wsig2, 2            ; absorb slowly
        specdisp        wsig2, .1           ; & display both spectra
        specdisp        wsig3, .1
```

See Also

specaddm, *specdiff*, *specfilt*, *specscal*

specptrk

spectrk — Estimates the pitch of the most prominent complex tone in the spectrum.

Description

Estimate the pitch of the most prominent complex tone in the spectrum.

Syntax

```
koct, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \
      irolloff [, iodd] [, iconfs] [, interp] [, ifprd] [, iwtflg]
```

Initialization

ilo, ihi, istr -- pitch range conditioners (low, high, and starting) expressed in decimal octave form.

idbthresh -- energy threshold (in decibels) for pitch tracking to occur. Once begun, tracking will be continuous until the energy falls below one half the threshold (6 dB down), whence the *koct* and *kamp* outputs will be zero until the full threshold is again surpassed. *idbthresh* is a guiding value. At initialization it is first converted to the *idbout* mode of the source spectrum (and the 6 dB down point becomes .5, .25, or 1/root 2 for modes 0, 2 and 3). The values are also further scaled to allow for the weighted partial summation used during correlation.The actual thresholding is done using the internal weighted and summed *kamp* value that is visible as the second output parameter.

inptls, irolloff -- number of harmonic partials used as a matching template in the spectrally-based pitch detection, and an amplitude rolloff for the set expressed as some fraction per octave (linear, so don't roll off to negative). Since the partials and rolloff fraction can affect the pitch following, some experimentation will be useful: try 4 or 5 partials with .6 rolloff as an initial setting; raise to 10 or 12 partials with rolloff .75 for complex timbres like the bassoon (weak fundamental). Computation time is dependent on the number of partials sought. The maximum number is 16.

iodd (optional) -- if non-zero, employ only odd partials in the above set (e.g. *inptls* of 4 would employ partials 1,3,5,7). This improves the tracking of some instruments like the clarinet The default value is 0 (employ all partials).

iconfs (optional) -- number of confirmations required for the pitch tracker to jump an octave, pro-rated for fractions of an octave (i.e. the value 12 implies a semitone change needs 1 confirmation (two hits) at the *spectrum* generating *iprd*). This parameter limits spurious pitch analyses such as octave errors. A value of 0 means no confirmations required; the default value is 10.

interp (optional) -- if non-zero, interpolate each output signal (*koct*, *kamp*) between incoming *wsig* frames. The default value is 0 (repeat the signal values between frames).

ifprd (optional) -- if non-zero, display the internally computed spectrum of candidate fundamentals. The default value is 0 (no display).

iwtftg (optional) -- wait flag. If non-zero, hold each display until released by the user. The default value is 0 (no wait).

Performance

At note initialization this unit creates a template of *inptls* harmonically related partials (odd partials, if

iodd non-zero) with amplitude rolloff to the fraction *irolloff* per octave. At each new frame of *wsig*, the spectrum is cross-correlated with this template to provide an internal spectrum of candidate fundamentals (optionally displayed). A likely pitch/amp pair (*koct*, *kamp*, in decimal octave and summed *idbout* form) is then estimated. *koct* varies from the previous *koct* by no more than plus or minus *kvar* decimal octave units. It is also guaranteed to lie within the hard limit range *ilo* -- *ihi* (decimal octave low and high pitch). *kvar* can be dynamic, e.g. onset amp dependent. Pitch resolution uses the originating *spectrum ifrqs* bins/octave, with further parabolic interpolation between adjacent bins. Settings of root magnitude, *ifrqs* = 24, *iq* = 15 should capture all the inflections of interest. Between frames, the output is either repeated or interpolated at the k-rate. (See *spectrum*.)

Examples

```
a1,a2   ins                                                              ; read a stereo clarinet input
krms    rms        a1, 20                                                ; find a monaural rms value
kvar    =          0.6 + krms/8000                                       ; & use to gate the pitch varianc
wsig    spectrum   a1, .01, 7, 24, 15, 0, 3                              ; get a 7-oct spectrum, 24 bibs/o
        specdisp   wsig, .2                                              ; display this and now estimate
koct,ka spectrk    wsig, kvar, 7.0, 10, 9, 20,  4, .7, 1, 5, 1, .2       ; the pch and amp
aosc    oscil      ka*ka*10, cpsoct(koct),2                              ; & generate \ new tone with thes
koct    =          (koct<7.0?7.0:koct)                                 ; replace non pitch with low C
        display    koct-7.0, .25, 20                                     ; & display the pitch track
        display    ka, .25, 20                                           ; plus the summed root mag
        outs       a1, aosc                                              ; output 1 original and 1 new tra
```

specscal

specscal — Scales an input spectral datablock with spectral envelopes.

Description

Scales an input spectral datablock with spectral envelopes.

Syntax

```
wsig specscal wsigin, ifscale, ifthresh
```

Initialization

ifscale -- scale function table. A function table containing values by which a value's magnitude is rescaled.

ifthresh -- threshold function table. If *ifthresh* is non-zero, each magnitude is reduced by its corresponding table-value (to not less than zero)

Performance

wsig -- the output spectrum

wsigin -- the input spectra

Scales an input spectral datablock with spectral envelopes. Function tables *ifthresh* and *ifscale* are initially sampled across the (logarithmic) frequency space of the input spectrum; then each time a new input spectrum is sensed the sampled values are used to scale each of its magnitude channels as follows: if *ifthresh* is non-zero, each magnitude is reduced by its corresponding table-value (to not less than zero); then each magnitude is rescaled by the corresponding *ifscale* value, and the resulting spectrum written to *wsig*.

Examples

```
wsig2    specdiff         wsig1                ; sense onsets
wsig3    specfilt         wsig2, 2             ; absorb slowly
         specdisp         wsig2, .1            ; & display both spectra
         specdisp         wsig3, .1
```

See Also

specaddm, *specdiff*, *specfilt*, *spechist*

specsum

specsum — Sums the magnitudes across all channels of the spectrum.

Description

Sums the magnitudes across all channels of the spectrum.

Syntax

```
ksum specsum wsig [, interp]
```

Initialization

interp (optional, default-0) -- if non-zero, interpolate the output signal (*koct* or *ksum*). The default value is 0 (repeat the signal value between changes).

Performance

ksum -- the output signal.

wsig -- the input spectrum.

Sums the magnitudes across all channels of the spectrum. At each new frame of *wsig*, the magnitudes are summed and released as a scalar *ksum* signal. Between frames, the output is either repeated or interpolated at the k-rate. This unit produces a k-signal summation of the magnitudes present in the spectral data, and is thereby a running measure of its moment-to-moment overall strength.

Examples

```
  ksum     specsum   wsig,  1                        ; sum the spec bins, and ksmooth
           if        ksum < 2000   kgoto  zero   ; if sufficient amplitude
  koct     specptrk  wsig                            ;    pitch-track the signal
           kgoto      contin
zero:
  koct    =     0                                    ; else output zero
contin:
```

See Also

specdisp

spectrum

spectrum — Generate a constant-Q, exponentially-spaced DFT.

Description

Generate a constant-Q, exponentially-spaced DFT across all octaves of a multiply-downsampled control or audio input signal.

Syntax

```
wsig spectrum xsig, iprd, iocts, ifrqa [, iq] [, ihann] [, idbout] \
      [, idsprd] [, idsinrs]
```

Initialization

ihann (optional) -- apply a Hamming or Hanning window to the input. The default is 0 (Hamming window)

idbout (optional) -- coded conversion of the DFT output:

- 0 = magnitude
- 1 = dB
- 2 = mag squared
- 3 = root magnitude

The default value is 0 (magnitude).

idisprd (optional) -- if non-zero, display the composite downsampling buffer every *idisprd* seconds. The default value is 0 (no display).

idsines (optional) -- if non-zero, display the Hamming or Hanning windowed sinusoids used in DFT filtering. The default value is 0 (no sinusoid display).

Performance

This unit first puts signal *asig* or *ksig* through *iocts* of successive octave decimation and downsampling, and preserves a buffer of down-sampled values in each octave (optionally displayed as a composite buffer every *idisprd* seconds). Then at every *iprd* seconds, the preserved samples are passed through a filter bank (*ifrqs* parallel filters per octave, exponentially spaced, with frequency/bandwidth Q of *iq*), and the output magnitudes optionally converted (*idbout*) to produce a band-limited spectrum that can be read by other units.

The stages in this process are computationally intensive, and computation time varies directly with *iocts*, *ifrqs*, *iq*, and inversely with *iprd*. Settings of *ifrqs* = 12, *iq* = 10, *idbout* = 3, and *iprd* = .02 will normally be adequate, but experimentation is encouraged. *ifrqs* currently has a maximum of 120 divisions per octave. For audio input, the frequency bins are tuned to coincide with A440.

This unit produces a self-defining spectral datablock *wsig*, whose characteristics used (*iprd*, *iocts*, *ifrqs*, *idbout*) are passed via the data block itself to all derivative *wsigs*. There can be any number of spectrum

units in an instrument or orchestra, but all *wsig* names must be unique.

Examples

```
asig in                              ; get external audio
wsig spectrum  asig,.02,6,12,33,0,1,1  ; downsample in 6 octs & calc a 72 pt dft (Q 33, dB out) every 2
```

splitrig

splitrig — Split a trigger signal

Description

splitrig splits a trigger signal (i.e. a timed sequence of control-rate impulses) into several channels following a structure designed by the user.

Syntax

```
splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]
```

Initialization

imaxtics - number of tics belonging to largest pattern

ifn - number of table containing channel-data structuring

Performance

asig - incoming (input) signal

ktrig - trigger signal

The *splitrig* opcode splits a trigger signal into several output channels according to one or more patterns provided by the user. Normally the regular timed trigger signal generated by metro opcode is used to be transformed into rhythmic pattern that can trig several independent melodies or percussion riffs. But you can also start from non-isocronous trigger signals. This allows to use some "interpretative" and less "mechanic" groove variations. Patterns are looped and each numtics_of_pattern_N the cicle is repeated.

The scheme of patterns is defined by the user and is stored into ifn table according to the following format:

```
gi1  ftgen 1,0,1024,  -2 \  ; table is generated with GEN02 in this case
\                           ;
numtics_of_pattern_1, \ ;pattern 1
   tic1_out1, tic1_out2, ... , tic1_outN,\
   tic2_out1, tic2_out2, ... , tic2_outN,\
   tic3_out1, tic3_out2, ... , tic3_outN,\
   .....
   ticN_out1, ticN_out2, ... , ticN_outN,\
\
numtics_of_pattern_2, \ ;pattern 2
   tic1_out1, tic1_out2, ... , tic1_outN,\
   tic2_out1, tic2_out2, ... , tic2_outN,\
   tic3_out1, tic3_out2, ... , tic3_outN,\
   .....
   ticN_out1, ticN_out2, ... , ticN_outN,\
   .....
\
numtics_of_pattern_N,\ ;pattern N
   tic1_out1, tic1_out2, ... , tic1_outN,\
   tic2_out1, tic2_out2, ... , tic2_outN,\
   tic3_out1, tic3_out2, ... , tic3_outN,\
   .....
   ticN_out1, ticN_out2, ... , ticN_outN,\
```

This scheme can contain more than one pattern, each one with a different number of rows. Each pattern is preceded by a a special row containing a single *numtics_of_pattern_N* field; this field expresses the number of tics that makes up the corresponding pattern. Each pattern's row makes up a tic. Each pattern's column corresponds to a cannel, and each field of a row is a number that makes up the value outputted by the corresponding *koutXX* channel (if number is a zero, corresponding output channel will not trigger anything in that particular arguments). Obviously, all rows must contain the same number of fields that must be equal to the number of *koutXX* channel. All patterns must contain the same number of rows, this number must be equal to the largest pattern and is defined by *imaxtics* variable. Even if a pattern has less tics than the largest pattern, it must be made up of the same number of rows, in this case, some of these rows, at the end of the pattern itself, will not be used (and can be set to any value, because it doesn't matter).

The *kndx* variable chooses the number of the pattern to be played, zero indicating the first pattern. Each time the integer part of *kndx* changes, tic counter is reset to zero.

Patterns are looped and each *numtics_of_pattern_N* the cicle is repeated.

examples 4 - calculate average value of asig in the time interval

This opcode can be useful in several situations, for example to implement a vu-meter

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

spsend

spsend — Generates output signals based on a previously defined *space* opcode.

Description

spsend depends upon the existence of a previously defined *space*. The output signals from *spsend* are derived from the values given for xy and reverb in the *space* and are ready to be sent to local or global reverb units (see example below).

Syntax

```
a1, a2, a3, a4 spsend
```

Performance

The configuration of the xy coordinates in space places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of xy can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. x=0, y=1, will place the signal equally balanced between left and right front channels, x=y=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the xy's are kept so that Y>=1, it should work well to do panning and fixed localization in a stereo field.

Examples

```
instr 1
  asig    ;some audio signal
  ktime              line  0, p3, p10
  a1, a2, a3, a4     space asig,1, ktime, .1
  ar1, ar2, ar3, ar4 spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                     outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument
```

```
  a1 reverb2 ga1, 2.5, .5
  a2 reverb2 ga2, 2.5, .5
  a3 reverb2 ga3, 2.5, .5
  a4 reverb2 ga4, 2.5, .5

     outq a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

space can useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using xy values from the score instead of a function table.

```
instr 1
  ...
  a1, a2, a3, a4     space asig, 0, 0, .1, p4, p5
  ar1, ar2, ar3, ar4 spsend

  ga1=ga1+ar1
  ga2=ga2+ar2
                     outs  a1, a2
endin

instr 99 ; reverb....
  ....
endin
```

A few notes: p4 and p5 are the X and Y values

```
  ;place the sound in the left speaker and near
    i1 0 1 -1 1
  ;place the sound in the right speaker and far
    i1 1 1 45 45
  ;place the sound equally between left and right and in the middle ground distance
    i1 2 1 0 12
e
```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```
  ktime              line   0, p3, 10
  kdist              spdist 1, ktime
  kfreq = (ifreq * 340) / (340 + kdist)
  asig               oscili iamp, kfreq, 1

  a1, a2, a3, a4     space  asig, 1, ktime, .1
  ar1, ar2, ar3, ar4 spsend
```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

See Also

space, *spdist*

Credits

Author: Richard Karpen
Seattle, WA USA
1998

New in Csound version 3.48

sprintf

sprintf — printf-style formatted output to a string variable.

Description

sprintf write printf-style formatted output to a string variable, similarly to the C function sprintf(). sprintf runs at i-time only.

Syntax

```
Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]
```

```
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

Initialization

Sfmt -- format string, has the same format as in printf() and other similar C functions, except length modifiers (l, ll, h, etc.) are not supported. The following conversion specifiers are allowed:

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

xarg1, xarg2, ... -- input arguments (max. 30) for format, should be i-rate for all conversion specifiers except %s, which requires a string argument. Integer formats like %d round the input values to the nearest integer.

Performance

Sdst -- output string variable

Example

```
Sname   sprintf "soundin-%04d.wav", ifileno
Smsg    sprintf "The file name is: '%s'", Sname
        puts Smsg, 1
asig soundin Sname
```

See also

sprintfk

Credits

Author: Istvan Varga
2005

sprintfk

sprintfk — printf-style formatted output to a string variable at k-rate.

Description

sprintfk writes printf-style formatted output to a string variable, similarly to the C function sprintf(). *sprintfk* runs both at initialization and performance time.

Syntax

```
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

Initialization

Sfmt -- format string, has the same format as in printf() and other similar C functions, except length modifiers (l, ll, h, etc.) are not supported. The following conversion specifiers are allowed:

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

xarg1, xarg2, ... -- input arguments (max. 30) for format, should be i-rate for all conversion specifiers except %s, which requires a string argument. sprintfk also allows k-rate number arguments, but these should still be valid at init time as well (unless sprintfk is skipped with igoto). Integer formats like %d round the input values to the nearest integer.

Performance

Sdst -- output string variable

Examples

Here is an example of the sprintfk opcode. It uses the file *sprintfk.csd* [examples/sprintfk.csd].

Exemple 480. Example of the sprintfk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sprintfk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

  sr     =  48000
  ksmps     =  16
  nchnls    =  2
  0dbfs     =  1
```

```
; Example by Jonathan Murphy 2007

    instr 1

  S1      =   "1"
  S2      =   " + 1"
  ktrig      init       0
  kval     init       2
if (ktrig == 1) then
  S1     strcatk    S1, S2
  kval      =  kval + 1
endif
  String     sprintfk   "%s = %d", S1, kval
             puts       String, kval
  ktrig      metro      1

    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

See also

sprintf, *puts*, *strcatk*

Credits

Author: Istvan Varga
2005
Example by Jonathan Murphy

sqrt

sqrt — Retourne une racine carrée.

Description

Retourne la racine carrée de *x* (*x* non-negatif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

Syntaxe

```
sqrt(x) (pas de restriction de taux)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

Exemples

Voici un exemple de l'opcode sqrt. Il utilise le fichier *sqrt.csd* [examples/sqrt.csd].

Exemple 481. Exemple de l'opcode sqrt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sqrt.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = sqrt(64)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 8.000
```

Voir Aussi

abs, *exp*, *frac*, *int*, *log*, *log10*, *i*

Crédits

Exemple écrit par Kevin Conder.

sr

sr — Fixe la taux d'échantillonnage audio.

Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

Syntaxe

```
sr = iarg
```

Initialisation

sr = (facultatif) -- fixe le taux d'échantillonnage à *iarg* échantillons par seconde par canal. La valeur par défaut est 44100.

De plus, toute *variable globale* [54] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *sr*. Le taux d'échantillonnage sera calculé à partir de *kr* et de *ksmps*, mais le résultat doit être une valeur entière. Si aucune de ces valeurs globales n'est définie, le taux d'échantillonnage par défaut sera 44100. Habituellement, vous utiliserez une valeur supportée par votre carte son, comme 44100 ou 48000, sinon, le résultat audio généré par csound risque d'être injouable, ou bien vous aurez une erreur si vous essayez une exécution en temps-réel. Vous pouvez naturellement utiliser un taux d'échantillonnage comme 96000, pour un rendu différé, même si votre carte son ne le supporte pas. Csound générera un fichier valide jouable sur des systèmes offrant cette possibilité.

Exemples

```
sr = 10000
kr = 500
ksmps = 20
gi1 = sr/2.
ga init 0
itranspose = octpch(.01)
```

Voir Aussi

kr, *ksmps*, *nchnls*

stack

stack — Initializes the stack.

Description

Initializes and sets the size of the global stack.

Syntax

```
stack  iStackSize
```

Initialization

iStackSize - size of the stack in bytes.

Performance

Csound implements a single global stack. Initializing the stack with the *stack* opcode is not required - it is optional, and if not done, the first use of *push* or *push_f* will automatically create a stack of 32768 bytes. Otherwise, *stack* is normally called from the orchestra header, and takes a stack size parameter in bytes (there is an upper limit of about 16 MB). Once set, the stack size is fixed and cannot be changed during performance.

The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

push and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push_f* only pushes a "reference" to the f-signal, and then the corresponding *pop_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push_f* before *pop_f* is called is not recommended, and if the instrument instance owning the variable that was passed by push_f is deactivated before pop_f is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

See also

pop, *push*, *pop_f* and *push_f*.

Credits

By: Istvan Varga.

2006

statevar

statevar — State-variable filter.

Description

Statevar is a new digital implementation of the analogue state-variable filter. This filter has four simultaneous outputs: high-pass, low-pass, band-pass and band-reject. This filter uses oversampling for sharper resonance (default: 3 times oversampling). It includes a resonance limiter that prevents the filter from getting unstable.

Syntax

```
ahp,alp,abp,abr statevar ain, kcf, kq [, iosamps, istor]
```

Initialization

iosamps -- number of times of oversampling used in the filtering process. This will determine the maximum sharpness of the filter resonance (Q). More oversampling allows higher Qs, less oversampling will limit the resonance. The default is 3 times (iosamps=0).

istor --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

Performance

ahp -- high-pass output signal.

alp -- low-pass output signal.

abp -- band-pass signal.

abr -- band-reject signal.

asig -- input signal.

kcf -- filter cutoff frequency

kq -- filter Q. This value is limited internally depending on the frequency and the number of times of oversampling used in the process (3-times oversampling by default).

Examples

Exemple 482. Example

```
kenv              linseg 0,0.1,1, p3-0.2,1, 0.1, 0
asig              buzz 16000*kenv, 100, 100, 1;
kf                expseg 100, p3/2, 5000, p3/2, 1000
ahp,alp,abp,abr   statevar  asig, kf, 200
```

```
outs alp,ahp
```

Credits

Author: Victor Lazzarini;
January 2005

New plugin in version 5

January 2005.

stix

stix — Modèle semi-physique d'un son de baguette.

Description

stix est un modèle semi-physique d'un son de baguette. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]
```

Initialisation

iamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 30.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,998 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,998 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

Exemples

Voici un exemple de l'opcode stix. Il utilise le fichier *stix.csd* [examples/stix.csd].

Exemple 483. Exemple de l'opcode stix.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o stix.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra ---------------

  sr =            44100
  kr =             4410
  ksmps =               10
  nchnls =                1

instr 01                  ;an example of stix
  a1      line 20, p3, 20                         ;preset amplitude increase
  a2      stix p4, 0.01   ;stix needs a little amp help at these settings
  a3      product a1, a2                             ;increase amplitude
          out a3
          endin


</CsInstruments>
<CsScore>

;score -------------------

   i1 0 1 26000
   e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

cabasa, *crunch*, *sandpaper*, *sekere*

Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

strchar

strchar — Return the ASCII code of a character in a string

Description

Return the ASCII code of the character in Sstr at ipos (defaults to zero which means the first character), or zero if ipos is out of range. strchar runs at init time only.

Syntax

```
ichr strchar Sstr[, ipos]
```

See also

strchark

Credits

Author: Istvan Varga
2006

strchark

strchark — Return the ASCII code of a character in a string

Description

Return the ASCII code of the character in Sstr at kpos (defaults to zero which means the first character), or zero if kpos is out of range. strchark runs both at init and performance time.

Syntax

```
kchr strchark Sstr[, kpos]
```

See also

strchar

Credits

Author: Istvan Varga
2006

New in version 5.02

strcpy

strcpy — Assign value to a string variable

Description

Assign to a string variable by copying the source which may be a constant or another string variable. strcpy and = copy the string at i-time only.

Syntax

```
Sdst strcpy Ssrc
```

```
Sdst = Ssrc
```

Example

```
Sfoo    strcpy "Hello, world !"
        puts Sfoo, 1
```

See also

strcpyk

Credits

Author: Istvan Varga
2005

strcpyk

strcpyk — Assign value to a string variable (k-rate)

Description

Assign to a string variable by copying the source which may be a constant or another string variable. *strcpyk* does the assignment both at initialization and performance time.

Syntax

```
Sdst strcpyk Ssrc
```

See also

strcpy

Credits

Author: Istvan Varga
2005

strcat

strcat — Concatenate strings

Description

Concatenate two strings and store the result in a variable. *strcat* runs at i-time only. It is allowed for any of the input arguments to be the same as the output variable.

Syntax

```
Sdst strcat Ssrc1, Ssrc2
```

Example

```
Sname   =  "beats"
Sname   strcat Sname, ".wav"
asig    soundin Sname
```

See also

strcatk

Credits

Author: Istvan Varga
2005

New in version 5.02

strcatk

strcatk — Concatenate strings (k-rate)

Description

Concatenate two strings and store the result in a variable. *strcatk* does the concatenation both at initialization and performance time. It is allowed for any of the input arguments to be the same as the output variable.

Syntax

```
Sdst strcatk Ssrc1, Ssrc2
```

See also

strcat

Credits

Author: Istvan Varga
2005

New in version 5.02

strcmp

strcmp — Compare strings

Description

Compare strings and set the result to -1, 0, or 1 if the first string is less than, equal to, or greater than the second, respectively. strcmp compares at i-time only.

Syntax

```
ires strcmp S1, S2
```

See also

strcmpk

Credits

Author: Istvan Varga
2005

strcmpk

strcmp — Compare strings

Description

Compare strings and set the result to -1, 0, or 1 if the first string is less than, equal to, or greater than the second, respectively. *strcmpk* does the comparison both at initialization and performance time.

Syntax

```
kres strcmpk S1, S2
```

See also

strcmp

Credits

Author: Istvan Varga
2005

streson

streson — A string resonator with variable fundamental frequency.

Description

An audio signal is modified by a string resonator with variable fundamental frequency.

Syntax

```
ares streson asig, kfr, ifdbgain
```

Initialization

ifdbgain -- feedback gain, between 0 and 1, of the internal delay line. A value close to 1 creates a slower decay and a more pronounced resonance. Small values may leave the input signal unaffected. Depending on the filter frequency, typical values are > .9.

Performance

asig -- the input audio signal.

kfr -- the fundamental frequency of the string.

streson passes the input *asig* through a network composed of comb, low-pass and all-pass filters, similar to the one used in some versions of the Karplus-Strong algorithm, creating a string resonator effect. The fundamental frequency of the « string » is controlled by the k-rate variable *kfr*.This opcode can be used to simulate sympathetic resonances to an input signal.

See *Modal Frequency Ratios* for frequency ratios of real intruments which can be used to determine the values of *kfrq*.

streson is an adaptation of the StringFlt object of the SndObj Sound Object Library developed by the author.

Examples

Here is an example of the streson opcode. It uses the file *streson.csd* [examples/streson.csd].

Exemple 484. Example of the streson opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o streson.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a normal sine wave.
  asig oscils 8000, 440, 1

  ; Vary the fundamental frequency of the string
  ; resonator linearly from 220 to 880 Hertz.
  kfr line 220, p3, 880
  ifdbgain = 0.95

  ; Run our sine wave through the string resonator.
  astres streson asig, kfr, ifdbgain

  ; The resonance can get quite loud.
  ; So we'll clip the signal at 30,000.
  a1 clip astres, 1, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for five seconds.
i 1 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Victor Lazzarini
Music Department
National University of Ireland, Maynooth
Maynooth, Co. Kildare
1998

Example written by Kevin Conder.

New in Csound version 3.494

strget

strget — Set string variable to value from strset table or string p-field

Description

strget sets a string variable at initialization time to the value stored in *strset* table at the specified index, or a string p-field from the score. If there is no string defined for the index, the variable is set to an empty string.

Syntax

```
Sdst strget indx
```

Initialization

indx -- strset index, or score p-field

Sdst -- destination string variable

See also

strset

Credits

Author: Istvan Varga

2005

strindex

strindex — Return the position of the first occurence of a string in another string

Description

Return the position of the first occurence of S2 in S1, or -1 if not found. If S2 is empty, 0 is returned. strindex runs at init time only.

Syntax

```
ipos strindex S1, S2
```

See also

strindexk

Credits

Author: Istvan Varga
2006

New in version 5.02

strindexk

strindexk — Return the position of the first occurence of a string in another string

Description

Return the position of the first occurence of S2 in S1, or -1 if not found. If S2 is empty, 0 is returned. strindexk runs both at init and performance time.

Syntax

```
kpos strindexk S1, S2
```

See also

strindex

Credits

Author: Istvan Varga
2006

New in version 5.02

strlen

strlen — Return the length of a string

Description

Return the length of a string, or zero if it is empty. strlen runs at init time only.

Syntax

```
ilen strlen Sstr
```

See also

strlenk

Credits

Author: Istvan Varga
2006

New in version 5.02

strlenk

strlenk — Return the length of a string

Description

Return the length of a string, or zero if it is empty. strlenk runs both at init and performance time.

Syntax

```
klen strlenk Sstr
```

See also

strlen

Credits

Author: Istvan Varga
2006

New in version 5.02

strlower

strlower — Convert a string to lower case

Description

Convert Ssrc to lower case, and write the result to Sdst. strlower runs at init time only.

Syntax

```
Sdst strlower Ssrc
```

See also

strlowerk

Credits

Author: Istvan Varga
2006

New in version 5.02

strlowerk

strlowerk — Convert a string to lower case

Description

Convert Ssrc to lower case, and write the result to Sdst. strlowerk runs both at init and performance time.

Syntax

```
Sdst strlowerk Ssrc
```

See also

strlower

Credits

Author: Istvan Varga
2006

New in version 5.02

strrindex

strrindex — Return the position of the last occurence of a string in another string

Description

Return the position of the last occurence of S2 in S1, or -1 if not found. If S2 is empty, the length of S1 is returned. strrindex runs at init time only.

Syntax

```
ipos strrindex S1, S2
```

See also

strrindexk

Credits

Author: Istvan Varga
2006

New in version 5.02

strrindexk

strrindexk — Return the position of the last occurence of a string in another string

Description

Return the position of the last occurence of S2 in S1, or -1 if not found. If S2 is empty, the length of S1 is returned. strrindexk runs both at init and performance time.

Syntax

```
kpos strrindexk S1, S2
```

See also

strrindex

Credits

Author: Istvan Varga
2006

New in version 5.02

strset

strset — Allows a string to be linked with a numeric value.

Description

Allows a string to be linked with a numeric value.

Syntax

```
strset iarg, istring
```

Initialization

iarg -- the numeric value.

istring -- the alphanumeric string (in double-quotes).

strset (optional) allows a string, such as a filename, to be linked with a numeric value. Its use is optional.

Examples

The following statement, used in the orchestra header, will allow the numeric value 10 to be substituted anywhere the soundfile *asound.wav* is called for.

```
strset 10, "asound.wav"
```

Examples

Here is an example of the strset opcode. It uses the file *strset.csd* [examples/strset.csd].

Exemple 485. Example of the strset opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 1

;Example by Andres Cabrera 2008
```

```
; \\n is used to denote "new line"
strset 1, "String 1\\n"
strset 2, "String 2\\n"

instr 1
Str strget p4
prints Str
endin


</CsInstruments>
<CsScore>
;         p4 is used to select string
i 1 0 1  1
i 1 3 1  2
</CsScore>
</CsoundSynthesizer>
```

See Also

pset and *strget*

strsub

strsub — Extract a substring

Description

Return a substring of the source string. strsub runs at init time only.

Syntax

```
Sdst strsub Ssrc[, istart[, iend]]
```

Initialization

istart (optional, defaults to 0) -- start position in Ssrc, counting from 0. A negative value means the end of the string.

iend (optional, defaults to -1) -- end position in Ssrc, counting from 0. A negative value means the end of the string. If iend is less than istart, the output is reversed.

Examples

Here is an example of the strsub opcode. It uses the file *strsub.csd* [examples/strsub.csd].

Exemple 486. Example of the strsub opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     ;;;-d      RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o strsub.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

instr 1
    Smember strget p4

    ; Parse Smember
    istrlen     strlen    Smember
    idelimiter strindex Smember, ":"

    S1    strsub Smember, 0, idelimiter  ; "String1"
    S2    strsub Smember, idelimiter + 1, istrlen  ; "String2"

    printf "First string: %s\nSecond string: %s\n", 1, S1, S2

endin

</CsInstruments>
<CsScore>
i 1 0 1 "String1:String2"
</CsScore>
</CsoundSynthesizer>
```

See also

strsubk

Credits

Author: Istvan Varga
2006

strsubk

strsubk — Extract a substring

Description

Return a substring of the source string. strsubk runs both at init and performance time.

Syntax

```
Sdst strsubk Ssrc, kstart, kend
```

Performance

kstart -- start position in Ssrc, counting from 0. A negative value means the end of the string.

kend -- end position in Ssrc, counting from 0. A negative value means the end of the string. If kend is less than kstart, the output is reversed.

See also

strsub

Credits

Author: Istvan Varga
2006

strtod

strtod — Converts a string to a float (i-rate).

Description

Convert a string to a floating point value. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

Syntax

```
ir strtod Sstr
```

```
ir strtod indx
```

Initialization

Sstr -- String to convert.

indx -- index of string set by strset

Performance

ir -- Value of string as float.

Credits

Author: Istvan Varga
2005

strtodk

strtodk — Converts a string to a float (k-rate).

Description

Convert a string to a floating point value at i- or k-rate. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

Note

If a k-rate index variable is used, it should be valid at i-time as well.

Syntax

```
kr strtodk Sstr
```

```
kr strtodk kndx
```

Performance

kr -- Value of string as float.

Sstr -- String to convert.

indx -- index of string set by strset

Credits

Author: Istvan Varga
2005

strtol

strtol — Converts a string to a signed integer (i-rate).

Description

Convert a string to a signed integer value. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

Syntax

```
ir strtol Sstr
```

```
ir strtol indx
```

Initialization

Sstr -- String to convert.

indx -- index of string set by strset

strtol can parse numbers in decimal, octal (prefixed by 0), and hexadecimal (with a prefix of 0x) format.

Performance

ir -- Value of string as signed integer.

Credits

Author: Istvan Varga
2005

strtolk

strtolk — Converts a string to a signed integer (k-rate).

Description

Convert a string to a floating point value at i- or k-rate. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

Note

If a k-rate index variable is used, it should be valid at i-time as well.

Syntax

```
kr strtolk Sstr
```

```
kr strtolk kndx
```

strtolk can parse numbers in decimal, octal (prefixed by 0), and hexadecimal (with a prefix of 0x) format.

Performance

kr -- Value of string as signed integer.

Sstr -- String to convert.

indx -- index of string set by strset

Credits

Author: Istvan Varga
2005

strupper

strupper — Convert a string to upper case

Description

Convert Ssrc to upper case, and write the result to Sdst. strupper runs at init time only.

Syntax

```
Sdst strupper Ssrc
```

See also

strupperk

Credits

Author: Istvan Varga
2006

New in version 5.02

strupperk

strupperk — Convert a string to upper case

Description

Convert Ssrc to upper case, and write the result to Sdst. strupperk runs both at init and performance time.

Syntax

```
Sdst strupperk Ssrc
```

See also

strupper

Credits

Author: Istvan Varga
2006

New in version 5.02

subinstr

subinstr — Creates and runs a numbered instrument instance.

Description

Creates an instance of another instrument and is used as if it were an opcode.

Syntax

```
a1, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]

a1, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]
```

Initialization

instrnum -- Number of the instrument to be called.

« insname » -- A string (in double-quotes) representing a named instrument.

For more information about specifying input and output interfaces, see *Calling an Instrument within an Instrument*.

Performance

a1, ..., a8 -- The audio output from the called instrument. This is generated using the *signal output* opcodes.

p4, p5, ... -- Additional input values the are mapped to the called instrument p-fields, starting with p4.

The called instrument's p2 and p3 values will be identical to the host instrument's values. While the host instrument can *control its own duration*, any such attempts inside the called instrument will most likely have no effect.

See Also

Calling an Instrument within an Instrument, *event*, *schedule*, *subinstrinit*

Examples

Here is an example of the subinstr opcode. It uses the file *subinstr.csd* [examples/subinstr.csd].

Exemple 487. Example of the subinstr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subinstr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Creates a basic tone.
instr 1
  ; Print the value of p4, should be equal to
  ; Instrument #2's iamp field.
  print p4

  ; Print the value of p5, should be equal to
  ; Instrument #2's ipitch field.
  print p5

  ; Create a tone.
  asig oscils p4, p5, 0

  out asig
endin


; Instrument #2 - Demonstrates the subinstr opcode.
instr 2
  iamp = 20000
  ipitch = 440

  ; Use Instrument #1 to create a basic sine-wave tone.
  ; Its p4 parameter will be set using the iamp variable.
  ; Its p5 parameter will be set using the ipitch variable.
  abasic subinstr 1, iamp, ipitch

  ; Output the basic tone that we have created.
  out abasic
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Here is an example of the subinstr opcode using a named instrument. It uses the file *subinstr_named.csd* [examples/subinstr_named.csd].

Exemple 488. Example of the subinstr opcode using a named instrument.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subinstr_named.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument "basic_tone" - Creates a basic tone.
instr basic_tone
  ; Print the value of p4, should be equal to
  ; Instrument #2's iamp field.
  print p4

  ; Print the value of p5, should be equal to
  ; Instrument #2's ipitch field.
  print p5

  ; Create a tone.
  asig oscils p4, p5, 0

  out asig
endin


; Instrument #1 - Demonstrates the subinstr opcode.
instr 1
  iamp = 20000
  ipitch = 440

  ; Use the "basic_tone" named instrument to create a
  ; basic sine-wave tone.
  ; Its p4 parameter will be set using the iamp variable.
  ; Its p5 parameter will be set using the ipitch variable.
  abasic subinstr "basic_tone", iamp, ipitch

  ; Output the basic tone that we have created.
  out abasic
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

New in version 4.21

subinstrinit

subinstrinit — Creates and runs a numbered instrument instance at init-time.

Description

Same as *subinstr*, but init-time only and has no output arguments.

Syntax

```
subinstrinit instrnum [, p4] [, p5] [...]
```

```
subinstrinit "insname" [, p4] [, p5] [...]
```

Initialization

instrnum -- Number of the instrument to be called.

« insname » -- A string (in double-quotes) representing a named instrument.

For more information about specifying input and output interfaces, see *Calling an Instrument within an Instrument*.

Performance

p4, p5, ... -- Additional input values the are mapped to the called instrument p-fields, starting with p4.

The called instrument's p2 and p3 values will be identical to the host instrument's values. While the host instrument can *control its own duration*, any such attempts inside the called instrument will most likely have no effect.

See Also

Calling an Instrument within an Instrument, *event*, *schedule*, *subinstr*

Credits

New in version 4.23

sum

sum — Somme de n'importe quel nombre de signaux de taux-a.

Description

Somme de n'importe quel nombre de signaux de taux-a.

Syntaxe

```
ares sum asig1 [, asig2] [, asig3] [...]
```

Exécution

asig1, asig2, ... -- signaux de taux-a à additionner (à mélanger).

Crédits

Auteur : Gabriel Maldonado
Italie
Avril 1999

Nouveau dans le version 3.54 de Csound

svfilter

svfilter — A resonant second order filter, with simultaneous lowpass, highpass and bandpass outputs.

Description

Implementation of a resonant second order filter, with simultaneous lowpass, highpass and bandpass outputs.

Syntax

```
alow, ahigh, aband svfilter  asig, kcf, kq [, iscl]
```

Initialization

iscl -- coded scaling factor, similar to that in *reson*. A non-zero value signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

Performance

svfilter is a second order state-variable filter, with k-rate controls for cutoff frequency and Q. As Q is increased, a resonant peak forms around the cutoff frequency. *svfilter* has simultaneous lowpass, highpass, and bandpass filter outputs; by mixing the outputs together, a variety of frequency responses can be generated. The state-variable filter, or "multimode" filter was a common feature in early analog synthesizers, due to the wide variety of sounds available from the interaction between cutoff, resonance, and output mix ratios. *svfilter* is well suited to the emulation of "analog" sounds, as well as other applications where resonant filters are called for.

asig -- Input signal to be filtered.

kcf -- Cutoff or resonant frequency of the filter, measured in Hz.

kq -- Q of the filter, which is defined (for bandpass filters) as bandwidth/cutoff. *kq* should be in a range between 1 and 500. As *kq* is increased, the resonance of the filter increases, which corresponds to an increase in the magnitude and "sharpness" of the resonant peak. When using *svfilter* without any scaling of the signal (where *iscl* is either absent or 0), the volume of the resonant peak increases as Q increases. For high values of Q, it is recommended that *iscl* be set to a non-zero value, or that an external scaling function such as *balance* is used.

svfilter is based upon an algorithm in Hal Chamberlin's *Musical Applications of Microprocessor*s (Hayden Books, 1985).

Examples

Here is an example of the svfilter opcode. It uses the file *svfilter.csd* [examples/svfilter.csd].

Exemple 489. Example of the svfilter opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o svfilter.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The seperate outputs of the filter are scaled by values from the score,
; and are mixed together.
sr = 44100
kr = 2205
ksmps = 20
nchnls = 1

instr 1

  idur     = p3
  ifreq    = p4
  iamp     = p5
  ilowamp  = p6               ; determines amount of lowpass output in signal
  ihighamp = p7               ; determines amount of highpass output in signal
  ibandamp = p8               ; determines amount of bandpass output in signal
  iq       = p9               ; value of q

  iharms   =        (sr*.4) / ifreq

  asig    gbuzz 1, ifreq, iharms, 1, .9, 1             ; Sawtooth-like waveform
  kfreq   linseg 1, idur * 0.5, 4000, idur * 0.5, 1     ; Envelope to control filter cutoff

  alow, ahigh, aband   svfilter asig, kfreq, iq

  aout1   =         alow * ilowamp
  aout2   =         ahigh * ihighamp
  aout3   =         aband * ibandamp
  asum    =         aout1 + aout2 + aout3
  kenv    linseg 0, .1, iamp, idur -.2, iamp, .1, 0     ; Simple amplitude envelope
          out asum * kenv

endin


</CsInstruments>
<CsScore>

f1 0 8192 9 1 1 .25

i1  0 5 100 1000 1 0 0  5  ; lowpass sweep
i1  5 5 200 1000 1 0 0 30  ; lowpass sweep, octave higher, higher q
i1 10 5 100 1000 0 1 0  5  ; highpass sweep
i1 15 5 200 1000 0 1 0 30  ; highpass sweep, octave higher, higher q
i1 20 5 100 1000 0 0 1  5  ; bandpass sweep
i1 25 5 200 1000 0 0 1 30  ; bandpass sweep, octave higher, higher q
i1 30 5 200 2000 .4 .6  0  ; notch sweep - notch formed by combining highpass and lowpass outputs
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Sean Costello
Seattle, Washington
1999

New in Csound version 3.55

syncgrain

syncgrain — Synthèse granulaire synchrone.

Description

syncgrain implémente la synthèse granulaire synchrone. La source de son pour les grains est obtenue par la lecture d'une table de fonction contenant les échantillons de la forme d'onde source. Pour les sources de son échantillonné, on utilise *GEN01*. *syncgrain* acceptera des tables allouées en différé.

Le générateur de grain exerce un contrôle total sur la fréquence (grains/sec), l'amplitude générale, la hauteur du grain (incrément d'échantillonnage) et la taille du grain (en sec), comme paramètres constants ou variant dans le temps (signaux). Le taux du pointeur de grain est un paramètre supplémentaire qui contrôle à quelle position le générateur commencera à lire les échantillons dans la table pour chaque grain successif. Il est mesuré en fraction de la taille du grain ; s'il vaut 1 (la valeur par défaut) chaque grain successif est lu à partir de l'endroit où le grain précédent s'est terminé. S'il vaut 0,5 le grain suivant commencera à mi-chemin entre la position de début et la position de fin du grain précédent, etc. S'il vaut 0 le générateur lira toujours à partir de la même position dans la table (quelque soit l'endroit où le pointeur se trouvait juste avant). Avec une valeur négative le pointeur évoluera en décrémentant sa position. Ce contrôle apporte plus de flexibilité dans la création de modifications de l'échelle temporelle lors de la resynthèse.

syncgrain générera n'importe quel nombre de flux parallèles de grains (en fonction de la densité/fréquence de grains), borné supérieurement par la valeur de *iolaps* (100 par défaut). Le nombre de flux (grains se chevauchant) est déterminé par taille_du_grain*fréquence_du_grain. Plus il y aura de chevauchements de grains, plus il y aura de calculs et il se peut que la synthèse ne s'effectue pas en temps réel (cela dépend de la puissance du processeur).

syncgrain peut simuler une synthèse formantique à la FOF, si l'on utilise une forme d'enveloppe de grain adéquate et une sinusoïde comme forme d'onde du grain. Dans ce cas, on pourra utiliser des tailles de grain d'environ 0,04 sec. La fréquence centrale du formant est déterminée par la hauteur du grain. Comme l'incrément est en échantillons, si l'on veut utiliser une fréquence en Hz, cette valeur doit être multipliée par taille_de_la_table/*sr*. La fréquence du grain détermine le fondamental.

syncgrain utilise des indices en virgule flottante, ce qui fait qu'il n'est pas affecté par des tables de grande taille. Cet opcode est basé sur la class SyncGrain de la bibliothèque SndObj.

Syntaxe

```
asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \
      ifun2, iolaps
```

Initialisation

ifun1 -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

ifun2 -- table de fonction de l'enveloppe du grain.

iolaps -- nombre maximum de chevauchements, max(*kfreq*)*max(*kgrsize*). Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

Exécution

kamp -- pondération de l'amplitude.

kfreq -- fréquence de génération des grains, ou densité, en grains/sec.

kpitch -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

kgrsize -- taille du grain en secondes.

kprate -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

Exemples

Exemple 490. Exemple

```
iolaps  = 2
igrsize = 0.04
ifreq   = iolaps/igrsize
ips     = 1/iolaps

istr    = .5  /* timescale  */
ipitch  = 1   /* pitchscale */

a1 syncgrain 16000, ifreq, ipitch, igrsize, ips*istr, 1, 2, iolaps
   out       a1
```

Crédits

Auteur: Victor Lazzarini;
Janvier 2005

Nouveau plugin dans la version 5

Janvier 2005.

syncloop

syncloop — Synthèse granulaire synchrone.

Description

syncloop est une variation sur *syncgrain*, qui implémente la synthèse granulaire synchrone. *syncloop* ajoute des points de début et de fin de boucle et une position de départ facultative. Le début et la fin de boucle contrôlent les positions de démarrage des grains, si bien que les grains réalisés peuvent s'étendre au-delà des points de la boucle (si les points de la boucle ne sont pas aux extrémités de la table), ce qui permet des transitions fluides. Pour plus d'information sur le procédé de synthèse granulaire, voir la page du manuel sur *syncgrain*.

Syntaxe

```
asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \
      klend, ifun1, ifun2, iolaps[,istart, iskip]
```

Initialisation

ifun1 -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

ifun2 -- table de fonction de l'enveloppe du grain.

iolaps -- nombre maximum de chevauchements, max(*kfreq*)*max(*kgrsize*). Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

istart -- point de départ de la synthèse en secs (0 par défaut).

iskip -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position à l'intérieur de la boucle où la note précédente s'est terminée. La valeur par défaut de 0 signifie que l'initialisation n'est pas ignorée.

Exécution

kamp -- pondération de l'amplitude.

kfreq -- fréquence de génération des grains, ou densité, en grains/sec.

kpitch -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

kgrsize -- taille du grain en secondes.

kprate -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

klstart -- début de la boucle en secs.

klend -- fin de la boucle en secs.

Exemples

Exemple 491. Exemple

```
iolaps  = 2
igrsize = 0.04
ifreq   = iolaps/igrsize
ips     = 1/iolaps

istr    = .5  /* timescale  */
ipitch  = 1   /* pitchscale */

a1 syncloop 16000, ifreq, ipitch, igrsize, ips*istr, 1, 2, 1, 2, iolaps
   out      a1
```

Crédits

Auteur : Victor Lazzarini;
Janvier 2005

Nouveau plugin dans la version 5

Janvier 2005.

syncphasor

syncphasor — Produit une valeur de phase mobile normalisée avec entrée et sortie de synchronisation.

Description

Produit une valeur de phase mobile entre zéro et un et une impulsion supplémentaire en sortie ("sync out") chaque fois que sa valeur de phase traverse le zéro ou est remise à zéro. La phase peut être réinitialisée à tout instant par une impulsion sur le paramètre "sync in".

Syntaxe

```
aphase, asyncout syncphasor xcps, asyncin, [, iphs]
```

Initialisation

iphs (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

Exécution

aphase -- la valeur de phase en sortie ; toujours entre 0 et 1.

asyncout -- la sortie de synchronisation prend la valeur 1.0 durant un échantillon chaque fois que la valeur de phase traverse le zéro ou que l'entrée de synchronisation a une valeur non nulle. Elle vaut zéro aux autres moments.

asyncin -- l'entrée de synchronisation provoque la remise à zéro de la phase chaque fois que *asyncin* est non nul.

xcps -- fréquence du phaseur en Hertz. Si *xcps* est négatif, la phase sera décrémentée de 1 à 0 au lieu d'être incrémentée.

Une phase interne est augmentée successivement selon la fréquence de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle 0 <= phs < 1. Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

La phase de *syncphasor* peut être synchronisée à un autre phaseur (ou à un autre signal) au moyen du paramètre *asyncin*. Chaque fois que *asyncin* prend une valeur non nulle, la valeur de *aphase* est remise à zéro. *syncphasor* sort aussi son propre signal de "synchro" qui consiste en une impulsion d'un échantillon chaque fois que sa phase traverse le zéro ou est réinitialisée. On peut ainsi facilement mettre en série plusieurs opcodes *syncphasor* pour créer un effet d'oscillateur "hard sync".

Exemples

Voici un exemple de l'opcode syncphasor. Il utilise le fichier *syncphasor.csd* [examples/syncphasor.csd].

Exemple 492. Exemple de l'opcode syncphasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

instr 1
    ; Use two syncphasors - one is the "master",
    ; the other the "slave"

    ; master's frequency determines pitch
    imastercps  =           cpspch(p4)
    imaxamp     =           10000

    ; the slave's frequency affects the timbre
    kslavecps   line        imastercps, p3, imastercps * 3

    ; the master "oscillator"
    ; the master has no sync input
    anosync     init        0.0
    am, async   syncphasor  imastercps, anosync

    ; the slave "oscillator"
    aout, as    syncphasor  kslavecps, async

    adeclick    linseg      0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

    ; Output the slave's phase value which is a rising
    ; sawtooth wave.  This produces aliasing, but hey, this
    ; this is just an example ;)

              out         aout * adeclick * imaxamp
endin

</CsInstruments>
<CsScore>

i1 0 1    7.00
i1 + 0.5  7.02
i1 + .    7.05
i1 + .    7.07
i1 + .    7.09
i1 + 2    7.06

e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode syncphasor. Il utilise le fichier *syncphasor-CZresonance.csd* [examples/syncphasor-CZresonance.csd].

Exemple 493. Un autre exemple de l'opcode syncphasor.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o syncphasor-CZresonance.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; by Anthony Kozar. February 2008
```

```
; http://www.anthonykozar.net/

; Imitation of the Casio CZ-series synthesizer's "Resonance" waveforms
; using a synced phasor to read a sinusoid table.  The jumps at the sync
; points are smoothed by multiplying with a windowing function controlled
; by the master phasor.

; Based on information from the Wikipedia article on phase distortion:
; http://en.wikipedia.org/wiki/Phase_distortion_synthesis

; Sawtooth Resonance waveform. Smoothing function is just the inverted
; master phasor.

; The Wikipedia article shows an inverted cosine as the stored waveform,
; which implies that it must be unipolar for the smoothing to work.
; I have substituted a sine wave in the first phrase to keep the output
; bipolar.  The second phrase demonstrates the much "rezzier" sound of the
; bipolar cosine due to discontinuities.

instr 1
  ifreq      =          cpspch(p4)
  initReson  =          p5
  itable     =          p6
  imaxamp    =          10000
  anosync    init       0.0

  kslavecps       line        ifreq * initReson, p3, ifreq
  amaster, async  syncphasor  ifreq, anosync          ; pair of phasors
  aslave, async2  syncphasor  kslavecps, async          ; slave synced to master
  aosc            tablei      aslave, itable, 1          ; use slave phasor to read a (co)sine table
  aout            =           aosc * (1.0 - amaster) ; inverted master smoothes jumps
  adeclick        linseg      0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

                    out         aout * adeclick * imaxamp
endin

; Triangle or Trapezoidal Resonance waveform. Uses a second table to change
; the shape of the smoothing function.  (This is my best guess so far as to
; how these worked). The cosine table works fine with the triangular smoothing
; but we once again need to use a sine table with the trapezoidal smoothing.

; (It might be interesting to be able to vary the "width" of the trapezoid.
; This could be done with the pdhalf opcode).

instr 2
  ifreq      =          cpspch(p4)
  initReson  =          p5
  itable     =          p6
  ismoothtbl =          p7
  imaxamp    =          10000
  anosync    init       0.0

  kslavecps       line        ifreq * initReson, p3, ifreq
  amaster, async  syncphasor  ifreq, anosync          ; pair of phasors
  aslave, async2  syncphasor  kslavecps, async          ; slave synced to master
  aosc            tablei      aslave, itable, 1          ; use slave phasor to read a (co)sine table
  asmooth         tablei      amaster, ismoothtbl, 1 ; use master phasor to read smoothing table
  aout            =           aosc * asmooth
  adeclick        linseg      0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

  out          aout * adeclick * imaxamp
endin

</CsInstruments>
<CsScore>
f1 0 16385 10  1
f3 0 16385  9  1 1 270                      ; inverted cosine
f5 0 4097   7  0.0 2048 1.0 2049 0.0 ; unipolar triangle
f6 0 4097   7  1.0 2048 1.0 2049 0.0     ; "trapezoid"

; Sawtooth resonance with a sine table
i1 0 1    7.00  5.0  1
i. + 0.5  7.02  4.0
i. + .    7.05  3.0
i. + .    7.07  2.0
i. + .    7.09  1.0
i. + 2    7.06  12.0
f0 6
s

; Sawtooth resonance with a cosine table
```

```
i1 0 1    7.00  5.0  3
i. + 0.5  7.02  4.0
i. + .    7.05  3.0
i. + .    7.07  2.0
i. + .    7.09  1.0
i. + 2    7.06  12.0
f0 6
s

; Triangle resonance with a cosine table
i2 0 1    7.00  5.0  3  5
i. + 0.5  7.02  4.0
i. + .    7.05  3.0
i. + .    7.07  2.0
i. + .    7.09  1.0
i. + 2    7.06  12.0
f0 6
s

; Trapezoidal resonance with a sine table
i2 0 1    7.00  5.0  1  6
i. + 0.5  7.02  4.0
i. + .    7.05  3.0
i. + .    7.07  2.0
i. + .    7.09  1.0
i. + 2    7.06  12.0

e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

phasor.

Et les opcodes d'*Accès aux Table* comme : *table*, *tablei*, *table3* et *tab*.

Crédits

Adapté d'après l'opcode *phasor* par Anthony Kozar
Janvier 2008

Nouveau dans la version 5.08 de Csound

system

system — Call an external program via the system call

Description

system and **system_i** call any external command understood by the operating system, similarly to the C function system(). **system_i** runs at i-time only, while **system** runs both at initialization and performance time.

Syntax

```
ires system_i itrig, Scmd, [inowait]
```

```
kres system ktrig, Scmd, [knowait]
```

Initialization

Scmd -- command string

itrig -- if greater than zero the opcode performs the printing; otherwise it is an null operation.

Performance

ktrig -- if greater than zero and different from the value on the previous control cycle the opcode performs the requested printing. Initially this previous value is taken as zero.

inowait,knowait -- if given an non zero the command is run in the background and the command does not wait for the result. (default = 0)

ires, kres -- the return code of the command in wait mode and if the command is run.In other cases returns zero.

More than one system command (a script) can be executed with a single **system** opcode by using double braces strings {{ }}.

Note

This opcode is very system dependant, so should be used with extreme care (or not used) if platform neutrality is desired.

Example

Here is an example of the system_i opcode. It uses the file *system.csd* [examples/system.csd].

Exemple 494. Example of the system opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          ; -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o system.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; Waits for command to execute before continuing
ires system_i 1,{{     ps
            date
            cd ~/Desktop
            pwd
            ls -l
            whois csounds.com
        }}
print ires
turnoff
endin

instr 2
; Runs command in a separate thread
ires system_i 1,{{     ps
            date
            cd ~/Desktop
            pwd
            ls -l
            whois csounds.com
        }}, 1

print ires
turnoff
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 1
i 2 5 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch
2007

New in version 5.06

tb

tb0, tb1, tb2, tb3, tb4, tb5, tb6, tb7, tb8, tb9, tb10, tb11, tb12, tb13, tb14, tb15, tb0_init, tb1_init, tb2_init, tb3_init, tb4_init, tb5_init, tb6_init, tb7_init, tb8_init, tb9_init, tb10_init, tb11_init, tb12_init, tb13_init, tb14_init, tb15_init — Table Read Access inside expressions.

Description

Allow to read tables in function fashion, to be used inside expressions. At present time Csound only supports functions with a single input argument. However, to access table elements, user must provide two numbers, i.e. the number of table and the index of element. So, in order to allow to access a table element with a function, a previous preparation step should be done.

Syntax

```
tb0_init ifn
```

```
tb1_init ifn
```

```
tb2_init ifn
```

```
tb3_init ifn
```

```
tb4_init ifn
```

```
tb5_init ifn
```

```
tb6_init ifn
```

```
tb7_init ifn
```

```
tb8_init ifn
```

```
tb9_init ifn
```

```
tb10_init ifn
```

```
tb11_init ifn
```

```
tb12_init ifn
```

```
tb13_init ifn
```

```
tb14_init ifn
```

```
tb15_init ifn
```

```
iout = tb0(iIndex)
```

```
kout = tb0(kIndex)
```

```
iout = tb1(iIndex)

kout = tb1(kIndex)

iout = tb2(iIndex)

kout = tb2(kIndex)

iout = tb3(iIndex)

kout = tb3(kIndex)

iout = tb4(iIndex)

kout = tb4(kIndex)

iout = tb5(iIndex)

kout = tb5(kIndex)

iout = tb6(iIndex)

kout = tb6(kIndex)

iout = tb7(iIndex)

kout = tb7(kIndex)

iout = tb8(iIndex)

kout = tb8(kIndex)

iout = tb9(iIndex)

kout = tb9(kIndex)

iout = tb10(iIndex)

kout = tb10(kIndex)

iout = tb11(iIndex)

kout = tb11(kIndex)

iout = tb12(iIndex)

kout = tb12(kIndex)

iout = tb13(iIndex)

kout = tb13(kIndex)

iout = tb14(iIndex)
```

```
kout = tb14(kIndex)

iout = tb15(iIndex)

kout = tb15(kIndex)
```

Performance

There are 16 different opcodes whose name is associated with a number from 0 to 15. User can associate a specific table with each opcode (so the maximum number of tables that can be accessed in function fashion is 16). Prior to access a table, user must associate the table with one of the 16 opcodes by means of an opcode chosen among tb0_init...tb15_init. For example,

tb0_init 1

associates table 1 with tb0( ) function, so that, each element of table 1 can be accessed (in function fashion) with:

kvar = tb0(k_some_index_of_table1) * k_some_other_var

ivar = tb0(i_some_index_of_table1) + i_some_other_var etc...

By using these opcodes, user can drastically reduce the number of lines of an orchestra, improving its readability.

Credits

Written by Gabriel Maldonado.

tab

tab — Fast table opcodes.

Description

Fast table opcodes. Faster than table and tablew because don't allow wrap-around and limit and don't check index validity. Have been implemented in order to provide fast access to arrays. Support non-power of two tables (can be generated by any GEN function by giving a negative length value).

Syntax

```
ir tab_i indx, ifn[, ixmode]
```

```
kr tab kndx, ifn[, ixmode]
```

```
ar tab xndx, ifn[, ixmode]
```

```
tabw_i isig, indx, ifn [,ixmode]
```

```
tabw ksig, kndx, ifn [,ixmode]
```

```
tabw asig, andx, ifn [,ixmode]
```

Initialization

ifn -- table number

ixmode -- defaults to zero. If zero xndx and ixoff ranges match the length of the table; if non zero xndx and ixoff have a 0 to 1 range.

isig -- input value to write.

indx -- table index

Performance

asig, ksig -- input signal to write.

andx, kndx -- table index.

tab and *tabw* opcodes are similar to *table* and *tablew*, but are faster and support tables having non-power-of-two length.

Special care of index value must be taken into account. Index values out of the table allocated space will crash Csound.

Credits

Written by Gabriel Maldonado.

tabrec

tabrec — Recording of control signals.

Description

Records control-rate signals on trigger-temporization basis.

Syntax

```
tabrec   ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]
```

Performance

ktrig_start -- start recording when non-zero.

ktrig_stop -- stop recording when knumtics trigger impulses are received by this input argument.

knumtics -- stop recording or reset playing pointer to zero when the number of tics defined by this argument is reached.

kfn -- table where k-rate signals are recorded.

kin1,...,kinN -- input signals to record.

The *tabrec* and *tabplay* opcodes allow to record/playback control signals on trigger-temporization basis.

tabrec opcode records a group of k-rate signals by storing them into kfn table. Each time ktrig_start is triggered, tabrec resets the table pointer to zero and begins to record. Recording phase stops after knumtics trigger impluses have been received by ktrig_stop argument.

These opcodes can be used like a sort of ``middle-term'' memory that ``remembers'' generated signals. Such memory can be used to supply generative music with a coherent iterative compositional structure.

See Also

tabplay

Credits

Written by Gabriel Maldonado.

table

table — Accesses table values by direct indexing.

Description

Accesses table values by direct indexing.

Syntax

```
ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]

ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]

kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

Initialization

ifn -- function table number.

ixmode (optional) -- index data mode. The default value is 0.

- 0 = raw index
- 1 = normalized (0 to 1)

ixoff (optional) -- amount by which index is to be offset. For a table with origin at center, use tablesize/2 (raw) or .5 (normalized). The default value is 0.

iwrap (optional) -- wraparound index flag. The default value is 0.

- 0 = nowrap (index < 0 treated as index=0; index tablesize sticks at index=size)
- 1 = wraparound.

Performance

table invokes table lookup on behalf of init, control or audio indices. These indices can be raw entry numbers (0,l,2...size - 1) or scaled values (0 to 1-e). Indices are first modified by the offset value then checked for range before table lookup (see *iwrap*). If index is likely to be full scale, or if interpolation is being used, the table should have an extended guard point. *table* indexed by a periodic phasor (see *phasor*) will simulate an oscillator.

Examples

Here is an example of the table opcode. It uses the file *table.csd* [examples/table.csd].

Exemple 495. Example of the table opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o table.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Vary our index linearly from 0 to 1.
  kndx line 0, p3, 1

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kfreq table kndx, ifn, ixmode

  ; Generate a sine waveform, use our table values
  ; to vary its frequency.
  a1 oscil 20000, kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

tablei, *table3*, *oscil1*, *oscil1i*, *osciln*

Credits

Example written by Kevin Conder.

table3

table3 — Accesses table values by direct indexing with cubic interpolation.

Description

Accesses table values by direct indexing with cubic interpolation.

Syntax

```
ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

Initialization

ifn -- function table number.

ixmode (optional) -- index data mode. The default value is 0.

- 0 = raw index
- 1 = normalized (0 to 1)

ixoff (optional) -- amount by which index is to be offset. For a table with origin at center, use tablesize/2 (raw) or .5 (normalized). The default value is 0.

iwrap (optional) -- wraparound index flag. The default value is 0.

- 0 = nowrap (index < 0 treated as index=0; index tablesize sticks at index=size)
- 1 = wraparound.

Performance

table3 is identical to *tablei*, except that it uses cubic interpolation. (New in Csound version 3.50.)

See Also

table, *tablei*, *oscil1*, *oscil1i*, *osciln*

tablecopy

tablecopy — Simple, fast table copy opcode.

Description

Simple, fast table copy opcode.

Syntax

```
tablecopy kdft, ksft
```

Performance

kdft -- Destination function table.

ksft -- Number of source function table.

tablecopy -- Simple, fast table copy opcode. Takes the table length from the destination table, and reads from the start of the source table. For speed reasons, does not check the source length - just copies regardless - in « wrap » mode. This may read through the source table several times. A source table with length 1 will cause all values in the destination table to be written to its value.

tablecopy cannot read or write the guardpoint. To read it use *table*, with *ndx* = the table length. Likewise use table write to write it.

To write the guardpoint to the value in location 0, use *tablegpw*.

This is primarily to change function tables quickly in a real-time situation.

See Also

tablegpw, *tablemix*, *tableicopy*, *tableigpw*, *tableimix*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tablegpw

tablegpw — Writes a table's guard point.

Description

Writes a table's guard point.

Syntax

```
tablegpw kfn
```

Performance

kfn -- Table number to be interrogated

tablegpw -- For writing the table's guard point, with the value which is in location 0. Does nothing if table does not exist.

Likely to be useful after manipulating a table with *tablemix* or *tablecopy*.

See Also

tablecopy, *tablemix*, *tableicopy*, *tableigpw*, *tableimix*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tablei

tablei — Accesses table values by direct indexing with linear interpolation.

Description

Accesses table values by direct indexing with linear interpolation.

Syntax

```
ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

Initialization

ifn -- function table number. *tablei* requires the extended guard point.

ixmode (optional) -- index data mode. The default value is 0.

- 0 = raw index
- 1 = normalized (0 to 1)

ixoff (optional) -- amount by which index is to be offset. For a table with origin at center, use tablesize/2 (raw) or .5 (normalized). The default value is 0.

iwrap (optional) -- wraparound index flag. The default value is 0.

- 0 = nowrap (index < 0 treated as index=0; index tablesize sticks at index=size)
- 1 = wraparound.

Performance

tablei is a interpolating unit in which the fractional part of index is used to interpolate between adjacent table entries. The smoothness gained by interpolation is at some small cost in execution time (see also *oscili*, etc.), but the interpolating and non-interpolating units are otherwise interchangeable. Note that when *tablei* uses a periodic index whose modulo *n* is less than the power of 2 table length, the interpolation process requires that there be an (*n*+ 1)th table value that is a repeat of the 1st (see *f Statement* in score).

See Also

table, *table3*, *oscil1*, *oscil1i*, *osciln*

tableicopy

tableicopy — Simple, fast table copy opcode.

Description

Simple, fast table copy opcode.

Syntax

```
tableicopy idft, isft
```

Initialization

idft -- Destination function table.

isft -- Number of source function table.

Performance

tableicopy -- Simple, fast table copy opcodes. Takes the table length from the destination table, and reads from the start of the source table. For speed reasons, does not check the source length - just copies regardless - in "wrap" mode. This may read through the source table several times. A source table with length 1 will cause all values in the destination table to be written to its value.

tableicopy cannot read or write the guardpoint. To read it use *table*, with *ndx* = the table length. Likewise use table write to write it.

To write the guardpoint to the value in location 0, use *tablegpw*.

This is primarily to change function tables quickly in a real-time situation.

See Also

tablecopy, *tablegpw*, *tablemix*, *tableigpw*, *tableimix*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tableigpw

tableigpw — Writes a table's guard point.

Description

Writes a table's guard point.

Syntax

```
tableigpw ifn
```

Initialization

ifn -- Table number to be interrogated

Performance

tableigpw -- For writing the table's guard point, with the value which is in location 0. Does nothing if table does not exist.

Likely to be useful after manipulating a table with *tablemix* or *tablecopy*.

See Also

tablecopy, *tablegpw*, *tablemix*, *tableicopy*, *tableimix*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tableikt

tableikt — Provides k-rate control over table numbers.

Description

k-rate control over table numbers.

The standard Csound opcode *tablei*, when producing a k- or a-rate result, can only use an init-time variable to select the table number. *tableikt* accepts k-rate control as well as i-time. In all other respects they are similar to the original opcodes.

Syntax

```
ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tableikt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

Initialization

ixmode -- if 0, *xndx* and *ixoff* ranges match the length of the table. if non-zero *xndx* and *ixoff* have a 0 to 1 range. Default is 0

ixoff -- if 0, total index is controlled directly by *xndx,* ie. the indexing starts from the start of the table. If non-zero, start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* not equal to 0). Default is 0.

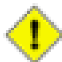
iwrap -- if *iwrap* = 0, *Limit mode*: when total index is below 0, then final index is 0.Total index above table length results in a final index of the table length - high out of range total indexes stick at the upper limit of the table. If *iwrap* not equal to 0, *Wrap mode*: total index is wrapped modulo the table length so that all total indexes map into the table. For instance, in a table of length 8, *xndx* = 5 and *ixoff* = 6 gives a total index of 11, which wraps to a final index of 3. Default is 0.

Performance

kndx -- Index into table, either a positive number range

xndx -- matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* not equal to 0)

kfn -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.



Caution with k-rate table numbers

At k-rate, if a table number of < 1 is given, or the table number points to a non-existent table, or to one which has a length of 0 (it is to be loaded from a file later) then an error will result and the instrument will be deactivated. *kfn* must be initialized at the appropriate rate using *init*. Attempting to load an i-rate value into *kfn* will result in an error.

See Also

tablekt

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tableimix

tableimix — Mixes two tables.

Description

Mixes two tables.

Syntax

```
tableimix idft, idoff, ilen, is1ft, is1off, is1g, is2ft, is2off, is2g
```

Initialization

idft -- Destination function table.

idoff -- Offset to start writing from. Can be negative.

ilen -- Number of write operations to perform. Negative means work backwards.

is1ft, *is2ft* -- Source function tables. These can be the same as the destination table, if care is exercised about direction of copying data.

is1off, *is2off* -- Offsets to start reading from in source tables.

is1g, *is2g* -- Gains to apply when reading from the source tables. The results are added and the sum is written to the destination table.

Performance

tableimix -- This opcode mixes from two tables, with separate gains into the destination table. Writing is done for *klen* locations, usually stepping forward through the table - if *klen* is positive. If it is negative, then the writing and reading order is backwards - towards lower indexes in the tables. This bi-directional option makes it easy to shift the contents of a table sideways by reading from it and writing back to it with a different offset.

If *klen* is 0, no writing occurs. Note that the internal integer value of *klen* is derived from the ANSI C floor() function - which returns the next most negative integer. Hence a fractional negative *klen* value of -2.3 would create an internal length of 3, and cause the copying to start from the offset locations and proceed for two locations to the left.

The total index for table reading and writing is calculated from the starting offset for each table, plus the index value, which starts at 0 and then increments (or decrements) by 1 as mixing proceeds.

These total indexes can potentially be very large, since there is no restriction on the offset or the *klen*. However each total index for each table is ANDed with a length mask (such as 0000 0111 for a table of length 8) to form a final index which is actually used for reading or writing. So no reading or writing can occur outside the tables. This is the same as « wrap » mode in table read and write. These opcodes do not read or write the guardpoint. If a table has been rewritten with one of these, then if it has a guard-point which is supposed to contain the same value as the location 0, then call *tablegpw* afterwards.

The indexes and offsets are all in table steps - they are not normalized to 0 - 1. So for a table of length 256, *klen* should be set to 256 if all the table was to be read or written.

The tables do not need to be the same length - wrapping occurs individually for each table.

See Also

tablecopy, *tablegpw*, *tablemix*, *tableicopy*, *tableigpw*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tableiw

tableiw — Change the contents of existing function tables.

Description

This opcode operates on existing function tables, changing their contents. *tableiw* is used when all inputs are init time variables or constants and you only want to run it at the initialization of the instrument. The valid combinations of variable types are shown by the first letter of the variable names.

Syntax

```
tableiw isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

Initialization

isig -- Input value to write to the table.

indx -- Index into table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* not equal to 0)

ifn -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.

ixmode (optional, default=0) -- index mode.

- 0 = *indx* and *ixoff* ranges match the length of the table.
- not equal to 0 = *indx* and *ixoff* have a 0 to 1 range.

ixoff (optional, default=0) -- index offset.

- 0 = Total index is controlled directly by *indx*, i.e. the indexing starts from the start of the table.
- Not equal to 0 = Start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* not equal to 0).

iwgmode (optional, default=0) -- Wrap and guard point mode.

- 0 = Limit mode.
- 1 = Wrap mode.
- 2 = Guardpoint mode.

Performance

Limit mode (0)

Limit the total index (*indx* + *ixoff*) to between 0 and the guard point. For a table of length 5, this means that locations 0 to 3 and location 4 (the guard point) can be written. A negative total index writes to location 0.

Wrap mode (1)

Wrap total index value into locations 0 to E, where E is either one less than the table length or the factor of 2 number which is one less than the table length. For example, wrap into a 0 to 3 range - so that total index 6 writes to location 2.

Guardpoint mode (2)

The guardpoint is written at the same time as location 0 is written - with the same value.

This facilitates writing to tables which are intended to be read with interpolation for producing smooth cyclic waveforms. In addition, before it is used, the total index is incremented by half the range between one location and the next, before being rounded down to the integer address of a table location.

Normally (*iwgmode* = 0 or 1) for a table of length 5 - which has locations 0 to 3 as the main table and location 4 as the guard point, a total index in the range of 0 to 0.999 will write to location 0. ("0.999" means just less than 1.0.) 1.0 to 1.999 will write to location 1 etc. A similar pattern holds for all total indexes 0 to 4.999 (*igwmode* = 0) or to 3.999 (*igwmode* = 1). *igwmode* = 0 enables locations 0 to 4 to be written - with the guardpoint (4) being written with a potentially different value from location 0.

With a table of length 5 and the *iwgmode* = 2, then when the total index is in the range 0 to 0.499, it will write to locations 0 and 4. Range 0.5 to 1.499 will write to location 1 etc. 3.5 to 4.0 will also write to locations 0 and 4.

This way, the writing operation most closely approximates the results of interpolated reading. Guard point mode should only be used with tables that have a guardpoint.

Guardpoint mode is accomplished by adding 0.5 to the total index, rounding to the next lowest integer, wrapping it modulo the factor of two which is one less than the table length, writing the table (locations 0 to 3 in our example) and then writing to the guard point if index = 0.

See Also

tablew, *tablewkt*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

Updated August 2002, thanks go to Abram Hindle for pointing out the correct syntax.

tablekt

tablekt — Provides k-rate control over table numbers.

Description

k-rate control over table numbers.

The standard Csound opcode *table* when producing a k- or a-rate result, can only use an init-time variable to select the table number. *tablekt* accepts k-rate control as well as i-time. In all other respects they are similar to the original opcodes.

Syntax

```
ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tablekt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

Initialization

ixmode -- if 0, *xndx* and *ixoff* ranges match the length of the table. if non-zero *xndx* and *ixoff* have a 0 to 1 range. Default is 0

ixoff -- if 0, total index is controlled directly by *xndx,* ie. the indexing starts from the start of the table. If non-zero, start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* not equal to 0). Default is 0.

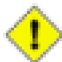
iwrap -- if *iwrap* = 0, *Limit mode*: when total index is below 0, then final index is 0.Total index above table length results in a final index of the table length - high out of range total indexes stick at the upper limit of the table. If *iwrap* not equal to 0, *Wrap mode*: total index is wrapped modulo the table length so that all total indexes map into the table. For instance, in a table of length 8, *xndx* = 5 and *ixoff* = 6 gives a total index of 11, which wraps to a final index of 3. Default is 0.

Performance

kndx -- Index into table, either a positive number range

xndx -- matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* not equal to 0)

kfn -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.



Caution with k-rate table numbers

At k-rate, if a table number of < 1 is given, or the table number points to a non-existent table, or to one which has a length of 0 (it is to be loaded from a file later) then an error will result and the instrument will be deactivated. *kfn* must be initialized at the appropriate rate using *init*. Attempting to load an i-rate value into *kfn* will result in an error.

See Also

tableikt

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tablemix

tablemix — Mixes two tables.

Description

Mixes two tables.

Syntax

```
tablemix kdft, kdoff, klen, ks1ft, ks1off, ks1g, ks2ft, ks2off, ks2g
```

Performance

kdft -- Destination function table.

kdoff -- Offset to start writing from. Can be negative.

klen -- Number of write operations to perform. Negative means work backwards.

ks1ft, *ks2ft* -- Source function tables. These can be the same as the destination table, if care is exercised about direction of copying data.

ks1off, *ks2off* -- Offsets to start reading from in source tables.

ks1g, *ks2g* -- Gains to apply when reading from the source tables. The results are added and the sum is written to the destination table.

tablemix -- This opcode mixes from two tables, with separate gains into the destination table. Writing is done for *klen* locations, usually stepping forward through the table - if *klen* is positive. If it is negative, then the writing and reading order is backwards - towards lower indexes in the tables. This bi-directional option makes it easy to shift the contents of a table sideways by reading from it and writing back to it with a different offset.

If *klen* is 0, no writing occurs. Note that the internal integer value of *klen* is derived from the ANSI C floor() function - which returns the next most negative integer. Hence a fractional negative *klen* value of -2.3 would create an internal length of 3, and cause the copying to start from the offset locations and proceed for two locations to the left.

The total index for table reading and writing is calculated from the starting offset for each table, plus the index value, which starts at 0 and then increments (or decrements) by 1 as mixing proceeds.

These total indexes can potentially be very large, since there is no restriction on the offset or the *klen*. However each total index for each table is ANDed with a length mask (such as 0000 0111 for a table of length 8) to form a final index which is actually used for reading or writing. So no reading or writing can occur outside the tables. This is the same as « wrap » mode in table read and write. These opcodes do not read or write the guardpoint. If a table has been rewritten with one of these, then if it has a guard-point which is supposed to contain the same value as the location 0, then call *tablegpw* afterwards.

The indexes and offsets are all in table steps - they are not normalized to 0 - 1. So for a table of length 256, *klen* should be set to 256 if all the table was to be read or written.

The tables do not need to be the same length - wrapping occurs individually for each table.

See Also

tablecopy, *tablegpw*, *tableicopy*, *tableigpw*, *tableimix*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

tableng

tableng — Interrogates a function table for length.

Description

Interrogates a function table for length.

Syntax

```
ires tableng ifn
```

```
kres tableng kfn
```

Initialization

ifn -- Table number to be interrogated

Performance

kfn -- Table number to be interrogated

tableng returns the length of the specified table. This will be a power of two number in most circumstances. It will not show whether a table has a guardpoint or not. It seems this information is not available in the table's data structure. If the specified table is not found, then 0 will be returned.

Likely to be useful for setting up code for table manipulation operations, such as *tablemix* and *tablecopy*.

Examples

Here is an example of the tableng opcode. It uses the file *tableng.csd* [examples/tableng.csd].

Exemple 496. Example of the tableng opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tableng.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Let's look at Table #1.
  ifn = 1
  ilen tableng ifn

  print ilen
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

The table is 16,384 samples long. So its output should include a line like this:

```
instr 1:  ilen = 16384.000
```

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

tablera

tablera — Reads tables in sequential locations.

Description

These opcode reads tables in sequential locations to an a-rate variable. Some thought is required before using it. It has at least two major, and quite different, applications which are discussed below.

Syntax

```
ares tablera kfn, kstart, koff
```

Performance

ares -- a-rate destination for reading *ksmps* values from a table.

kfn -- i- or k-rate number of the table to read or write.

kstart -- Where in table to read or write.

koff -- i- or k-rate offset into table. Range unlimited - see explanation at end of this section.

In one application, *tablera* is intended to be used in pair with *tablewa*, or with several *tablera* opcodes before a *tablewa* -- all sharing the same *kstart* variable.

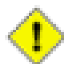
These read from and write to sequential locations in a table at audio rates, with *ksmps* floats being written and read each cycle.

tablera starts reading from location *kstart*. *tablewa* starts writing to location *kstart*, and then writes to *kstart* with the number of the location one more than the one it last wrote. (Note that for *tablewa*, *kstart* is both an input and output variable.) If the writing index reaches the end of the table, then no further writing occurs and zero is written to *kstart*.

For instance, if the table's length was 16 (locations 0 to 15), and *ksmps* was 5. Then the following steps would occur with repetitive runs of the *tablewa* opcode, assuming that *kstart* started at 0.

Run Number	Initial kstart	Final kstart	Locations Written
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

This is to facilitate processing table data using standard a-rate orchestra code between the *tablera* and *tablewa*opcodes. They allow all Csound k-rate operators to be used (with caution) on a-rate variables - something that would only be possible otherwise by *ksmps* = 1, downsamp and upsamp.



Several cautions

- The k-rate code in the processing loop is really running at a-rate, so time dependent functions like *port* and *oscil* work faster than normal - their code is expecting to be running at k-rate.
- This system will produce undesirable results unless the *ksmps* fits within the table length. For instance a table of length 16 will accommodate 1 to 16 samples, so this example will work with *ksmps* = 1 to 16.

Both these opcodes generate an error and deactivate the instrument if a table with length < *ksmps* is selected. Likewise an error occurs if *kstart* is below 0 or greater than the highest entry in the table - if *kstart* = table length.

- *kstart* is intended to contain integer values between 0 and (table length - 1). Fractional values above this should not affect operation but do not achieve anything useful.
- These opcodes are not interpolating, and the *kstart* and *koff* parameters always have a range of 0 to (table length - 1) - not 0 to 1 as is available in other table read/write opcodes. *koff* can be outside this range but it is wrapped around by the final AND operation.
- These opcodes are permanently in wrap mode. When *koff* is 0, no wrapping needs to occur, since the *kstart*++ index will always be within the table's normal range. *koff* not equal to 0 can lead to wrapping.
- The offset does not affect the number of read/write cycles performed, or the value written to *kstart* by *tablewa*.
- These opcodes cannot read or write the guardpoint. Use *tablegpw* to write the guardpoint after manipulations have been done with *tablewa*.

Examples

```
kstart   =       0

lab1:
  atemp  tablera ktabsource, kstart, 0  ; Read 5 values from table into an
         ; a-rate variable.

  atemp  =       log(atemp)  ; Process the values using a-rate
         ; code.

  kstart tablewa ktabdest, atemp, 0   ; Write it back to the table

if ktemp  0 goto lab1      ; Loop until all table locations
         ; have been processed.
```

The above example shows a processing loop, which runs every k-cycle, reading each location in the table *ktabsource*, and writing the log of those values into the same locations of table *ktabdest*.

This enables whole tables, parts of tables (with offsets and different control loops) and data from several tables at once to be manipulated with a-rate code and written back to another (or to the same) table. This is a bit of a fudge, but it is faster than doing it with k-rate table read and write code.

Another application is:

```
kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0  ; ksmps a-rate samples written
       ; into locations 0 to (ksmps -1) of table 23.

lab1: ktemp table kloop, 23  ; Start a loop which runs ksmps times,
       ; in which each cycle processes one of
 [ Some code to manipulate ]  ; table 23's values with k-rate orchestra
 [ the value of ktemp. ]  ; code.

 tablew ktemp, kloop, 23  ; Write the processed value to the table.

kloop = kloop + 1      ; Increment the kloop, which is both the
       ; pointer into the table and the loop
if kloop < ksmps goto lab1  ; counter. Keep looping until all values
       ; in the table have been processed.

asignal   tablera 23, 0, 0  ; Copy the table contents back
       ; to an a-rate variable.
```

koff -- This is an offset which is added to the sum of *kstart* and the internal index variable which steps through the table. The result is then ANDed with the lengthmask (000 0111 for a table of length 8 - or 9 with guardpoint) and that final index is used to read or write to the table. *koff* can be any value. It is converted into a long using the ANSI floor() function so that -4.3 becomes -5. This is what we would want when using offsets which range above and below zero.

Ideally this would be an optional variable, defaulting to 0, however with the existing Csound orchestra read code, such default parameters must be init time only. We want k-rate here, so we cannot have a default.

See Also

tablewa

tableseg

tableseg — Creates a new function table by making linear segments between values in stored function tables.

Description

tableseg is like *linseg* but interpolate between values in a stored function tables. The result is a new function table passed internally to any following *vpvoc* which occurs before a subsequent *tableseg* (much like *lpread*/*lpreson* pairs work). The uses of these are described below under *vpvoc*.

Syntax

```
tableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

Initialization

ifn1, *ifn2*, *ifn3*, etc. -- function table numbers. *ifn1*, *ifn2*, and so on, must be the same size.

idur1, *idur2*, etc. -- durations during which interpolation from one table to the next will take place.

See Also

pvbufread, *pvcross*, *pvinterp*, *pvread*, *tablexseg*

Credits

Author: Richard Karpen
Seattle, Wash
1997

New in version 3.44

tablew

tablew — Change the contents of existing function tables.

Description

This opcode operates on existing function tables, changing their contents. *tablew* is for writing at k- or at a-rates, with the table number being specified at init time. The valid combinations of variable types are shown by the first letter of the variable names.

Syntax

```
tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

```
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

```
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

Initialization

asig, *isig, ksig* -- The value to be written into the table.

andx, *indx*, *kndx* -- Index into table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0)

ifn -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.

ixmode (optional, default=0) -- index mode.

- 0 = *xndx* and *ixoff* ranges match the length of the table.
- !=0 = *xndx* and *ixoff* have a 0 to 1 range.

ixoff (optional, default=0) -- index offset.

- 0 = Total index is controlled directly by *xndx*, i.e. the indexing starts from the start of the table.
- !=0 = Start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* != 0).

iwgmode (optional, default=0) -- Wrap and guardpoint mode.

- 0 = Limit mode.
- 1 = Wrap mode.
- 2 = Guardpoint mode.

Performance

Limit mode (0)

Limit the total index (*ndx* + *ixoff*) to between 0 and the guard point. For a table of length 5, this means that locations 0 to 3 and location 4 (the guard point) can be written. A negative total index writes to location 0.

Wrap mode (1)

Wrap total index value into locations 0 to E, where E is either one less than the table length or the factor of 2 number which is one less than the table length. For example, wrap into a 0 to 3 range - so that total index 6 writes to location 2.

Guardpoint mode (2)

The guardpoint is written at the same time as location 0 is written - with the same value.

This facilitates writing to tables which are intended to be read with interpolation for producing smooth cyclic waveforms. In addition, before it is used, the total index is incremented by half the range between one location and the next, before being rounded down to the integer address of a table location.

Normally (*igwmode* = 0 or 1) for a table of length 5 - which has locations 0 to 3 as the main table and location 4 as the guard point, a total index in the range of 0 to 0.999 will write to location 0. ("0.999" means just less than 1.0.) 1.0 to 1.999 will write to location 1 etc. A similar pattern holds for all total indexes 0 to 4.999 (*igwmode* = 0) or to 3.999 (*igwmode* = 1). *igwmode* = 0 enables locations 0 to 4 to be written - with the guardpoint (4) being written with a potentially different value from location 0.

With a table of length 5 and the *iwgmode* = 2, then when the total index is in the range 0 to 0.499, it will write to locations 0 and 4. Range 0.5 to 1.499 will write to location 1 etc. 3.5 to 4.0 will also write to locations 0 and 4.

This way, the writing operation most closely approximates the results of interpolated reading. Guard point mode should only be used with tables that have a guardpoint.

Guardpoint mode is accomplished by adding 0.5 to the total index, rounding to the next lowest integer, wrapping it modulo the factor of two which is one less than the table length, writing the table (locations 0 to 3 in our example) and then writing to the guard point if index = 0.

tablew has no output value. The last three parameters are optional and have default values of 0.

Caution with k-rate table numbers

At k-rate or a-rate, if a table number of < 1 is given, or the table number points to a non-existent table, or to one which has a length of 0 (it is to be loaded from a file later) then an error will result and the instrument will be deactivated. *kfn* and *afn* must be initialized at the appropriate rate using *init*. Attempting to load an i-rate value into *kfn* or *afn* will result in an error.

See Also

tableiw, *tablewkt*

Credits

Author: Robin Whittle
Australia
May 1997

tablewa

tablewa — Writes tables in sequential locations.

Description

This opcode writes to a table in sequential locations to and from an a-rate variable. Some thought is required before using it. It has at least two major, and quite different, applications which are discussed below.

Syntax

```
kstart tablewa kfn, asig, koff
```

Performance

kstart -- Where in table to read or write.

kfn -- i- or k-rate number of the table to read or write.

asig -- a-rate signal to read from when writing to the table.

koff -- i- or k-rate offset into table. Range unlimited - see explanation at end of this section.

In one application, it is intended to be used with one or with several *tablera* opcodes before a *tablewa* -- all sharing the same *kstart* variable.

These read from and write to sequential locations in a table at audio rates, with *ksmps* floats being written and read each cycle.

tablera starts reading from location *kstart*. *tablewa* starts writing to location *kstart*, and then writes to *kstart* with the number of the location one more than the one it last wrote. (Note that for *tablewa*, *kstart* is both an input and output variable.) If the writing index reaches the end of the table, then no further writing occurs and zero is written to *kstart*.

For instance, if the table's length was 16 (locations 0 to 15), and *ksmps* was 5. Then the following steps would occur with repetitive runs of the *tablewa* opcode, assuming that *kstart* started at 0.

Run Number	Initial kstart	Final kstart	Locations Written
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

This is to facilitate processing table data using standard a-rate orchestra code between the *tablera* and *tablewa* opcodes. They allow all Csound k-rate operators to be used (with caution) on a-rate variables - something that would only be possible otherwise by *ksmps* = 1, downsamp and upsamp.

Several cautions

- The k-rate code in the processing loop is really running at a-rate, so time dependent functions like *port* and *oscil* work faster than normal - their code is expecting to be running at k-rate.
- This system will produce undesirable results unless the *ksmps* fits within the table length. For instance a table of length 16 will accommodate 1 to 16 samples, so this example will work with *ksmps* = 1 to 16.

Both these opcodes generate an error and deactivate the instrument if a table with length < *ksmps* is selected. Likewise an error occurs if *kstart* is below 0 or greater than the highest entry in the table - if *kstart* = table length.

- *kstart* is intended to contain integer values between 0 and (table length - 1). Fractional values above this should not affect operation but do not achieve anything useful.
- These opcodes are not interpolating, and the *kstart* and *koff* parameters always have a range of 0 to (table length - 1) - not 0 to 1 as is available in other table read/write opcodes. *koff* can be outside this range but it is wrapped around by the final AND operation.
- These opcodes are permanently in wrap mode. When *koff* is 0, no wrapping needs to occur, since the *kstart*++ index will always be within the table's normal range. *koff* not equal to 0 can lead to wrapping.
- The offset does not affect the number of read/write cycles performed, or the value written to *kstart* by *tablewa*.
- These opcodes cannot read or write the guardpoint. Use *tablegpw* to write the guardpoint after manipulations have been done with *tablewa*.

Examples

```
kstart   =       0

lab1:
  atemp  tablera ktabsource, kstart, 0  ; Read 5 values from table into an
         ; a-rate variable.

  atemp  =       log(atemp)  ; Process the values using a-rate
         ; code.

  kstart tablewa ktabdest, atemp, 0   ; Write it back to the table

if ktemp  0 goto lab1      ; Loop until all table locations
         ; have been processed.
```

The above example shows a processing loop, which runs every k-cycle, reading each location in the table *ktabsource*, and writing the log of those values into the same locations of table *ktabdest*.

This enables whole tables, parts of tables (with offsets and different control loops) and data from several tables at once to be manipulated with a-rate code and written back to another (or to the same) table. This is a bit of a fudge, but it is faster than doing it with k-rate table read and write code.

Another application is:

```
kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0  ; ksmps a-rate samples written
       ; into locations 0 to (ksmps -1) of table 23.

lab1: ktemp table kloop, 23  ; Start a loop which runs ksmps times,
       ; in which each cycle processes one of
 [ Some code to manipulate ]  ; table 23's values with k-rate orchestra
 [ the value of ktemp. ]  ; code.

 tablew ktemp, kloop, 23  ; Write the processed value to the table.

kloop = kloop + 1      ; Increment the kloop, which is both the
       ; pointer into the table and the loop
if kloop < ksmps goto lab1  ; counter. Keep looping until all values
       ; in the table have been processed.

asignal   tablera 23, 0, 0  ; Copy the table contents back
       ; to an a-rate variable.
```

koff -- This is an offset which is added to the sum of *kstart* and the internal index variable which steps through the table. The result is then ANDed with the lengthmask (000 0111 for a table of length 8 - or 9 with guardpoint) and that final index is used to read or write to the table. *koff* can be any value. It is converted into a long using the ANSI floor() function so that -4.3 becomes -5. This is what we would want when using offsets which range above and below zero.

Ideally this would be an optional variable, defaulting to 0, however with the existing Csound orchestra read code, such default parameters must be init time only. We want k-rate here, so we cannot have a default.

Credits

Author: Robin Whittle
Australia

tablewkt

tablewkt — Change the contents of existing function tables.

Description

This opcode operates on existing function tables, changing their contents. *tablewkt* uses a k-rate variable for selecting the table number. The valid combinations of variable types are shown by the first letter of the variable names.

Syntax

```
tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwgmode]
```

```
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwgmode]
```

Initialization

asig, *ksig* -- The value to be written into the table.

andx, *kndx* -- Index into table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0)

kfn -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.

ixmode -- index mode. Default is zero.

- 0 = *xndx* and *ixoff* ranges match the length of the table.
- Not equal to 0 = *xndx* and *ixoff* have a 0 to 1 range.

ixoff -- index offset. Default is 0.

- 0 = Total index is controlled directly by *xndx*, i.e. the indexing starts from the start of the table.
- Not equal to 0 = Start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* != 0).

iwgmode -- table writing mode. Default is 0.

- 0 = Limit mode.
- 1 = Wrap mode.
- 2 = Guardpoint mode.

Performance

Limit mode (0)

Limit the total index (*ndx + ixoff*) to between 0 and the guard point. For a table of length 5, this means that locations 0 to 3 and location 4 (the guard point) can be written. A negative total index writes to location 0.

Wrap mode (1)

Wrap total index value into locations 0 to E, where E is one less than either the table length or the factor of 2 number which is one less than the table length. For example, wrap into a 0 to 3 range - so that total index 6 writes to location 2.

Guardpoint mode (2)

The guardpoint is written at the same time as location 0 is written - with the same value.

This facilitates writing to tables which are intended to be read with interpolation for producing smooth cyclic waveforms. In addition, before it is used, the total index is incremented by half the range between one location and the next, before being rounded down to the integer address of a table location.

Normally (*igwmode* = 0 or 1) for a table of length 5 - which has locations 0 to 3 as the main table and location 4 as the guard point, a total index in the range of 0 to 0.999 will write to location 0. ("0.999" means just less than 1.0.) 1.0 to 1.999 will write to location 1 etc. A similar pattern holds for all total indexes 0 to 4.999 (*igwmode* = 0) or to 3.999 (*igwmode* = 1). *igwmode* = 0 enables locations 0 to 4 to be written - with the guardpoint (4) being written with a potentially different value from location 0.

With a table of length 5 and the *iwgmode* = 2, then when the total index is in the range 0 to 0.499, it will write to locations 0 and 4. Range 0.5 to 1.499 will write to location 1 etc. 3.5 to 4.0 will also write to locations 0 and 4.

This way, the writing operation most closely approximates the results of interpolated reading. Guard point mode should only be used with tables that have a guardpoint.

Guardpoint mode is accomplished by adding 0.5 to the total index, rounding to the next lowest integer, wrapping it modulo the factor of two which is one less than the table length, writing the table (locations 0 to 3 in our example) and then writing to the guard point if index = 0.

Caution with k-rate table numbers

At k-rate or a-rate, if a table number of < 1 is given, or the table number points to a non-existent table, or to one which has a length of 0 (it is to be loaded from a file later) then an error will result and the instrument will be deactivated. *kfn* and *afn* must be initialized at the appropriate rate using *init*. Attempting to load an i-rate value into *kfn* or *afn* will result in an error.

See Also

tableiw, *tablew*

Credits

Author: Robin Whittle
Australia
May 1997

tablexkt

tablexkt — Reads function tables with linear, cubic, or sinc interpolation.

Description

Reads function tables with linear, cubic, or sinc interpolation.

Syntax

```
ares tablexkt xndx, kfn, kwarp, iwsize [, ixmode] [, ixoff] [, iwrap]
```

Initialization

iwsize -- This parameter controls the type of interpolation to be used:

- *2:* Use linear interpolation. This is the lowest quality, but also the fastest mode.
- *4:* Cubic interpolation. Slightly better quality than iwsize = 2, at the expense of being somewhat slower.
- *8 and above (up to 1024):* sinc interpolation with window size set to iwsize (should be an integer multiply of 4). Better quality than linear or cubic interpolation, but very slow. When transposing up, a kwarp value above 1 can be used for anti-aliasing (this is even slower).

ixmode1 (optional) -- index data mode. The default value is 0.

- *0:* raw index
- *any non-zero value:* normalized (0 to 1)

Notes

if *tablexkt* is used to play back samples with looping (e.g. table index is generated by lphasor), there must be at least iwsize / 2 extra samples after the loop end point for interpolation, otherwise audible clicking may occur (also, at least iwsize / 2 samples should be before the loop start point).

ixoff (optional) -- amount by which index is to be offset. For a table with origin at center, use tablesize / 2 (raw) or 0.5 (normalized). The default value is 0.

iwrap (optional) -- wraparound index flag. The default value is 0.

- *0:* Nowrap (index < 0 treated as index = 0; index >= tablesize (or 1.0 in normalized mode) sticks at the guard point).
- *any non-zero value:* Index is wrapped to the allowed range (not including the guard point in this case).

Note

iwrap also applies to extra samples for interpolation.

Performance

ares -- audio output

xndx -- table index

kfn -- function table number

kwarp -- if greater than 1, use sin (x / kwarp) / x function for sinc interpolation, instead of the default sin (x) / x. This is useful to avoid aliasing when transposing up (*kwarp* should be set to the transpose factor in this case, e.g. 2.0 for one octave), however it makes rendering up to twice as slow. Also, *iwsize* should be at least kwarp * 8. This feature is experimental, and may be optimized both in terms of speed and quality in new versions.

Note

kwarp has no effect if it is less than, or equal to 1, or linear or cubic interpolation is used.

Examples

Here is an example of the tablexkt opcode. It uses the file *tablexkt.csd* [examples/tablexkt.csd].

Exemple 497. Example of the tablexkt opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tablexkt.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Jonathan Murphy

  sr      =  44100
  ksmps      =  10
  nchnls     =  1

    instr 1

  ifn      =  1     ; query f1 as to number of samples
  ilen      =  nsamp(ifn)

  itrns      =  4    ; transpose up 4 octaves
  ilps      =  16  ; allow iwsize/2 samples at start
  ilpe      =  ilen - 16    ; and at end
  imode      =  3  ; loop forwards and backwards
  istrt      =  16  ; start 16 samples into loop

  alphs      lphasor    itrns, ilps, ilpe, imode, istrt
        ; use lphasor as index
  andx      =  alphs
```

```
  kfn      =  1   ; read f1
  kwarp     =  4 ; anti-aliasing, should be same value as itrns above
  iwsize    =  32  ; iwsize must be at least 8 * kwarp

  atab     tablexkt  andx, kfn, kwarp, iwsize

  atab     =  atab * 10000

            out       atab

    endin

</CsInstruments>
<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
i1 0 60
e
</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Istvan Varga
January 2002
Example by: Jonathan Murphy 2006

New in version 4.18

tablexseg

tablexseg — Creates a new function table by making exponential segments between values in stored function tables.

Description

tablexseg is like *expseg* but interpolate between values in a stored function tables. The result is a new function table passed internally to any following *vpvoc* which occurs before a subsequent *tablexseg* (much like *lpread*/*lpreson* pairs work). The uses of these are described below under *vpvoc*.

Syntax

```
tablexseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

Initialization

ifn1, *ifn2*, *ifn3*, etc. -- function table numbers. *ifn1*, *ifn2*, and so on, must be the same size.

idur1, *idur2*, etc. -- durations during which interpolation from one table to the next will take place.

See Also

pvbufread, *pvcross*, *pvinterp*, *pvread*, *tableseg*

Credits

Author: Richard Karpen
Seattle, WA USA
1997

tabmorph

tabmorph — Allow morphing between a set of tables.

Description

tabmorph allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

Syntax

```
kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

Initialization

ifn1, ifn2 [, ifn3, ifn4,…ifnN] - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *ktabnum1* and *ktabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

Performance

kout - The output value for index *kindex*, resulting from morphing two tables (see below).

kindex - main index index of the morphed resultant table. The range is 0 to table_length (not included).

kweightpoint - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

ktabnum1 - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

ktabnum2 - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorph* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (ifn2 [, ifn3, ifn4,…ifnN]). Then he can choose a pair of tables in the set in order to perform the morphing: *ktabnum1* and *ktabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *kweightpoint* parameter. After that the resulting table can be indexed with the *kindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

tabmorph acts similarly to the *table* opcode, that is, without using interpolation. This means that it trun-

cates the fractional part of the *kindex* argument. Anyway, fractional parts of *ktabnum1* and *ktabnum2* are significant, resulting in linear interpolation between the same element of two adjacent subsequent tables.

See Also

table, *tabmorphi*, *tabmorpha*, *tabmorphak*, *ftmorf*,

Credits

Author: Gabriel Maldonado

New in version 5.06

tabmorpha

tabmorpha — Allow morphing between a set of tables at audio rate with interpolation.

Description

tabmorpha allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

Syntax

```
aout tabmorpha aindex, aweightpoint, atabnum1, atabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

Initialization

ifn1, ifn2 , ifn3, ifn4, ... ifnN - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *atabnum1* and *atabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

Performance

aout - The output value for index *aindex*, resulting from morphing two tables (see below).

aindex - main index index of the morphed resultant table. The range is 0 to table_length (not included).

aweightpoint - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

atabnum1 - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

atabnum2 - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorpha* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (ifn2 [, ifn3, ifn4,…ifnN]). Then he can choose a pair of tables in the set in order to perform the morphing: *atabnum1* and *aatabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *aweightpoint* parameter. After that the resulting table can be indexed with the *aindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

tabmorpha is the audio-rate version of *tabmorphi* (it uses interpolation). All input arguments work at a-

rate.

See Also

table, *tabmorph*, *tabmorphi*, *tabmorphak*, *ftmorf*,

Credits

Author: Gabriel Maldonado

New in version 5.06

tabmorphak

tabmorphak — Allow morphing between a set of tables at audio rate with interpolation.

Description

tabmorphak allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

Syntax

```
aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

Initialization

ifn1, ifn2 , ifn3, ifn4, ... ifnN - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *atabnum1* and *atabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

Performance

aout - The output value for index *aindex*, resulting from morphing two tables (see below).

aindex - main index index of the morphed resultant table. The range is 0 to table_length (not included).

kweightpoint - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

ktabnum1 - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

ktabnum2 - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorphak* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (ifn2 [, ifn3, ifn4, ... ifnN]). Then he can choose a pair of tables in the set in order to perform the morphing: *atabnum1* and *atabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *aweightpoint* parameter. After that the resulting table can be indexed with the *aindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

tabmorphak works at a-rate, but kweightpoint, ktabnum1 and ktabnum2 are working at k-rate, making it

more efficient than *tabmorpha*, since there are less calculations. Except the rate of these three arguments, it is identical to *tabmorpha*.

See Also

table, *tabmorph*, *tabmorphi*, *tabmorpha*, *ftmorf*,

Credits

Author: Gabriel Maldonado

New in version 5.06

tabmorphi

tabmorphi — Allow morphing between a set of tables with interpolation.

Description

tabmorphi allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

Syntax

```
kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

Initialization

ifn1, ifn2 [, ifn3, ifn4,…ifnN] - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *ktabnum1* and *ktabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

Performance

kout - The output value for index *kindex*, resulting from morphing two tables (see below).

kindex - main index index of the morphed resultant table. The range is 0 to table_length (not included).

kweightpoint - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

ktabnum1 - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

ktabnum2 - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorphi* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (ifn2 [, ifn3, ifn4,…ifnN]). Then he can choose a pair of tables in the set in order to perform the morphing: *ktabnum1* and *ktabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *kweightpoint* parameter. After that the resulting table can be indexed with the *kindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

tabmorphi is identical to *tabmorph*, but it performs linear interpolation for non-integer values of *kindex*,

much like *tablei*.

See Also

table, *tabmorph*, *tabmorpha*, *tabmorphak*, *ftmorf*,

Credits

Author: Gabriel Maldonado

New in version 5.06

tabplay

tabplay — Playing-back control signals.

Description

Plays-back control-rate signals on trigger-temporization basis.

Syntax

```
tabplay  ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]
```

Performance

ktrig -- starts playing when non-zero.

knumtics -- stop recording or reset playing pointer to zero when the number of tics defined by this argument is reached.

kfn -- table where k-rate signals are recorded.

kout1,...,koutN -- playback output signals.

The *tabplay* and *tabrec* opcodes allow to record/playback control signals on trigger-temporization basis.

tabplay plays back a group of k-rate signals, previously recorded by tabrec into a table. Each time ktrig argument is triggered, an internal counter is increased of one unit. After knumtics trigger impluses are received by ktrig argument, the internal counter is zeroed and playback is restarted from the beginning, in looping style.

These opcodes can be used like a sort of ``middle-term'' memory that ``remembers'' generated signals. Such memory can be used to supply generative music with a coherent iterative compositional structure.

See Also

tabrec

Credits

Written by Gabriel Maldonado.

tambourine

tambourine — Modèle semi-physique d'un son de tambourin.

Description

tambourine est un modèle semi-physique d'un son de tambourin. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

Syntaxe

```
ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]
```

Initialisation

idettack -- période de temps durant laquelle tous les sons sont stoppés.

inum (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

idamp (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

damping_amount = 0,9985 + (idamp * 0,002)

La valeur par défaut de *damping_amount* est 0,9985 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,75.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

imaxshake (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

ifreq (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2300.

ifreq1 (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5600.

ifreq2 (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 8100.

Exécution

kamp -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

Exemples

Voici un exemple de l'opcode tambourine. Il utilise le fichier *tambourine.csd* [examples/tambourine.csd].

Exemple 498. Exemple de l'opcode tambourine.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tambourine.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1: An example of a tambourine.
instr 01
  a1 tambourine 15000, 0.01

  out a1
endin


</CsInstruments>
<CsScore>

i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

bamboo, *dripwater*, *guiro*, *sleighbells*

Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)
Adapté par John ffitch
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

tan

tan — Calcule une fonction tangente.

Description

Retourne tangente de *x* (*x* en radians).

Syntaxe

```
tan(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode tan. Il utilise le fichier *tan.csd* [examples/tan.csd].

Exemple 499. Exemple de l'opcode tan.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tan.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = tan(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = -0.134
```

Voir Aussi

cos, *cosh*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *taninv*

Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

tanh

tanh — Calcule une fonction tangente hyperbolique.

Description

Retourne tangente hyperbolique de *x* (*x* en radians).

Syntaxe

```
tanh(x) (pas de restriction de taux)
```

Exemples

Voici un exemple de l'opcode tanh. Il utilise le fichier *tanh.csd* [examples/tanh.csd].

Exemple 500. Exemple de l'opcode tanh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tanh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = tanh(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.762
```

Voir Aussi

cos, *cosh*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *taninv*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

taninv

taninv — Calcule une fonction arctangente.

Description

Retourne arctangente de *x* (*x* en radians).

Syntaxe

```
taninv(x) (pas de restriction de taux)
```

Exemples

Voici une exemple de l'opcode taninv. Il utilise le fichier *taninv.csd* [examples/taninv.csd].

Exemple 501. Exemple de l'opcode taninv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o taninv.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = taninv(irad)

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.464
```

Voir Aussi

cos, *cosh*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv2*

Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

taninv2

taninv2 — Retourne une tangente inverse (arctangente).

Description

Retourne arctangente de *iy/ix*, *ky/kx*, ou *ay/ax*.

Syntax

```
ares taninv2 ay, ax

ires taninv2 iy, ix

kres taninv2 ky, kx
```

Retourne arctangente de *iy/ix*, *ky/kx*, ou *ay/ax*. Si *y* vaut zéro, *taninv2* retourne zéro quelque soit la valeur de *x*. Si *x* vaut zéro, la valeur de retour est :

- *#/2*, si *y* est positif.
- *-#/2*, si *y* est négatif.
- *0*, si *y* vaut 0.

Initialisation

iy, ix -- valeurs à transformer

Exécution

ky, kx -- signaux de taux de contrôle à transformer

ay, ax -- signaux de taux audio à transformer

Exemples

Voici un exemple de l'opcode taninv2. Il utilise le fichier *taninv2.csd* [examples/taninv2.csd].

Exemple 502. Exemple de l'opcode taninv2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o taninv2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Returns the arctangent for 1/2.
  i1 taninv2 1, 2

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.464
```

Voir Aussi

taninv

Crédits

Auteur : John ffitch
Université de Bath/Codemist Ltd.
Bath, UK
Avril 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

Corrigé en mai 2002, grâce à Istvan Varga.

tbvcf

tbvcf — Models some of the filter characteristics of a Roland TB303 voltage-controlled filter.

Description

This opcode attempts to model some of the filter characteristics of a Roland TB303 voltage-controlled filter. Euler's method is used to approximate the system, rather than traditional filter methods. Cutoff frequency, Q, and distortion are all coupled. Empirical methods were used to try to unentwine, but frequency is only approximate as a result. Future fixes for some problems with this opcode may break existing orchestras relying on this version of *tbvcf*.

Syntax

```
ares tbvcf asig, xfco, xres, kdist, kasym [, iskip]
```

Initialization

iskip (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

Performance

asig -- input signal. Should be normalized to ±1.

xfco -- filter cutoff frequency. Optimum range is 10,000 to 1500. Values below 1000 may cause problems.

xres -- resonance or Q. Typically in the range 0 to 2.

kdist -- amount of distortion. Typical value is 2. Changing *kdist* significantly from 2 may cause odd interaction with *xfco* and *xres*.

kasym -- asymmetry of resonance. Typically in the range 0 to 1.

Examples

Here is an example of the tbvcf opcode. It uses the file *tbvcf.csd* [examples/tbvcf.csd].

Exemple 503. Example of the tbvcf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tbvcf.wav -W ;;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

;---------------------------------------------------------
; TBVCF Test
; Coded by Hans Mikelson December, 2000
;---------------------------------------------------------
  sr =  44100   ; Sample rate
  kr =  4410   ; Kontrol rate
  ksmps =  10   ; Samples/Kontrol period
  nchnls =  2        ; Normal stereo
  zakinit 50, 50

          instr 10

  idur    =          p3                                ; Duration
  iamp    =          p4                                ; Amplitude
  ifqc    =          cpspch(p5)                        ; Pitch to frequency
  ipanl   =          sqrt(p6)                          ; Pan left
  ipanr   =          sqrt(1-p6)                        ; Pan right
  iq      =          p7
  idist   =          p8
  iasym   =          p9

  kdclck  linseg 0, .002, 1, idur-.004, 1, .002, 0 ; Declick envelope

  kfco    expseg 10000, idur, 1000                 ; Frequency envelope

  ax      vco 1, ifqc, 2, 1                    ; Square wave
  ay      tbvcf ax, kfco, iq, idist, iasym       ; TB-VCF
  ay      buthp ay/1, 100                        ; Hi-pass

          outs ay*iamp*ipanl*kdclck, ay*iamp*ipanr*kdclck
          endin

</CsInstruments>
<CsScore>

f1 0 65536 10 1

; TeeBee Test
;    Sta  Dur  Amp    Pitch Pan  Q    Dist1 Asym
i10 0    0.2  32767  7.00  .5   0.0  2.0   0.0
i10 0.3  0.2  32767  7.00  .5   0.8  2.0   0.0
i10 0.6  0.2  32767  7.00  .5   1.6  2.0   0.0
i10 0.9  0.2  32767  7.00  .5   2.4  2.0   0.0
i10 1.2  0.2  32767  7.00  .5   3.2  2.0   0.0
i10 1.5  0.2  32767  7.00  .5   4.0  2.0   0.0
i10 1.8  0.2  32767  7.00  .5   0.0  2.0   0.25
i10 2.1  0.2  32767  7.00  .5   0.8  2.0   0.25
i10 2.4  0.2  32767  7.00  .5   1.6  2.0   0.25
i10 2.7  0.2  32767  7.00  .5   2.4  2.0   0.25
i10 3.0  0.2  32767  7.00  .5   3.2  2.0   0.25
i10 3.3  0.2  32767  7.00  .5   4.0  2.0   0.25
i10 3.6  0.2  32767  7.00  .5   0.0  2.0   0.5
i10 3.9  0.2  32767  7.00  .5   0.8  2.0   0.5
i10 4.2  0.2  32767  7.00  .5   1.6  2.0   0.5
i10 4.5  0.2  32767  7.00  .5   2.4  2.0   0.5
i10 4.8  0.2  32767  7.00  .5   3.2  2.0   0.5
i10 5.1  0.2  32767  7.00  .5   4.0  2.0   0.5
i10 5.4  0.2  32767  7.00  .5   0.0  2.0   0.75
i10 5.7  0.2  32767  7.00  .5   0.8  2.0   0.75
i10 6.0  0.2  32767  7.00  .5   1.6  2.0   0.75
i10 6.3  0.2  32767  7.00  .5   2.4  2.0   0.75
i10 6.6  0.2  32767  7.00  .5   3.2  2.0   0.75
i10 6.9  0.2  32767  7.00  .5   4.0  2.0   0.75
i10 7.2  0.2  32767  7.00  .5   0.0  2.0   1.0
i10 7.5  0.2  32767  7.00  .5   0.8  2.0   1.0
i10 7.8  0.2  32767  7.00  .5   1.6  2.0   1.0
i10 8.1  0.2  32767  7.00  .5   2.4  2.0   1.0
i10 8.4  0.2  32767  7.00  .5   3.2  2.0   1.0
i10 8.7  0.2  32767  7.00  .5   4.0  2.0   1.0
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Hans Mikelson
December, 2000 -- January, 2001

New in Csound 4.10

tempest

tempest — Estimate the tempo of beat patterns in a control signal.

Description

Estimate the tempo of beat patterns in a control signal.

Syntax

```
ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \
      istartempo, ifn [, idisprd] [, itweek]
```

Initialization

iprd -- period between analyses (in seconds). Typically about .02 seconds.

imindur -- minimum duration (in seconds) to serve as a unit of tempo. Typically about .2 seconds.

imemdur -- duration (in seconds) of the *kin* short-term memory buffer which will be scanned for periodic patterns. Typically about 3 seconds.

ihp -- half-power point (in Hz) of a low-pass filter used to smooth input *kin* prior to other processing. This will tend to suppress activity that moves much faster. Typically 2 Hz.

ithresh -- loudness threshold by which the low-passed *kin* is center-clipped before being placed in the short-term buffer as tempo-relevant data. Typically at the noise floor of the incoming data.

ihtim -- half-time (in seconds) of an internal forward-masking filter that masks new *kin* data in the presence of recent, louder data. Typically about .005 seconds.

ixfdbak -- proportion of this unit's *anticipated value* to be mixed with the incoming *kin* prior to all processing. Typically about .3.

istartempo -- initial tempo (in beats per minute). Typically 60.

ifn -- table number of a stored function (drawn left-to-right) by which the short-term memory data is attenuated over time.

idisprd (optional) -- if non-zero, display the short-term past and future buffers every *idisprd* seconds (normally a multiple of *iprd*). The default value is 0 (no display).

itweek (optional) -- fine-tune adjust this unit so that it is stable when analyzing events controlled by its own output. The default value is 1 (no change).

Performance

tempest examines *kin* for amplitude periodicity, and estimates a current tempo. The input is first low-pass filtered, then center-clipped, and the residue placed in a short-term memory buffer (attenuated over time) where it is analyzed for periodicity using a form of autocorrelation. The period, expressed as a *tempo* in beats per minute, is output as *ktemp*. The period is also used internally to make predictions about future amplitude patterns, and these are placed in a buffer adjacent to that of the input. The two adjacent buffers can be periodically displayed, and the predicted values optionally mixed with the inco-

ming signal to simulate expectation.

This unit is useful for sensing the metric implications of any k-signal (e.g.- the RMS of an audio signal, or the second derivative of a conducting gesture), before sending to a *tempo* statement.

Examples

Here is an example of the tempest opcode. It uses the file *tempest.csd* [examples/tempest.csd], and *beats.wav* [examples/beats.wav].

Exemple 504. Example of the tempest opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempest.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use the "beats.wav" sound file.
  asig soundin "beats.wav"
  ; Extract the pitch and the envelope.
  kcps, krms pitchamdf asig, 150, 500, 200

  iprd = 0.01
  imindur = 0.1
  imemdur = 3
  ihp = 1
  ithresh = 30
  ihtim = 0.005
  ixfdbak = 0.05
  istartempo = 110
  ifn = 1

  ; Estimate its tempo.
  k1 tempest krms, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, istartempo, ifn
  printk2 k1

  out asig
endin


</CsInstruments>
<CsScore>

; Table #1, a declining line.
f 1 0 128 16 1 128 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

The tempo of the audio file « beats.wav » is 120 beats per minute. In this examples, tempest will print out its best guess as the audio file plays. Its output should include lines like this:

```
. i1   118.24654
. i1   121.72949
```

tempo

tempo — Apply tempo control to an uninterpreted score.

Description

Apply tempo control to an uninterpreted score.

Syntax

```
tempo ktempo, istartempo
```

Initialization

istartempo -- initial tempo (in beats per minute). Typically 60.

Performance

ktempo -- The tempo to which the score will be adjusted.

tempo allows the performance speed of Csound scored events to be controlled from within an orchestra. It operates only in the presence of the Csound *-t* flag. When that flag is set, scored events will be performed from their uninterpreted p2 and p3 (beat) parameters, initially at the given command-line tempo. When a *tempo* statement is activated in any instrument (*ktempo* 0.), the operating tempo will be adjusted to *ktempo* beats per minute. There may be any number of *tempo* statements in an orchestra, but coincident activation is best avoided.

Examples

Here is an example of the tempo opcode. Remember, it only works if you use the *-t* flag with Csound. The example uses the file *tempo.csd* [examples/tempo.csd].

Exemple 505. Example of the tempo opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempo.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
  kval tempoval

  printk 0.1, kval

  ; If the fourth p-field is 1, increase the tempo.
  if (p4 == 1) kgoto speedup
    kgoto playit

speedup:
  ; Increase the tempo to 150 beats per minute.
  tempo 150, 60

playit:

  a1 oscil 10000, 440, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = plays at a faster tempo (when p4=1).
; Play Instrument #1 at the normal tempo, repeat 3 times.
r3
i 1 00.00 00.25 0
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

; Play Instrument #1 at a faster tempo, repeat 3 times.
r3
i 1 00.00 00.25 1
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

e

</CsScore>
</CsoundSynthesizer>
```

See Also

tempoval

Credits

Example written by Kevin Conder.

tempoval

tempoval — Reads the current value of the tempo.

Description

Reads the current value of the tempo.

Syntax

```
kres tempoval
```

Performance

kres -- the value of the tempo. If you use a positive value with the *-t command-line flag*, *tempoval* returns the percentage increase/decrease from the original tempo of 60 beats per minute. If you don't, its value will be 60 (for 60 beats per minute).

Examples

Here is an example of the tempoval opcode. Remember, it only works if you use the *-t* flag with Csound. It uses the file *tempoval.csd* [examples/tempoval.csd].

Exemple 506. Example of the tempoval opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempoval.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Adjust the tempo to 120 beats per minute.
  tempo 120, 60

  ; Get the tempo value.
  kval tempoval

  printks "kval = %f\\n", 0.1, kval
endin


</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Since 120 beats per minute is a 50% increase over the original 60 beats per minute, its output should include lines like:

```
kval = 0.500000
```

See Also

tempo and *miditempo*

Credits

Example written by Kevin Conder.

New in version 4.15

December 2002. Thanks to Drake Wilson for pointing out unclear documentation.

tigoto

tigoto — Transfer control at i-time when a new note is being tied onto a previously held note

Description

Similar to *igoto* but effective only during an i-time pass at which a new note is being « tied » onto a previously held note. (See *i Statement*) It does not work when a tie has not taken place. Allows an instrument to skip initialization of units according to whether a proposed tie was in fact successful. (See also *tival*, *delay*).

Syntax

```
tigoto label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

See Also

cigoto, *goto*, *if*, *igoto*, *kgoto*, *timout*

Credits

Added a note by Jim Aikin.

timedseq

timedseq — Time Variant Sequencer

Description

An event-sequencer in which time can be controlled by a time-pointer. Sequence data are stored into a table.

Syntax

```
ktrig  timedseq  ktimpnt, ifn, kp1 [,kp2, kp3, ...,kpN]
```

Initialization

ifn -- number of table containing sequence data.

Performance

ktri -- output trigger signal

ktimpnt -- time pointer into sequence file, in seconds.

kp1,...,kpN -- output p-fields of notes. kp2 meaning is relative action time and kp3 is the duration of notes in seconds.

timedseq is a sequencer that allows to schedule notes starting from a user sequence, and depending from an external timing given by a time-pointer value (ktimpnt argument). User should fill table ifn with a list of notes, that can be provided in an external text file by using GEN23, or by typing it directly in the orchestra (or score) file with GEN02. The format of the text file containing the sequence is made up simply by rows containing several numbers separated by space (similarly to normal Csound score). The first value of each row must be a positve or null value, except for a special case that will be explained below. This first value is normally used to define the instrument number corresponding to that particular note (like normal score). The second value of each row must contain the action time of corresponding note and the third value its duration. This is an example:

```
0 0    0.25 1  93
0 0.25 0.25 2  63
0 0.5  0.25 3  91
0 0.75 0.25 4  70
0 1    0.25 5  83
0 1.25 0.25 6  75
0 1.5  0.25 7  78
0 1.75 0.25 8  78
0 2    0.25 9  83
0 2.25 0.25 10 70
0 2.5  0.25 11 54
0 2.75 0.25 12 80
-1 3   -1   -1 -1  ;; last row of the sequence
```

In this example, the first value of each row is always zero (it is a dummy value, but this p-field can be used, for example, to express a MIDI channel or an instrument number), except the last row, that begins with -1. This value (-1) is a special value, that indicates the end of sequence. It has itself an action time, because sequnces can be looped. So the previous sequence has a default duration of 3 seconds, being value 3 the last action time of the sequence.

It is important that ALL lines contains the same number of values (in the example all rows contains exactly 5 values). The number of values contained by each row, MUST be the number of kpXX output arguments (notice that, even if kp1, kp2 etc. are placed at the right of the opcode, they are output arguments, not input arguments).

ktimpnt argument provide the real temporization of the sequence. Actually the passage of time through sequence is specified by ktimpnt itself, which represents the time in seconds. ktimpnt must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the sequence file, in the same way of pvoc or lpread. When ktimpnt crosses the action time of a note, a trigger signal is sent to ktrig output argument, and kp1, kp2,...kpN arguments are updated with the values of that note. This information can then be used with schedk or schedkwhen to actually activate note events. Notice that kp1,...kpn data can be further processed (for example delayed with delayk, transposed, etc.) before feeding schedk or schedkwhen.

ktimepoint can be controlled by linear signal, for example:

```
ktimpnt line     0,p3,3  ; orignal sequence duration was 3 secs
ktrig   timedseq ktimpnt,1,kp1,kp2,kp3,kp4,kp5
        schedk   ktrig, 105, 2, 0, kp3,kp4,kp5
```

in this case the complete sequence (with orginal duration of 3 seconds) will be played in p3 seconds.

You can loop a sequence by contolling it with a phasor:

```
kphs    phasor   1/3
ktimpnt =        kphs * 3
ktrig   timedseq ktimpnt,1,kp1,kp2,kp3,kp4,kp5
        schedk   ktrig, 105, 2, 0, kp3,kp4,kp5
```

Obviously you can play only a fragment of the sequence, read it backward, and non-linearly access sequence data in the same way of pvoc and lpread opcodes.

With timedseq opcode you can do almost all things of a normal score, except you have the following limitations: 1. You can't have two notes exactly starting with the same action time; actually at least a k-cycle should separate timing of two notes (otherwise the schedk mechanism eats one of them). 2. all notes of the sequence must have the same number of p-fields (even if they activate different instruments). You can remedy this limitation by filling with dummy values notes that belongs to instruments with less p-fields than other ones.

See Also

GEN02, *GEN23*, *seqtime*, *seqtime2*, *trigseq*

Credits

Author: Gabriel Maldonado

timeinstk

timeinstk — Read absolute time in k-rate cycles.

Description

Read absolute time, in k-rate cycles, since the start of an instance of an instrument. Called at both i-time as well as k-time.

Syntax

```
kres timeinstk
```

```
kres timeinsts
```

Performance

timeinstk is for time in k-rate cycles. So with:

```
  sr    = 44100
  kr    = 6300
  ksmps = 7
```

then after half a second, the *timek* opcode would report 3150. It will always report an integer.

timeinstk produces a k-rate variable for output. There are no input parameters.

timeinstk is similar to *timek* except it returns the time since the start of this instance of the instrument.

Examples

Here is an example of the timeinstk opcode. It uses the file *timeinstk.csd* [examples/timeinstk.csd].

Exemple 507. Example of the timeinstk opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o timeinstk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the value from timeinstk every half-second.
  k1 timeinstk
  printks "k1 = %f samples\\n", 0.5, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples
```

See Also

timeinsts, *timek*, *times*

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

timeinsts

timeinsts — Read absolute time in seconds.

Description

Read absolute time, in seconds, since the start of an instance of an instrument.

Syntax

```
kres timeinsts
```

Performance

Time in seconds is available with *timeinsts*. This would return 0.5 after half a second.

timeinsts produces a k-rate variable for output. There are no input parameters.

timeinsts is similar to *times* except it returns the time since the start of this instance of the instrument.

Examples

Here is an example of the timeinsts opcode. It uses the file *timeinsts.csd* [examples/timeinsts.csd].

Exemple 508. Example of the timeinsts opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o timeinsts.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the value from timeinsts every half-second.
  k1 timeinsts
  printks "k1 = %f seconds\\n", 0.5, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
k1 = 0.000227 seconds
k1 = 0.500000 seconds
k1 = 1.000000 seconds
k1 = 1.500000 seconds
k1 = 2.000000 seconds
```

See Also

timeinstk, *timek*, *times*

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

timek

timek — Read absolute time in k-rate cycles.

Description

Read absolute time, in k-rate cycles, since the start of the performance.

Syntax

```
ires timek
```

```
kres timek
```

Performance

timek is for time in k-rate cycles. So with:

```
  sr    = 44100
  kr    = 6300
  ksmps = 7
```

then after half a second, the *timek* opcode would report 3150. It will always report an integer.

timek can produce a k-rate variable for output. There are no input parameters.

timek can also operate only at the start of the instance of the instrument. It produces an i-rate variable (starting with *i* or *gi*) as its output.

Examples

Here is an example of the timek opcode. It uses the file *timek.csd* [examples/timek.csd].

Exemple 509. Example of the timek opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o timek.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the value from timek every half-second.
  k1 timek
  printks "k1 = %f samples\\n", 0.5, k1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples
```

See Also

timeinstk, *timensts*, *times*

Credits

Author: Robin Whittle
Australia
May 1997

New in version 3.47

Example written by Kevin Conder.

times

times — Read absolute time in seconds.

Description

Read absolute time, in seconds, since the start of the performance.

Syntax

```
ires times
```

```
kres times
```

Performance

Time in seconds is available with *times*. This would return 0.5 after half a second.

times can both produce a k-rate variable for output. There are no input parameters.

times can also operate at the start of the instance of the instrument. It produces an i-rate variable (starting with *i* or *gi*) as its output.

Examples

Here is an example of the times opcode. It uses the file *times.csd* [examples/times.csd].

Exemple 510. Example of the times opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o times.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print out the value from times every half-second.
  k1 times
  printks "k1 = %f seconds\\n", 0.5, k1
endin


</CsInstruments>
```

```
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
k1 = 0.000227 seconds
k1 = 0.500000 seconds
k1 = 1.000000 seconds
k1 = 1.500000 seconds
k1 = 2.000000 seconds
```

See Also

timeinstk, *timeinsts*, *timek*

Credits

Author: Robin Whittle
Australia
May 1997

Example written by Kevin Conder.

timout

timout — Conditional branch during p-time depending on elapsed note time.

Description

Conditional branch during p-time depending on elapsed note time. *istrt* and *idur* specify time in seconds. The branch to *label* will become effective at time *istrt*, and will remain so for just *idur* seconds. Note that *timout* can be reinitialized for multiple activation within a single note (see example under *reinit*).

Syntax

```
timout istrt, idur, label
```

where *label* is in the same instrument block and is not an expression, and where *R* is one of the Relational operators (<, =, <=, ==, !=) (and = for convenience, see also under *Conditional Values*).

See Also

goto, *if*, *igoto*, *kgoto*, *tigoto*

Credits

Added a note by Jim Aikin.

tival

tival — Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

Syntaxe

```
ir tival
```

Description

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

Initialisation

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i. Affecte 1 si la note est « liée » à une note tenue précédente (voir l'*instruction i*) ; affecte 0 s'il n'y a pas de liaison. (Voir aussi *tigoto*.)

Voir Aussi

=, *divz*, *init*

tlineto

tlineto — Generate glissandos starting from a control signal.

Description

Generate glissandos starting from a control signal with a trigger.

Syntax

```
kres tlineto ksig, ktime, ktrig
```

Performance

kres -- Output signal.

ksig -- Input signal.

ktime -- Time length of glissando in seconds.

ktrig -- Trigger signal.

tlineto is similar to *lineto* but can be applied to any kind of signal (not only stepped signals) without producing discontinuities. Last value of each segment is sampled and held from input signal each time *ktrig* value is set to a nonzero value. Normally *ktrig* signal consists of a sequence of zeroes (see *trigger opcode*).

The effect of glissando is quite different from *port*. Since in these cases, the lines are straight. Also the context of useage is different.

See Also

lineto

Credits

Author: Gabriel Maldonado

New in Version 4.13

tone

tone — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

tone est filtre RII à un terme. Sa formule est :

$y_n = c1 * x_n + c2 * y_{n-1}$

où

- b = 2 - cos(2 # hp/sr);
- $c2 = b - sqrt(b^2 - 1.0)$
- c1 = 1 - c2

Syntaxe

```
ares tone asig, khp [, iskip]
```

Initialisation

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

ares -- le signal audio de sortie.

asig -- le signal audio en entrée.

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

tone implémente un filtre passe-bas récursif du premier ordre dans lequel la variable *khp* (en Hz) détermine le point à mi-puissance de la courbe de réponse. La mi-puissance est définie par puissance maximale / racine de 2.

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tonek*

tonek

tonek — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

Syntaxe

```
kres tonek ksig, khp [, iskip]
```

Initialisation

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

kres -- le signal de sortie au taux de contrôle.

ksig -- le signal d'entrée au taux de contrôle.

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

tonek est semblabe à *tone* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

Voir Aussi

areson, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

tonex

tonex — Emule une série de filtres utilisant l'opcode *tone*.

Description

tonex est équivalent à un filtre constitué de plusieurs couches de filtres *tone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de k cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

Syntaxe

```
ares tonex  asig, khp [, inumlayer] [, iskip]
```

Initialisation

inumlayer (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

iskip (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

Exécution

asig -- signal d'entrée

khp -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

Voir Aussi

atonex, *resonx*

Crédits

Auteur : Gabriel Maldonado (adapté par John ffitch)
Italie

Nouveau dans la version 3.49 de Csound

trandom

trandom — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale en fonction d'un déclencheur.

Description

Génère au taux-k une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale chaque fois que le paramètre de déclenchement est différent de 0.

Syntaxe

```
kout trandom ktrig, kmin, kmax
```

Exécution

ktrig -- déclencheur (l'opcode produit un nouveau nombre aléatoire chaque fois que cette valeur est différente de 0.

kmin -- limite inférieure de l'intervalle

kmax -- limite supérieure de l'intervalle

trandom est presque identique à l'opcode *random* sauf que *trandom* ne renouvelle sa sortie avec une nouvelle valeur aléatoire que si l'argument *ktrig* est déclenché (c-à-d chaque fois qu'il est différent de zéro).

Voir Aussi

random

Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5.06

tradsyn

tradsyn — Streaming partial track additive synthesis

Description

The tradsyn opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials),as described in Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. It resynthesises the signal using linear amplitude and frequency interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls.

Syntax

```
asig tradsyn fin, kscal, kpitch, kmaxtracks, ifn
```

Performance

asig -- output audio rate signal

fin -- input pv stream in TRACKS format

kscal -- amplitude scaling

kpitch -- pitch scaling

kmaxtracks -- max number of tracks in resynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

ifn -- function table containing one cycle of a sinusoid (sine or cosine)

Examples

Exemple 511. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
      aout  tradsyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with pitch shifting.

Credits

Author: Victor Lazzarini;
June 2005

New plugin in version 5

November 2004.

transeg

transeg — Construit une enveloppe définie par l'utilisateur.

Description

Construit une enveloppe définie par l'utilisateur.

Syntaxe

```
ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

```
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

Initialisation

ia -- valeur de départ.

ib, ic, etc. -- valeur après *idur* secondes.

idur -- durée en secondes du premier segment. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

idur2,...idurx etc. -- durée en secondes de chaque segment.

itype, itype2, etc. -- s'il vaut 0, un segment de droite est produit. S'il est différent de 0, *transeg* crée la courbe suivante en *n* pas :

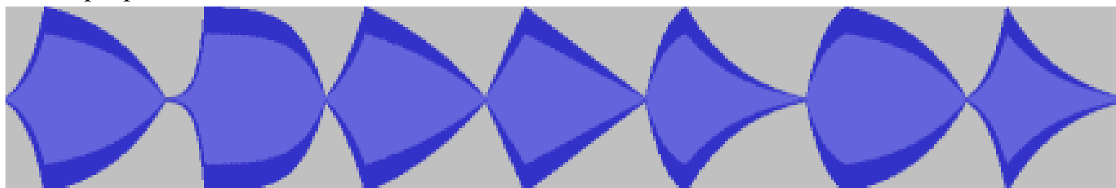
ibeg + (ivalue - ibeg) * (1 - exp(i*itype/(n-1))) / (1 - exp(itype))

Exécution

Si *type* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *type* < 0, la courbe monte rapidement (convexe) ou décroit rapidement (concave). Voir aussi *GEN16*.

Examples

Voici un exemple de l'opdcode transeg. Il utilise le fichier *transeg.csd* [examples/transeg.csd]. L'exemple produit la sortie suivante :



Sortie de l'exemple de transeg.

Exemple 512. Exemple de l'opdcode transeg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o transeg.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

0dbfs = 1

instr 1
;p4 and p5 determine the type of curve for each
;section of the envelope
kenv transeg 0.01, p3*0.25, p4, 1, p3*0.75, p5, 0.01
a1 oscil kenv, 440, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>
; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 2 2 2
i 1 + . 5 5
i 1 + . 1 1
i 1 + . 0 0
i 1 + . -2 -2
i 1 + . -2 2
i 1 + . 2 -2
e
</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

expsega, *expsegr*, *linseg*, *linsegr*

Crédits

Auteur : John ffitch
Université de Bath, Codemist. Ltd.
Bath, UK
Octobre 2000

Nouveau dans la version 4.09 de Csound

Merci à Matt Gerassimoff pour avoir précisé la syntaxe correcte de la commande.

trcross

trcross — Streaming partial track cross-synthesis.

Description

The trcross opcode takes two inputs containg TRACKS pv streaming signals (as generated, for instance by partials) and cross-synthesises them into a single TRACKS stream. Two different modes of operation are used: mode 0, cross-synthesis by multiplication of the amplitudes of the two inputs and mode 1, cross-synthesis by the substititution of the amplitudes of input 1 by the input 2. Frequencies and phases of input 1 are preserved in the output. The cross-synthesis is done by matching tracks between the two inputs using a 'search interval'. The matching algorithm will look for tracks in the second input that are within the search interval around each track in the first input. This interval can be changed at the control rate. Wider search intervals will find more matches.

Syntax

```
fsig trcross fin1, fin2, ksearch,kdepth[,kmode]
```

Performance

fsig -- output pv stream in TRACKS format

fin1 -- first input pv stream in TRACKS format.

fin2 -- second input pv stream in TRACKS format

ksearch -- search interval ratio, defining a 'search area' around each track of 1st input for matching purposes.

kdepth -- depth of effect (0-1).

kmode -- mode of cross-synthesis. 0, multiplication of amplitudes (filtering), 1, subsitution of amplitudes of input 1 by input 2 (akin to vocoding). Defaults to 0.

Examples

Exemple 513. Example

```
ain inch 1                  ; input signals
ain inch 2
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fs11,fsi12 pvsifd ain,2048,512,1 ; ifd analysis (second input)
fst1  partials fs11,fsi12,.003,1,3,500  ; partial tracking \(second input
fcr trcross fst,fst1, 1.05, 1 ; cross-synthesis (mode 0)
      aout  tradsyn fcr, 1, 1, 500, 1 ; resynthesis of tracks
out aout
```

The example above shows partial tracking of two ifd-analysis signals, cross-synthesis, followed by the

remix of the two parts of the spectrum and resynthesis.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trfilter

trfilter — Streaming partial track filtering.

Description

The trfilter opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials) and filters it using an amplitude response curve stored in a function table. The function table can have any size (no restriction to powers-of-two). The table lookup is done by linear-interpolation. It is possible to create time-varying filter curves by updating the amlitude response table with a table-writing opcode.

Syntax

```
fsig trfilter fin, kamnt, ifn
```

Performance

fsig -- output pv stream in TRACKS format

fin -- input pv stream in TRACKS format

kamnt -- amount of filtering (0-1)

ifn -- function table number. This will contain an amplitude response curve, from 0 Hz to the Nyquist (table indexes 0 to N). Any size is allowed. Larger tables will provide a smoother amplitude response curve. Table reading uses linear interpolation.

Examples

Exemple 514. Example

```
gifn ftgen 2, 0, -22050, 5 1 1000 1 4000 0.000001 17050 0.000001 ; low-pass filter curve of 22050 point
instr 1
ain inch 1                  ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fscl trfilter  fst, 1, gifn ; filtering using function table 2
      aout  tradsyn fscl, 1, 1, 500, 1 ; resynthesis
out aout
endin
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with low-pass filtering.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trhighest

trhighest — Extracts the highest-frequency track from a streaming track input signal.

Description

The trhighest opcode takes an input containg TRACKS pv streaming signals (as generated, for instance by partials) and outputs only the highest track. In addition it outputs two k-rate signals, corresponding to the frequency and amplitude of the highest track signal.

Syntax

```
fsig, kfr,kamp trhighest fin1, kscal
```

Performance

fsig -- output pv stream in TRACKS format

kfr -- frequency (in Hz) of the highest-frequency track

kamp -- amplitude of the highest-frequency track

fin -- input pv stream in TRACKS format.

kscal -- amplitude scaling of output.

Examples

Exemple 515. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fhi,kfr,kamp trhighest fst,1 ; highest freq-track
      aout  tradsyn fhi, 1, 1, 1, 1 ; resynthesis of highest frequency
out aout
```

The example above shows partial tracking of an ifd-analysis signal, extraction of the highest frequency and resynthesis.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trigger

trigger — Informs when a krate signal crosses a threshold.

Description

Informs when a krate signal crosses a threshold.

Syntax

```
kout trigger ksig, kthreshold, kmode
```

Performance

ksig -- input signal

kthreshold -- trigger threshold

kmode -- can be 0 , 1 or 2

Normally *trigger* outputs zeroes: only each time *ksig* crosses *kthreshold* *trigger* outputs a 1. There are three modes of using *ktrig*:

- *kmode* = 0 - (down-up) *ktrig* outputs a 1 when current value of *ksig* is higher than *kthreshold,* while old value of *ksig* was equal to or lower than *kthreshold*.
- *kmode* = 1 - (up-down) *ktrig* outputs a 1 when current value of *ksig* is lower than *kthreshold* while old value of *ksig* was equal or higher than *kthreshold*.
- *kmode* = 2 - (both) *ktrig* outputs a 1 in both the two previous cases.

Examples

Here is an example of the trigger opcode. It uses the file *trigger.csd* [examples/trigger.csd].

Exemple 516. Example of the trigger opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o trigger.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a square-wave low frequency oscillator as the trigger.
  klf lfo 1, 10, 3
  ktr trigger klf, 1, 2

  ; When the value of the trigger isn't equal to 0, print it out.
  if (ktr == 0) kgoto contin
    ; Print the value of the trigger and the time it occurred.
    ktm times
    printks "time = %f seconds, trigger = %f\\n", 0, ktm, ktr

contin:
  ; Continue with processing.
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
time = 0.050340 seconds, trigger = 1.000000
time = 0.150340 seconds, trigger = 1.000000
time = 0.250340 seconds, trigger = 1.000000
time = 0.350340 seconds, trigger = 1.000000
time = 0.450340 seconds, trigger = 1.000000
time = 0.550340 seconds, trigger = 1.000000
time = 0.650340 seconds, trigger = 1.000000
time = 0.750340 seconds, trigger = 1.000000
time = 0.850340 seconds, trigger = 1.000000
time = 0.950340 seconds, trigger = 1.000000
```

Credits

Author: Gabriel Maldonado
Italy

Example written by Kevin Conder.

New in Csound version 3.49

trigseq

trigseq — Accepts a trigger signal as input and outputs a group of values.

Description

Accepts a trigger signal as input and outputs a group of values.

Syntax

```
trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]
```

Performance

ktrig_in -- input trigger signal

kstart -- start index of looped section

kloop -- end index of looped section

kinitndx -- initial index

Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

kfn_values -- numer of a table containing a sequence of groups of values

kout1 -- output values

kout2, ... (optional) -- more output values

This opcode handles timed-sequences of groups of values stored into a table.

trigseq accepts a trigger signal (*ktrig_in*) as input and outputs group of values (contained in the *kfn_values* table) each time *ktrig_in* assumes a non-zero value. Each time a group of values is triggered, table pointer is advanced of a number of positions corresponding to the number of group-elements, in order to point to the next group of values. The number of elements of groups is determined by the number of *koutX* arguments.

It is possible to start the sequence from a value different than the first, by assigning to initndx an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *kinitndx*) correspond to valid table numbers, otherwise Csound will crash because no range-checking is implemented.

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* arguments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value.

trigseq is designed to be used together with *seqtime* or *trigger* opcodes.

See Also

seqtime, *trigger*

Credits

Author: Gabriel Maldonado

November 2002. Added a note about the *kinitndx* parameter, thanks to Rasmus Ekman.

January 2003. Thanks to a note from Oeyvind Brandtsegg, I corrected the credits.

New in version 4.06

trirand

trirand — Générateur de nombres aléatoires de distribution triangulaire.

Description

Générateur de nombres aléatoires de distribution triangulaire. C'est un générateur de bruit de classe x.

Syntaxe

```
ares trirand krange
```

```
ires trirand krange
```

```
kres trirand krange
```

Exécution

krange -- l'intervalle des nombres aléatoires (-*krange* à +*krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode trirand. Il utilise le fichier *trirand.csd* [examples/trirand.csd].

Exemple 517. Exemple de l'opcode trirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o trirand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Generate a random number between -1 and 1.
  ; krange = 1

  i1 trirand 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 7506.261
```

Voir Aussi

betarand, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *unirand*, *weibull*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

trlowest

trlowest — Extracts the lowest-frequency track from a streaming track input signal.

Description

The trlowest opcode takes an input containg TRACKS pv streaming signals (as generated, for instance by partials) and outputs only the lowest track. In addition it outputs two k-rate signals, corresponding to the frequency and amplitude of the lowest track signal.

Syntax

```
fsig, kfr,kamp trlowest fin1, kscal
```

Performance

fsig -- output pv stream in TRACKS format

kfr -- frequency (in Hz) of the lowest-frequency track

kamp -- amplitude of the lowest-frequency track

fin -- input pv stream in TRACKS format.

kscal -- amplitude scaling of output.

Examples

Exemple 518. Example

```
ain inch 1                  ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
flow,kfr,kamp trlowest fst,1 ; lowest freq-track
      aout  tradsyn flow, 1, 1, 1, 1 ; resynthesis of lowest frequency
out aout
```

The example above shows partial tracking of an ifd-analysis signal, extraction of the lowest frequency and resynthesis.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trmix

trmix — Streaming partial track mixing.

Description

The trmix opcode takes two inputs containg TRACKS pv streaming signals (as generated, for instance by partials) and mixes them into a single TRACKS stream. Tracks will be mixed up to the available space (defined by the original number of FFT bins in the analysed signals). If the sum of the input tracks exceeds this space, the higher-ordered tracks in the second input will be pruned.

Syntax

```
fsig trmix fin1, fin2
```

Performance

fsig -- output pv stream in TRACKS format

fin1 -- first input pv stream in TRACKS format.

fin2 -- second input pv stream in TRACKS format

Examples

Exemple 519. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fslo,fshi trsplit  fst, 1500 ; split partial tracks at 1500 Hz
fscl trscale fshi, 1.15  ; shift the upper tracks
fmix trmix fslo,fscl ; mix the shifted and unshifted tracks
      aout  tradsyn fmix, 1, 1, 500, 1 ; resynthesis of tracks
out aout
```

The example above shows partial tracking of an ifd-analysis signal, frequency splitting and pitch shifting of the upper part of the spectrum, followed by the remix of the two parts of the spectrum and resynthesis.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trscale

trscale — Streaming partial track frequency scaling.

Description

The trscale opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials) and scales all frequencies by a k-rate amount. It can also, optionally, scale the gain of the signal by a k-rate amount (default 1). The result is pitch shifting of the input tracks.

Syntax

```
fsig trscale fin, kpitch[, kgain]
```

Performance

fsig -- output pv stream in TRACKS format

fin -- input pv stream in TRACKS format

kpitch -- frequency scaling

kgain -- amplitude scaling (default 1)

Examples

Exemple 520. Example

```
ain inch 1                    ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fscl trscale  fst, 1.5 ; frequency scale (up a 5th)
      aout  tradsyn fscl, 1, 1, 500, 1 ; resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with pitch shifting.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trshift

trshift — Streaming partial track frequency scaling.

Description

The trshift opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials) and shifts all frequencies by a k-rate frequency. It can also, optionally, scale the gain of the signal by a k-rate amount (default 1). The result is frequency shifting of the input tracks.

Syntax

```
fsig trshift fin, kpshift[, kgain]
```

Performance

fsig -- output pv stream in TRACKS format

fin -- input pv stream in TRACKS format

kshift -- frequency shift in Hz

kgain -- amplitude scaling (default 1)

Examples

Exemple 521. Example

```
ain inch 1                    ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fscl trshift  fst, 150 ; frequency shift (adds 150Hz to all tracks)
      aout  tradsyn fscl, 1, 1, 500, 1 ; resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with frequency shifting.

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

trsplit

trsplit — Streaming partial track frequency splitting.

Description

The trsplit opcode takes an input containg a TRACKS pv streaming signal (as generated, for instance by partials) and splits it into two signals according to a k-rate frequency 'split point'. The first output will contain all tracks up from 0Hz to the split frequency and the second will contain the tracks from the split frequency up to the Nyquist. It can also, optionally, scale the gain of the output signals by a k-rate amount (default 1). The result is two output signals containing only part of the original spectrum.

Syntax

```
fsiglow, fsighi trsplit fin, ksplit[, kgainlow, kgainhigh]
```

Performance

fsiglow -- output pv stream in TRACKS format containing the tracks below the split point.

fsighi -- output pv stream in TRACKS format containing the tracks above and including the split point.

fin -- input pv stream in TRACKS format

ksplit -- frequency split point in Hz

kgainlow, kgainhig -- amplitude scaling of each one of the outputs (default 1).

Examples

Exemple 522. Example

```
ain inch 1                   ; input signal
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst  partials fs1,fsi2,.003,1,3,500  ; partial tracking
fslo,fshi trsplit  fst, 1500 ; split partial tracks at 1500 Hz
      aout  tradsyn fshi, 1, 1, 500, 1 ; resynthesis of tracks above 1500Hz
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis of the upper part of the spectrum (from 1500Hz).

Credits

Author: Victor Lazzarini;
February 2006

New in Csound5.01

turnoff

turnoff — Enables an instrument to turn itself off.

Description

Enables an instrument to turn itself off.

Syntax

```
turnoff
```

Performance

turnoff -- this p-time statement enables an instrument to turn itself off. Whether of finite duration or « held », the note currently being performed by this instrument is immediately removed from the active note list. No other notes are affected.

Examples

The following example uses the turnoff opcode. It will cause a note to terminate when a control signal passes a certain threshold (here the Nyquist frequency). It uses the file *turnoff.csd* [examples/turnoff.csd].

Exemple 523. Example of the turnoff opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o turnoff.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  k1 expon 440, p3/10,880     ; begin gliss and continue
  if k1 < sr/2  kgoto contin  ; until Nyquist detected
    turnoff  ; then quit

contin:
  a1 oscil 10000, k1, 1
  out a1
endin


</CsInstruments>
```

```
<CsScore>

; Table #1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for 4 seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

See Also

ihold

turnoff2

turnoff2 — Turn off instance(s) of other instruments at performance time.

Description

Turn off instance(s) of other instruments at performance time.

Syntax

```
turnoff2 kinsno, kmode, krelease
```

Performance

kinsno -- instrument to be turned off (can be fractional) if zero or negative, no instrument is turned off

kmode -- sum of the following values:

- 0, 1, or 2: turn off all instances (0), oldest only (1), or newest only (2)
- 4: only turn off notes with exactly matching (fractional) instrument number, rather than ignoring fractional part
- 8: only turn off notes with indefinite duration (p3 < 0 or MIDI)

krelease -- if non-zero, the turned off instances are allowed to release, otherwise are deactivated immediately (possibly resulting in clicks)

See Also

turnoff

Credits

Author: Istvan Varga
2005

New in Csound 5.00

turnon

turnon — Activate an instrument for an indefinite time.

Description

Activate an instrument for an indefinite time.

Syntax

```
turnon insnum [, itime]
```

Initialization

insnum -- instrument number to be activated

itime (optional, default=0) -- delay, in seconds, after which instrument *insnum* will be activated. Default is 0.

Performance

turnon activates instrument *insnum* after a delay of *itime* seconds, or immediately if *itime* is not specified. Instrument is active until explicitly turned off. (See *turnoff*.)

unirand

unirand — Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares unirand krange
```

```
ires unirand krange
```

```
kres unirand krange
```

Exécution

krange -- l'intervalle des nombres aléatoires (0 - *krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode unirand. Il utilise le fichier *unirand.csd* [examples/unirand.csd].

Exemple 524. Exemple de l'opcode unirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o unirand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; krange = 1

  i1 unirand 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.840
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *weibull*

Crédits

Auteur: Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

upsamp

upsamp — Modify a signal by up-sampling.

Description

Modify a signal by up-sampling.

Syntax

```
ares upsamp ksig
```

Performance

upsamp converts a control signal to an audio signal. It does it by simple repetition of the kval. *upsamp* is a slightly more efficient form of the assignment, *asig = ksig*.

Examples

```
asrc  buzz      10000,440,20, 1     ; band-limited pulse train
adif  diff      asrc                ; emphasize the highs
anew  balance   adif, asrc          ;   but retain the power
agate reson     asrc,0,440          ; use a lowpass of the original
asamp samphold  anew, agate         ;   to gate the new audiosig
aout  tone      asamp,100           ; smooth out the rough edges
```

See Also

diff, *downsamp*, *integ*, *interp*, *samphold*

urd

urd — Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

Description

Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

Syntaxe

```
aout = urd(ktableNum)
```

```
iout = urd(itableNum)
```

```
kout = urd(ktableNum)
```

Initialisation

itableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

Exécution

ktableNum -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

urd est le même opcode que *duserrnd*, mais on peut l'utiliser à la manière d'une fonction.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Voir Aussi

cuserrnd, *duserrnd*

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

vadd

vadd — Adds a scalar value to a vector in a table.

Description

Adds a scalar value to a vector in a table.

Syntax

```
vadd  ifn, kval, kelements [, kdstoffset] [, kverbose]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

Performance

kval - scalar value to be added

kelements - number of elements of the vector

kdstoffset - index offset for the destination table (Optional, default = 0)

kverbose - Selects whether or not warnings are printed (Default=0)

vadd adds the value of *kval* to each element of the vector contained in the table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is added every control period. Use with care or you will end up with very large numbers (or use *vadd_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

Examples

Here is an example of the vadd opcode. It uses the file *vadd.csd* [examples/vadd.csd].

Exemple 525. Example of the vadd opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd ifn1, ival, ielements, idstoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 5 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 8 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1 10 12
i2 1.6 0.2 1
```

```
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd_i, *vmult*, *vpow* and *vexp*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vadd_i

vadd_i — Adds a scalar value to a vector in a table.

Description

Adds a scalar value to a vector in a table.

Syntax

```
vadd_i  ifn, ival, ielements [, idstoffset]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

ielements - number of elements of the vector

ival - scalar value to be added

idstoffset - index offset for the destination table

Performance

vadd_i adds the value of *ival* to each element of the vector contained in the table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vadd*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Examples

Here is an example of the vadd_i opcode. It uses the file *vadd_i.csd* [examples/vadd_i.csd].

Exemple 526. Example of the vadd_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd, *vmult_i*, *vpow_i* and *vexp_i*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vaddv

vaddv — Performs addition between two vectorial control signals

Description

Performs addition between two vectorial control signals

Syntax

```
vaddv  ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

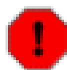
ksrcoffset - index offset for the source (ifn2) table (Default=0)

kverbose - Selects whether or not warnings are printed (Default=0)

vaddv adds two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i*opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

Please note that using the same table as source and destination table, might produce unexpected behavior so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are added). There's an i-rate ver-

sion of this opcode called *vaddv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vaddv opcode. It uses the file *vaddv.csd* [examples/vaddv.csd].

Exemple 527. Example of the vaddv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vaddv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
```

```
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2


i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vaddv_i

vaddv_i — Performs addition between two vectorial control signals at init time.

Description

Performs addition between two vectorial control signals at init time.

Syntax

```
vaddv_i  ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table (Default=0)

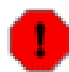
isrcoffset - index offset for the source (ifn2) table (Default=0)

Performance

vaddv_i adds two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vaddv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vaget

vaget — Access values of the current buffer of an a-rate variable by indexing.

Description

Access values of the current buffer of an a-rate variable by indexing. Useful for doing sample-by-sample manipulation at k-rate without using setksmps 1.

Note

Because this opcode does not do any bounds checking, the user must be careful not to try to read values past ksmps (the size of a buffer for an a-rate variable) by using index values greater than ksmps.

Syntax

```
kval vaget kndx, avar
```

Performance

kval - value read from avar

kndx - index of the sample to read from the current buffer of the given avar variable

avar - a-rate variable to read from

Examples

Here is an example of the vaget opcode. It uses the file *vaget.csd* [examples/vaget.csd].

Exemple 528. Example of the vaget opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarget.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=16
nchnls=2

        instr 1 ; Sqrt Signal
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

aout init 0
```

```
ksampnum init 0

kenv linseg 0, p3 * .5, 1, p3 * .5, 0

aout1 vco2 1, ifreq
aout2 vco2 .5, ifreq * 2
aout3 vco2 .2, ifreq * 4

aout sum         aout1, aout2, aout3

;Take Sqrt of signal, checking for negatives
kcount = 0

loopStart:

        kval vaget kcount,aout

        if (kval > .0) then
                kval = sqrt(kval)
        elseif (kval < 0) then
                kval = sqrt(-kval) * -1
        else
                kval = 0
        endif

        vaset kval, kcount,aout

loop_lt kcount, 1, ksmps, loopStart

aout = aout * kenv

aout moogladder aout, 8000, .1

aout = aout * iamp

outs aout, aout
        endin

</CsInstruments>
<CsScore>

i1 0.0 2 440 80
e

</CsScore>
</CsoundSynthesizer>
```

See Also

vaset

Credits

Author: Steven Yi

New in version 5.04

September 2006.

valpass

valpass — Variably reverberates an input signal with a flat frequency response.

Description

Variably reverberates an input signal with a flat frequency response.

Syntax

```
ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

Initialization

imaxlpt -- maximum loop time for *klpt*

iskip (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

insmps (optional, default=0) -- delay amount, as a number of samples.

Performance

krvt -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

xlpt -- variable loop time in seconds, same as *ilpt* in *comb*. Loop time can be as large as *imaxlpt*.

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Its output will begin to appear immediately.

See Also

alpass, *comb*, *reverb*, *vcomb*

Credits

Author: William « Pete » Moss
University of Texas at Austin
Austin, Texas USA
January 2002

vaset

vaset — Write value of into the current buffer of an a-rate variable by index.

Description

Write values into the current buffer of an a-rate variable at the given index. Useful for doing sample-by-sample manipulation at k-rate without using setksmps 1.

Note

Because this opcode does not do any bounds checking, the user must be careful not to try to write values past ksmps (the size of a buffer for an a-rate variable) by using index values greater than ksmps.

Syntax

```
vaset kval, kndx, avar
```

Performance

kval - value to write into avar

kndx - index of the sample to write to the current buffer of the given avar variable

avar - a-rate variable to write to

Examples

Here is an example of the vaset opcode. It uses the file *vaset.csd* [examples/vaset.csd].

Exemple 529. Example of the vaset opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarset.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=2

        instr 1 ; Sine Wave
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

kenv adsr 0.1, 0.05, .9, 0.2
```

```
aout init 0
ksampnum init 0

kcount = 0

iperiod = sr / ifreq

i2pi = 3.14159 * 2

loopStart:

kphase = (ksampnum % iperiod) / iperiod

knewval = sin(kphase * i2pi)

        vaset knewval, kcount,aout

        ksampnum = ksampnum + 1

loop_lt kcount, 1, ksmps, loopStart

aout = aout * iamp * kenv

outs aout, aout
        endin

</CsInstruments>
<CsScore>

i1 0.0 2 440 80
e

</CsScore>
</CsoundSynthesizer>
```

See Also

vaget

Credits

Author: Steven Yi

New in version 5.04

September 2006.

vbap16

vbap16 — Distributes an audio signal among 16 channels.

Description

Distributes an audio signal among 16 channels.

Syntax

```
ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]
```

Performance

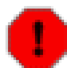
asig -- audio signal to be panned

kazim -- azimuth angle of the virtual source

kelev (optional) -- elevation angle of the virtual source

kspread (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

vbap16 takes an input signal, *asig*, and distribute it among 16 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16move, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csund 5.09.

vbap16move

vbap16move — Distribute an audio signal among 16 channels with moving virtual sources.

Description

Distribute an audio signal among 16 channels with moving virtual sources.

Syntax

```
ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]
```

Initialization

idur -- the duration over which the movement takes place.

ispread -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

ifldnum -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

ifld1, ifld2, ... -- azimuth angles or angular velocities, and relative durations of movement phases.

Performance

asig -- audio signal to be panned

vbap16move allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction total_time / number_of_intervals of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi_vel1*, [*iele_vel1*,] *iazi_vel2*, [*iele_vel2*,] .... Each velocity is applied to the note that is fraction total_time / number_of_velocities of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.

Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8move* for an example of usage of the *vbapXmove* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07

vbap4

vbap4 — Distributes an audio signal among 4 channels.

Description

Distributes an audio signal among 4 channels.

Syntax

```
ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]
```

Performance

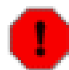
asig -- audio signal to be panned

kazim -- azimuth angle of the virtual source

kelev (optional) -- elevation angle of the virtual source

kspread (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

vbap4 takes an input signal, *asig* and distributes it among 4 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.06. Input parameters accept k-rate since Csund 5.09.

vbap4move

vbap4move — Distributes an audio signal among 4 channels with moving virtual sources.

Description

Distributes an audio signal among 4 channels with moving virtual sources.

Syntax

```
ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]
```

Initialization

idur -- the duration over which the movement takes place.

ispread -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

ifldnum -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

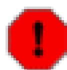
ifld1, ifld2, ... -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

Performance

asig -- audio signal to be panned

vbap4move allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction total_time / number_of_intervals of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi_vel1*, [*iele_vel1*,] *iazi_vel2*, [*iele_vel2*,] .... Each velocity is applied to the note that is fraction total_time / number_of_velocities of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8move* for an example of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07

vbap8

vbap8 — Distributes an audio signal among 8 channels.

Description

Distributes an audio signal among 8 channels.

Syntax

```
ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]
```

Performance

asig -- audio signal to be panned

kazim -- azimuth angle of the virtual source

kelev (optional) -- elevation angle of the virtual source

kspread (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

vbap8 takes an input signal, *asig*, and distributes it among 8 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.

Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Example

Here is a simple example of the *vbap8* opcode. It uses the file *vbap8.csd* [examples/vbap8.csd].

Exemple 530. Example of the vbap8 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac           -iadc    ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
-o vbap8.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

  sr        =           41000
  kr        =            441
  ksmps     =            100
  nchnls    =              4
  vbaplsinit            2, 8,   0, 45, 90, 135, 200, 245, 290, 315

          instr  1
  asig    oscil         20000, 440, 1
  a1,a2,a3,a4,a5,a6,a7,a8   vbap8   asig, p4, 0, 20 ;p4 = azimuth

  ;render twice with alternate outq  statements
  ;  to obtain two 4 channel .wav files:

          outq          a1,a2,a3,a4
  ;       outq          a5,a6,a7,a8
; or use an 8-channel output for realtime output (set nchnls to 8):
;        outo a1,a2,a3,a4,a5,a6,a7,a8
          endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
;          azimuth
i 1 0 1      20
i 1 + .      40
i 1 + .      60
i 1 + .      80
i 1 + .      100
i 1 + .      120
i 1 + .      140
i 1 + .      160
e


</CsScore>
</CsoundSynthesizer>
```

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap4move*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csund 5.09.

vbap8move

vbap8move — Distributes an audio signal among 8 channels with moving virtual sources.

Description

Distributes an audio signal among 8 channels with moving virtual sources.

Syntax

```
ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]
```

Initialization

idur -- the duration over which the movement takes place.

ispread -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

ifldnum -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

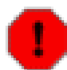
ifld1, ifld2, ... -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

Performance

asig -- audio signal to be panned

vbap8move allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction total_time / number_of_intervals of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi_vel1*, [*iele_vel1*,] *iazi_vel2*, [*iele_vel2*,] .... Each velocity is applied to the note that is fraction total_time / number_of_velocities of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Example

Here is a simple example of the *vbap8move* opcode. It uses the file *vbap8move.csd* [examples/vbap8move.csd].

Exemple 531. Example of the vbap8move opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vbap4move.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 10
nchnls = 8

;Example by Hector Centeno 2007

vbaplsinit    2, 8, 15, 65, 115, 165, 195, 245, 295, 345

    instr 1
ifldnum = 9
ispread = 30
idur = p3

;; Generate a sound source
kenv loopseg 10, 0, 0, 0, 0.5, 1, 10, 0
a1 pinkish 3000*kenv

;; Move circling around once all the speakers
aout1, aout2, aout3, aout4, aout5, aout6, aout7, aout8 vbap8move a1, idur, ispread, ifldnum, 15, 65, 11

;; Speaker mapping
aFL = aout8 ; Front Left
aFR = aout1 ; Front Right
aMFL = aout7 ; Mid Front Left
aMFR = aout2 ; Mid Front Right
aMBL = aout6 ; Mid Back Left
aMBR = aout3 ; Mid Back Right
aBL = aout5 ; Back Left
aBR = aout4 ; Back Right

outo aFL, aFR, aMFL, aMFR, aMBL, aMBR, aBL, aBR

    endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbaplsinit*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07

vbaplsinit

vbaplsinit — Configures VBAP output according to loudspeaker parameters.

Description

Configures VBAP output according to loudspeaker parameters.

Syntax

```
vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]
```

Initialization

idim -- dimensionality of loudspeaker array. Either 2 or 3.

ilsnum -- number of loudspeakers. In two dimensions, the number can vary from 2 to 16. In three dimensions, the number can vary from 3 and 16.

idir1, idir2, ..., idir32 -- directions of loudspeakers. Number of directions must be less than or equal to 16. In two-dimensional loudspeaker positioning, *idir*n is the azimuth angle respective to *n*th channel. In three-dimensional loudspeaker positioning, fields are the azimuth and elevation angles of each loudspeaker consequently (*azi1*, *ele1*, *azi2*, *ele2*, etc.).

Performance

VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.

Examples

See the entry for *vbap16move* and *vbap8* for examples of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbapz*, *vbapzmove*

Credits

Author: Ville Pulkki
Sibelius Academy Computer Music Studio
Laboratory of Acoustics and Audio Signal Processing

Helsinki University of Technology
Helsinki, Finland
May 2000

New in Csound Version 4.07

vbapz

vbapz — Writes a multi-channel audio signal to a ZAK array.

Description

Writes a multi-channel audio signal to a ZAK array.

Syntax

```
vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]
```

Initialization

inumchnls -- number of channels to write to the ZA array. Must be in the range 2 - 256.

istartndx -- first index or position in the ZA array to use

Performance

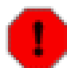
asig -- audio signal to be panned

kazim -- azimuth angle of the virtual source

kelev (optional) -- elevation angle of the virtual source

kspread (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

The opcode *vbapz* is the multiple channel analog of the opcodes like *vbap4*, working on *inumchnls* and using a ZAK array for output.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapzmove*

Credits

John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csund 5.09.

vbapzmove

vbapzmove — Writes a multi-channel audio signal to a ZAK array with moving virtual sources.

Description

Writes a multi-channel audio signal to a ZAK array with moving virtual sources.

Syntax

```
vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
      ifld2, [...]
```

Initialization

inumchnls -- number of channels to write to the ZA array. Must be in the range 2 - 256.

istartndx -- first index or position in the ZA array to use

idur -- the duration over which the movement takes place.

ispread -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

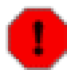
ifldnum -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

ifld1, ifld2, ... -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

Performance

asig -- audio signal to be panned

The opcode *vbapzmove* is the multiple channel analog of the opcodes like *vbap4move*, working on *inumchnls* and using a ZAK array for output.



Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

Examples

See the entry for *vbap8move* for an example of usage of the *vbap* opcodes.

Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

See Also

vbap16, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*,

Credits

John ffitch
University of Bath/Codemist Ltd.
Bath, UK
May 2000

New in Csound Version 4.07

vcella

vcella — Automate Cellulaire

Description

Automate Cellulaire unidimensionnel appliqué à des vecteurs de Csound.

Syntaxe

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \
      iRuleFunc, ielements, irulelen [, iradius]
```

Initialisation

ioutFunc - numéro de la table dans laquelle l'état de chaque cellule est stocké

initStateFunc - numéro de la table contenant l'état initial de chaque cellule

iRuleFunc - numéro de la table de consultation contenant les règles

ielements - nombre total de cellules

irulelen - nombre total de règles

iradius (facultatif) - rayon de l'Automate Cellulaire. Actuellement, le rayon de l'AC peut valoir 1 ou 2 (la valeur par défaut est 1)

Exécution

ktrig - signal de déclenchement. Chaque fois qu'il est non nul, une nouvelle génération de cellules est évaluée.

kreinit - signal de déclenchement. Chaque fois qu'il est non nul, l'état de toutes les cellules est forcé à celui de *initStateFunc*.

vcella met en œuvre un automate cellulaire pour lequel l'état de chaque cellule est stocké dans *ioutFunc*. Ainsi *ioutFunc* est un vecteur contenant l'état courant de chaque cellule. Ce vecteur variable peut être utilisé avec d'autres opcodes basés sur des vecteurs, tels que *adsynt*, *vmap*, *vpowv* etc.

initStateFunc est un vecteur d'entrée contenant la valeur initiale de la rangée de cellules, tandis que *iRuleFunc* est un vecteur d'entrée contenant les règles sous la forme d'une table de consultation. Notez que *initStateFunc* et *iRuleFunc* peuvent être modifiés pendant l'exécution au moyen d'autres opcodes basés sur des vecteurs (par exemple *vcopy*) afin de forcer un changement de règle et d'état pendant l'exécution.

Une nouvelle génération de cellules est évaluée chaque fois que *ktrig* contient une valeur non nulle. De plus, l'état de toutes les cellules peut être forcé à l'état correspondant dans *initStateFunc* chaque fois que *kreinit* contient une valeur non nulle.

Le rayon de l'algorithme d'AC peut valoir 1 ou 2 (argument facultatif *iradius*).

Exemples

Voici un exemple de l'opcode vcella. Il utilise le fichier *vcella.csd* [examples/vcella.csd].

L'exemple suivant utilise l'opcode *vcella*

Exemple 532. Exemple de l'opcode vcella.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcella.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; vcella.csd
; by Anthony Kozar

; This file demonstrates some of the new opcodes available in
; Csound 5 that come from Gabriel Maldonado's CsoundAV.

sr          = 44100
kr          = 4410
ksmps       = 10
nchnls      = 1

; Cellular automata-driven oscillator bank using vcella and adsynt
instr 1
  idur       = p3
  iCArate    = p4                               ; number of times per second the CA calculates new valu

  ; f-tables for CA parameters
  iCAinit    = p5                               ; CA initial states
  iCArule    = p6                               ; CA rule values
  ; The rule is used as follows:
  ; the states (values) of each cell are summed with their neighboring cells within
  ; the specied radius (+/- 1 or 2 cells).  Each sum is used as an index to read a
  ; value from the rule table which becomes the new state value for its cell.
  ; All new states are calculated first, then the new values are all applied
  ; simultaneously.

  ielements = ftlen(iCAinit)
  inumrules = ftlen(iCArule)
  iradius    = 1

  ; create some needed tables
  iCAstate  ftgen      0, 0, ielements, -2, 0    ; will hold the current CA states
  ifreqs    ftgen      0, 0, ielements, -2, 0    ; will hold the oscillator frequency for each cell
  iamps     ftgen      0, 0, ielements, -2, 0    ; will hold the amplitude for each cell

  ; calculate cellular automata state
  ktrig     metro      iCArate                  ; trigger the CA to update iCArate times per second
            vcella     ktrig, 0, iCAstate, iCAinit, iCArule, ielements, inumrules, iradius

  ; scale CA state for use as amplitudes of the oscillator bank
            vcopy      iamps, iCAstate, ielements
            vmult      iamps, (1/3), ielements   ; divide by 3 since state values are 0-3

            vport      iamps, .01, ielements     ; need to smooth the amplitude changes for adsynt
  ; we could use adsynt2 instead of adsynt, but it does not seem to be working

  ; i-time loop for calculating frequencies
  index     =          0
  inew      =          1
  iratio    =          1.125                    ; just major second (creating a whole tone scale)
loop1:
            tableiw    inew, index, ifreqs, 0    ; 0 indicates integer indices
  inew      =          inew * iratio
  index     =          index + 1
  if (index < ielements) igoto loop1
```

```
  ; create sound with additive oscillator bank
  ifreqbase = 64
  iwavefn   = 1
  iphs      = 2                                ; random oscillator phases

  kenv      linseg    0.0, 0.5, 1.0, idur - 1.0, 1.0, 0.5, 0.0
  aosc      adsynt    kenv, ifreqbase, iwavefn, ifreqs, iamps, ielements, iphs

            out       aosc * ampdb(68)
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1

; This example uses a 4-state cellular automata
; Possible state values are 0, 1, 2, and 3

; CA initial state
; We have 16 cells in our CA, so the initial state table is size 16
f10 0 16 -2  0 1 0 0  1 0 0 2  2 0 0 1  0 0 1 0

; CA rule
; The maximum sum with radius 1 (3 cells) is 9, so we need 10 values in the rule (0-9)
f11 0 16 -2  1 0 3 2 1  0 0 2 1 0

; Here is our one and only note!
i1 0  20  4  10  11

e

</CsScore>
</CsoundSynthesizer>
```

Crédits

Ecrit par : Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

Exemple par : Anthony Kozar

vco

vco — Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée.

Description

Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée, basée sur l'intégration d'impulsions à bande de fréquence limitée. *vco* peut être utilisé pour simuler différentes formes d'onde analogiques.

Syntaxe

```
ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \
      [, iphs] [, iskip]
```

Initialisation

iwave -- détermine la forme d'onde :

- *iwave* = 1 - dent de scie
- *iwave* = 2 - carrée/PWM
- *iwave* = 3 - triangle/dent de scie/rampe

ifn (facultatif, par défaut 1) -- numéro de table d'une fonction sinus stockée. Doit pointer sur une table valide qui contient une onde sinus. Csound rapportera une erreur si ce paramètre n'est pas fixé et que la table n°1 n'existe pas.

imaxd (facultatif, par défaut 1) -- temps de retard maximum. Une durée de 1/*ifqc* peut être nécessaire pour les formes d'onde PWM et triangle. Le temps d'ajustement de la hauteur à cette valeur peut aller jusqu'à 1/(fréquence minimale).

ileak (facultatif, par défaut 0) -- si *ileak* se situe entre zéro et un (0 < *ileak* < 1), *ileak* est utilisé comme facteur de fuite de l'intégrateur. Sinon un facteur de fuite de 0,999 est utilisé pour les ondes en dent de scie et carrée et de 0,995 pour l'onde triangle. On peut l'utiliser pour « aplatir » l'onde carrée ou « renforcer » l'onde en dent de scie dans les fréquences basses en fixant *ileak* à 0,99999 ou à une valeur semblable. Le résultat devrait être une onde carrée sonnant plus faux.

inyx (facultatif, par défaut 0,5) -- est utilisé pour déterminer le nombre d'harmoniques dans l'impulsion à bande de fréquence limitée. Tous les harmoniques jusqu'à *sr* * *inyx* seront utilisés. La valeur par défaut donne *sr* * 0,5 (*sr*/2). Pour *sr*/4 utiliser *inyx* = 0,25. Cela peut générer un son plus « gras » dans certains cas.

iphs (facultatif, par défaut 0) -- c'est une valeur de phase. Il y a un artefact (comme un bogue) dans *vco* qui se produit pendant la première demi-période de l'onde carrée et qui rend la forme d'onde plus grande en amplitude que les autres. La valeur de *iphs* a un effet sur cet artefact. En particulier, si l'on fixe *iphs* à 0,5 la première demi-période de l'onde carrée ressemblera à une petite onde triangulaire. Ceci peut être préférable à la grande forme d'onde de l'artefact qui est le comportement par défaut.

iskip (facultatif, par défaut 0) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

Exécution

kpw -- détermine soit la largeur de la pulsation (si *iwave* vaut 2) soit le caractère de la dent de scie / rampe (si *iwave* vaut 3). La valeur de *kpw* doit être supérieure à 0 et inférieure à 1. Une valeur de 0,5 générera une onde carrée (si *iwave* vaut 2) ou une onde triangle (si *iwave* vaut 3).

xamp -- détermine l'amplitude

xcps -- fréquence de l'onde en cycles par seconde.

Exemples

Voici un exemple de l'opcode vco. Il utilise le fichier *vco.csd* [examples/vco.csd].

Exemple 533. Exemple de l'opcode vco.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vco.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1
instr 1
  ; Set the amplitude.
  kamp = p4

  ; Set the frequency.
  kcps = cpspch(p5)

  ; Select the wave form.
  iwave = p6

  ; Set the pulse-width/saw-ramp character.
  kpw init 0.5

  ; Use Table #1.
  ifn = 1

  ; Generate the waveform.
  asig vco kamp, kcps, iwave, kpw, ifn

  ; Output and amplification.
  out asig
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 65536 10 1

; Define the score.
; p4 = raw amplitude (0-32767)
; p5 = frequency, in pitch-class notation.
```

```
; p6 = the waveform (1=Saw, 2=Square/PWM, 3=Tri/Saw-Ramp-Mod)
i 1 00 02 20000 05.00 1
i 1 02 02 20000 05.00 2
i 1 04 02 20000 05.00 3

i 1 06 02 20000 07.00 1
i 1 08 02 20000 07.00 2
i 1 10 02 20000 07.00 3

i 1 12 02 20000 09.00 1
i 1 14 02 20000 09.00 2
i 1 16 02 20000 09.00 3

i 1 18 02 20000 11.00 1
i 1 20 02 20000 11.00 2
i 1 22 02 20000 11.00 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

vco2

Crédits

Auteur : Hans Mikelson
Décembre 1998

Nouveau dans la version 3.50 de Csound

Novembre 2002. Correction de la documentation pour le paramètre *kpw*. Merci à Luis Jure et à Hans Mikelson.

vco2

vco2 — Implémentation d'un oscillateur à bande de fréquence limitée qui utilise des tables pré-calculées.

Description

vco2 est semblable à *vco*. Mais l'implémentation utilise des tables pré-calculées de formes d'onde à bande de fréquence limitée (voir aussi *GEN30*) plutôt que d'intégrer des impulsions. Cet opcode peut être plus rapide que *vco* (particulièrement lors de l'utilisation d'un faible taux de contrôle) et il permet également une meilleure qualité sonore. De plus, il y a plus de formes d'onde et la phase de l'oscillateur peut être modulée au taux-k. Il a pour inconvénient une utilisation plus importante de la mémoire. Pour plus de détails sur les tables de *vco2*, voir aussi *vco2init* et *vco2ft*.

Syntaxe

```
ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]
```

Initialisation

imode (facultatif, par défaut 0) -- somme des valeurs représentant la forme d'onde et ses valeurs de contrôle.

On peut utiliser ces valeurs pour *imode* :

- 16 : active le contrôle de la phase au taux-k (s'il est positionné, *kphs* est un paramètre de taux-k nécessaire pour permettre la modulation de la phase)
- 1 : ignorer l'initialisation

On peut utiliser exactement une seule de ces valeurs de *imode* pour choisir la forme d'onde à générer :

- 14 : forme d'onde -1 définie par l'utilisateur (nécessite l'utilisation de l'opcode *vco2init*)
- 12 : triangle (pas de rampe, plus rapide)
- 10 : onde carrée (pas de PWM, plus rapide)
- 8 : 4 * x * (1 - x) (c'est-à-dire l'intégration d'une dent de scie)
- 6 : pulsation (non normalisée)
- 4 : dent de scie / triangle / rampe
- 2 : carrée / PWM
- 0 : dent de scie

La valeur par défaut de *imode* est zéro, ce qui signifie une onde en dent de scie sans contrôle de la phase au taux-k.

inyx (facultatif, par défaut 0,5) -- largeur de bande de l'onde générée exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à *sr*/2), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 (*sr*/4), ou à 0,3333 (*sr*/3), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

Exécution

ares -- le signal audio en sortie.

kamp -- amplitude. Si *imode* vaut 6 (pulsation), le niveau de sortie réel peut être bien plus élevé que cette valeur.

kcps -- fréquence en Hz (doit être dans l'intervalle -*sr*/2 à *sr*/2).

kpw (facultatif) -- largeur de pulsation de l'onde carrée (*imode* = 2) ou caractéristiques de l'onde triangle ou rampe (*imode* = 4). Il n'est requis que pour ces formes d'onde et il est ignoré dans les autres cas. L'intervalle attendu va de 0 à 1, toutes les autres valeurs y étant ramenées cycliquement.

Avertissement

kpw ne doit pas être une valeur entière exacte (0 ou 1) lors de la génération d'une onde en dent de scie / triangle / rampe (*imode* = 4). Dans ce cas, l'intervalle recommandé est d'environ 0,01 à 0,99. Cette limitation n'existe pas pour une forme d'onde carrée/PWM.

kphs (facultatif) -- phase de l'oscillateur (en fonction de *imode*, ce sera un paramètre facultatif de taux-i qui vaut zéro par défaut ou un paramètre obligatoire de taux-k). Comme pour *kpw*, l'intervalle attendu va de 0 à 1.

Note

Si l'on utilise un faible taux de contrôle, la largeur de pulsation (*kpw*) et la modulation de phase (*kphs*) sont converties en interne en modulation de fréquence. Cela permet un traitement plus rapide et réduit le nombre d'artefacts. Mais dans le cas de notes très longues avec des changements rapides et continus de *kpw* ou de *kphs*, la phase peut se décaler par rapport à la valeur voulue. Dans la plupart des cas, l'erreur de phase sera au maximum de 0,037 par heure (en supposant un taux d'échantillonnage de 44100 Hz).

Ceci pose problème principalement avec la largeur d'impulsion (*kpw*) par la possible apparition de divers artefacts. En attendant la résolution de ces problèmes dans de futures versions de *vco2*, les recommandations suivantes peuvent être utiles :

- N'utiliser que des valeurs de *kpw* dans l'intervalle 0,05 à 0,95. (Il y a plus d'artefacts au voisinage des valeurs entières)
- Essayer d'éviter de moduler *kpw* par des formes d'onde asymétriques telles que l'onde en dent de scie. Il est très peu probable qu'une modulation symétrique relativement lente (<= 20 Hz) (par exemple une onde sinus ou triangle), que des fonctions splines aléatoires (également lentes) ou qu'une pulsation de largeur fixe causent des problèmes de synchronisation.
- Dans certains cas, l'ajout d'un tremblement aléatoire (par exemple, des fonctions spline avec une amplitude d'environ 0,01) à *kpw* peut aussi résoudre le problème.

Exemples

Voici un exemple de l'opcode vco2. Il utilise le fichier *vco2.csd* [examples/vco2.csd].

Exemple 534. Exemple de l'opcode vco2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vco2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr      =  44100
ksmps   =  10
nchnls  =  1

; user defined waveform -1: trapezoid wave with default parameters (can be
; accessed at ftables starting from 10000)
itmp    ftgen 1, 0, 16384, 7, 0, 2048, 1, 4096, 1, 4096, -1, 4096, -1, 2048, 0
ift     vco2init -1, 10000, 0, 0, 0, 1
; user defined waveform -2: fixed table size (4096), number of partials
; multiplier is 1.02 (~238 tables)
itmp    ftgen 2, 0, 16384, 7, 1, 4095, 1, 1, -1, 4095, -1, 1, 0, 8192, 0
ift     vco2init -2, ift, 1.02, 4096, 4096, 2

        instr 1
kcps    expon p4, p3, p5                ; instr 1: basic vco2 example
a1      vco2 12000, kcps                ; (sawtooth wave with default
        out a1                          ; parameters)
        endin

        instr 2
kcps    expon p4, p3, p5                        ; instr 2:
kpw     linseg 0.1, p3/2, 0.9, p3/2, 0.1        ; PWM example
a1      vco2 10000, kcps, 2, kpw
        out a1
        endin

        instr 3
kcps    expon p4, p3, p5                ; instr 3: vco2 with user
a1      vco2 14000, kcps, 14            ; defined waveform (-1)
aenv    linseg 1, p3 - 0.1, 1, 0.1, 0   ; de-click envelope
        out a1 * aenv
        endin

        instr 4
kcps    expon p4, p3, p5                ; instr 4: vco2ft example,
kfn     vco2ft kcps, -2, 0.25           ; with user defined waveform
a1      oscilikt 12000, kcps, kfn       ; (-2), and sr/4 bandwidth
        out a1
        endin

</CsInstruments>
<CsScore>

i 1  0 3 20 2000
i 2  4 2 200 400
i 3  7 3 400 20
i 4 11 2 100 200

f 0 14

e

</CsScore>
```

```
</CsoundSynthesizer>
```

Voir Aussi

vco, *vco2ft*, *vco2ift* et *vco2init*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

vco2ft

vco2ft — Retourne un numéro de table au taux-k pour une fréquence d'oscillateur donnée et une forme d'onde.

Description

vco2ft retourne le numéro d'une table de fonction pour générer la forme d'onde spécifiée à une fréquence donnée. Ce numéro de table de fonction peut être utilisé par n'importe quel opcode de Csound qui génère un signal en lisant une table de fonction (comme *oscilikt*). Les tables doivent avoir été calculées par *vco2init* avant l'appel de *vco2ft* et partagées comme ftables de Csound (*ibasfn*).

Syntaxe

```
kfn vco2ft kcps, iwave [, inyx]
```

Initialisation

iwave -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 : 4 * x * (1 - x) (intégration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

inyx (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à *sr*/2), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 (*sr*/4), ou à 0,3333 (*sr*/3), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

Exécution

kfn -- le numéro de la ftable , retourné au taux-k.

kcps -- fréquence en Hz, retournée au taux-k. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse *sr*/2 (ou *sr* * *inyx*), la table sélectionnée ne contiendra que du silence.

Exemples

Voir l'exemple de l'opcode *vco2*.

Voir Aussi

vco2ift, *vco2init* et *vco2*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

vco2ift

vco2ift — Retourne un numéro de table au temps-i pour une fréquence d'oscillateur donnée et une forme d'onde.

Description

vco2ift est le même que *vco2ft*, mais il travaille au temps-i. Il est prévu pour être utilisé avec les opcodes qui attendent un numéro de table au taux-i (par exemple, *oscili*).

Syntaxe

```
ifn vco2ift icps, iwave [, inyx]
```

Initialisation

ifn -- le numéro de ftable.

icps -- fréquence en Hz. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse *sr*/2 (ou *sr* * *inyx*), la table sélectionnée ne contiendra que du silence.

iwave -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 : 4 * x * (1 - x) (integration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

inyx (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à *sr*/2), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 (*sr*/4), ou à 0,3333 (*sr*/3), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

Voir Aussi

vco2ft, *vco2init* et *vco2*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

vco2init

vco2init — Calcul des tables à utiliser par l'opcode *vco2*.

Description

vco2init calcule des tables à utiliser par l'opcode *vco2*. En option, on peut accéder aussi à ces tables comme si elles étaient des tables de fonction standard de Csound. Dans ce cas, on peut utiliser *vco2ft* pour trouver le numéro de table correct pour une fréquence d'oscillateur donnée.

Dans la plupart des cas, cet opcode est appelé depuis l'en-tête de l'orchestre. L'utilisation de *vco2init* dans des instruments est possible mais non recommandée. En effet, le remplacement de tables durant l'exécution peut causer un plantage de Csound si d'autres opcodes sont en train d'accéder à ces tables au même moment.

Notez que *vco2init* n'est pas nécessaire au fonctionnement de *vco2* (les tables sont automatiquement allouées au premier appel de *vco2*, si ce n'est pas déjà fait), cependant il peut être utile dans certains cas :

- Pré-calcul des tables pendant le chargement de l'orchestre. C'est utile lorque l'on ne veut pas générer les tables pendant l'exécution, afin de ne pas risquer une interruption du traitement en temps réel.
- Partage des tables comme ftables Csound. Par défaut, ces tables ne sont accessibles que par *vco2*.
- Modification des paramètres par défaut des tables (par exemple leur taille) ou utilisation d'une forme d'onde définie par l'utilisateur spécifiée dans une table de fonction.

Syntaxe

```
ifn vco2init iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]
```

Initialisation

ifn -- le premier numéro de table libre après les tables allouées. Si *ibasfn* n'a pas été spécifié, -1 est retourné.

iwave -- somme des valeurs suivantes sélectionnant quelles tables d'onde il faut calculer :

- 16 : triangle
- 8 : onde carrée
- 4 : pulsation (non normalisée)
- 2 : 4 * x * (1 - x) (intégration d'une dent de scie)
- 1 : dent de scie

Alternativement, *iwave* peut être fixé à un entier négatif qui sélectionne une forme d'onde définie par l'utilisateur. Pour cela, le paramètre *isrcft* doit être aussi spécifié. *vco2* peut accéder à la forme d'onde numéro -1. Cependant, les autres formes d'onde définies par l'utilisateur ne sont utilisables qu'avec

vco2ft ou *vco2ift*.

ibasfn (facultatif, par défaut -1) -- numéro de ftable à partir duquel les opcodes autres que *vco2* peuvent accéder à l'ensemble de tables. Il est nécessaire pour les formes d'onde définies par l'utilisateur, à l'exception de -1. Si cette valeur est inférieure à 1, il n'est pas possible d'accéder aux tables calculées par *vco2init* en tant que tables de fonction de Csound.

ipmul (facultatif, par défaut 1,05) -- coefficient multiplicatif pour le nombre d'harmoniques. Si une table a n harmoniques, la suivante en aura n * *ipmul* (au moins n + 1). L'intervalle autorisé pour *ipmul* va de 1,01 à 2. Zéro et les valeurs négatives sélectionnent la valeur par défaut (1,05).

iminsiz (facultatif, par défaut -1) -- taille de table minimale.

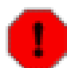
imaxsiz (facultatif, par défaut -1) -- taille de table maximale.

La taille de table réelle est calculée en multipliant la racine carrée du nombre d'harmoniques par *iminsiz*, puis en arrondissant le résultat à la puissance de deux supérieure, tout en l'obligeant à ne pas dépasser *imaxsiz*.

Les deux paramètres, *iminsiz* et *imaxsiz*, doivent être des puissances de deux, dans l'intervalle autorisé. L'intervalle autorisé va de 16 à 262144 pour *iminsiz* jusqu'à 16777216 pour *imaxsiz*. Zéro ou des valeurs négatives sélectionnent les réglages par défaut :

- La taille minimale est 128 pour toutes les formes d'onde sauf pour la pulsation (*iwave* = 4). Sa taille minimale est de 256.
- La taille maximale par défaut vaut normalement la taille minimale multipliée par 64, mais pas plus de 16384 si possible. Elle vaut toujours au moins le taille minimale.

isrcft (facultatif, par défaut -1) -- numéro de la ftable source pour les formes d'onde définies par l'utilisateur (si *iwave* < 0). *isrcft* doit pointer sur une table de fonction contenant la forme d'onde à utiliser pour générer le tableau de tables. Il est recommandé d'avoir une taille de table d'au moins *imaxsiz* points. Si *iwave* n'est pas négatif (les tables d'onde internes sont utilisées), *isrcft* est ignoré.



Avertissement

Le nombre et la taille des tables ne sont pas fixes. Les orchestres ne doivent pas dépendre de ces paramètres, car ils peuvent changer d'une version à l'autre de Csound.

Si la table sélectionnée existe déjà, elle est remplacée. Si un opcode est en train d'accéder aux tables au même moment, il est fort probable qu'un plantage se produise. C'est pourquoi il est recommandé de n'utiliser *vco2init* que dans l'en-tête de l'orchestre.

Il ne faut pas remplacer/écraser ces tables par les routines GEN ou l'opcode *ftgen*. Sinon, un comportement imprévisible voire un plantage de Csound peuvent se produire si *vco2* est utilisé. Le premier numéro de ftable libre après le tableau de tables est retourné dans *ifn*.

Exemples

Voir l'exemple de l'opcode *vco2*.

Voir Aussi

vco2ft, *vco2ift* et *vco2*.

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

vcomb

vcomb — Variably reverberates an input signal with a « colored » frequency response.

Description

Variably reverberates an input signal with a « colored » frequency response.

Syntax

```
ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

Initialization

imaxlpt -- maximum loop time for *klpt*

iskip (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

insmps (optional, default=0) -- delay amount, as a number of samples.

Performance

krvt -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

xlpt -- variable loop time in seconds, same as *ilpt* in *comb*. Loop time can be as large as *imaxlpt*.

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Output will appear only after *ilpt* seconds.

Examples

Here is an example of the vcomb opcode. It uses the file *vcomb.csd* [examples/vcomb.csd].

Exemple 535. Example of the vcomb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc       -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

; Example by Jonathan Murphy and Charles Gran 2007
  sr      =  44100
  ksmps      =  10
```

```
  nchnls    =  2

        ; new, and important. Make sure that midi note events are only
        ; received by instruments that actually need them.

        ; turn default midi routing off
        massign        0, 0
        ; route note events on channel 1 to instr 1
        massign        1, 1

; Define your midi controllers
#define C1 #21#
#define C2 #22#
#define C3 #23#

; Initialize MIDI controllers
            initc7    1, $C1, 0.5                    ;delay send
            initc7    1, $C2, 0.5                    ;delay: time to zero
            initc7    1, $C3, 0.5                    ;delay: rate

  gaosc     init      0

; Define an opcode to "smooth" the MIDI controller signal
    opcode smooth, k, k
  kin       xin
  kport     linseg    0, 0.0001, 0.01, 1, 0.01
  kin       portk     kin, kport
            xout      kin
    endop

instr   1
 ; Generate a sine wave at the frequency of the MIDI note that triggered the intrument
  ifqc     cpsmidi
  iamp     ampmidi   10000
  aenv     linenr    iamp, .01, .1, .01           ;envelope
  a1      oscil     aenv, ifqc, 1
; All sound goes to the global variable gaosc
  gaosc     =  gaosc + a1
    endin

    instr     198 ; ECHO
  kcmbsnd   ctrl7     1, $C1, 0, 1                    ;delay send
  ktime     ctrl7     1, $C2, 0.01, 6                    ;time loop fades out
  kloop     ctrl7     1, $C3, 0.01, 1                    ;loop speed
; Receive MIDI controller values and then smooth them
  kcmbsnd   smooth    kcmbsnd
  ktime     smooth    ktime
  kloop     smooth    kloop
  imaxlpt   =  1                                   ;max loop time
; Create a variable reverberation (delay) of the gaosc signal
  acomb     vcomb     gaosc, ktime, kloop, imaxlpt, 1
  aout     =  (acomb * kcmbsnd) + gaosc * (1 - kcmbsnd)
            outs      aout, aout
  gaosc     =  0
    endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1
i198 0 10000
e
</CsScore>
</CsoundSynthesizer>
```

See Also

alpass, *comb*, *reverb*, *valpass*

Credits

Author: William « Pete » Moss
University of Texas at Austin

Austin, Texas USA
January 2002

vcopy

vcopy — Copies between two vectorial control signals

Description

Copies between two vectorial control signals

Syntax

```
vcopy  ifn, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]
```

Initialization

ifn1 - number of the table where the vectorial signal will be copied (destination)

ifn2 - number of the table hosting the vectorial signal to be copied (source)

Performance

kelements - number of elements of the vector

kdstoffset - index offset for the destination (ifn1) table (Default=0)

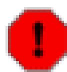
ksrcoffset - index offset for the source (ifn2) table (Default=0)

kverbose - Selects whether or not warnings are printed (Default=0)

vcopy copies *kelements* elements from *ifn2* (starting from position *isrcoffset*) to *ifn1* (starting from position *idstoffset*). Useful to keep old vector values, by storing them in another table.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *kdstoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are copied). There's an i-rate version of this opcode called *vcopy_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate.

This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexp*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Examples

Here is an example of the vcopy opcode. It uses the file *vcopy.csd* [examples/vcopy.csd].

Exemple 536. Example of the vcopy opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcopy.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


sr=44100
kr=4410
ksmps=10
nchnls=2

        instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
        endin

        instr 2
vcopy 2, 1, 20000 ;copy vector from sample to empty table
vmult 5, 20000, 262144 ;scale noise to make it audible
vcopy 1, 5, 20000  ;put noise into sample
turnoff
        endin

        instr 3
vcopy 1, 2, 20000 ;put original information back in
turnoff
        endin

</CsInstruments>
<CsScore>
f1  0 262144   -1 "beats.wav" 0 4 0
f2  0 262144   2  0

f5  0 262144   21  3 30000

i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
```

```
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vcopy_i

vcopy_i — Copies a vector from one table to another.

Description

Copies a vector from one table to another.

Syntax

```
vcopy_i  ifn, ifn2, ielements [,idstoffset, isrcoffset]
```

Initialization

ifn - number of the table where the vectorial signal will be copied

ifn - number of the table hosting the vectorial signal to be copied

ielements - number of elements of the vector

idstoffset - index offset for destination table

isrcoffset - index offset for source table

Performance

vcopy copies *ielements* elements from *ifn2* (starting from position *isrcoffset*) to *ifn1* (starting from position *idstoffset*). Useful to keep old vector values, by storing them in another table. This opcode is exactly the same as *vcopy* but performs all the copying on the intialization pass only.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 0). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector elements will be 0).

Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexp*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Examples

See *vcopy* for an example.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vdelay

vdelay — An interpolating variable time delay.

Description

This is an interpolating variable time delay, it is not very different from the existing implementation (*deltapi*), it is only easier to use.

Syntax

```
ares vdelay asig, adel, imaxdel [, iskip]
```

Initialization

imaxdel -- Maximum value of delay in milliseconds. If *adel* gains a value greater than *imaxdel* it is folded around *imaxdel*. This should not happen.

iskip -- Skip initialization if present and non-zero

Performance

With this unit generator it is possible to do Doppler effects or chorusing and flanging.

asig -- Input signal.

adel -- Current value of delay in milliseconds. Note that linear functions have no pitch change effects. Fast changing values of *adel* will cause discontinuities in the waveform resulting noise.

Examples

```
f1 0 8192 10 1
ims      =          100              ; Maximum delay time in msec
a1       oscil      10000, 1737, 1   ; Make a signal
a2       oscil      ims/2, 1/p3, 1   ; Make an LFO
a2       =          a2 + ims/2       ; Offset the LFO so that it is positive
a3       vdelay     a1, a2, ims      ; Use the LFO to control delay time
         out        a3
```

Two important points here. First, the delay time must be always positive. And second, even though the delay time can be controlled in k-rate, it is not advised to do so, since sudden time changes will create clicks.

See Also

vdelay3

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

vdelay3

vdelay3 — An variable time delay with cubic interpolation.

Description

vdelay3 is experimental. It is the same as *vdelay* except that it uses cubic interpolation. (New in Version 3.50.)

Syntax

```
ares vdelay3 asig, adel, imaxdel [, iskip]
```

Initialization

imaxdel -- Maximum value of delay in milliseconds. If *adel* gains a value greater than *imaxdel* it is folded around *imaxdel*. This should not happen.

iskip (optional) -- Skip initialization if present and non-zero.

Performance

With this unit generator it is possible to do Doppler effects or chorusing and flanging.

asig -- Input signal.

adel -- Current value of delay in milliseconds. Note that linear functions have no pitch change effects. Fast changing values of *adel* will cause discontinuities in the waveform resulting noise.

Examples

```
f1 0 8192 10 1
ims     =         100              ; Maximum delay time in msec
a1      oscil     10000, 1737, 1   ; Make a signal
a2      oscil     ims/2, 1/p3, 1   ; Make an LFO
a2      =         a2 + ims/2       ; Offset the LFO so that it is positive
a3      vdelay    a1, a2, ims      ; Use the LFO to control delay time
        out       a3
```

Two important points here. First, the delay time must be always positive. And second, even though the delay time can be controlled in k-rate, it is not advised to do so, since sudden time changes will create clicks.

See Also

vdelay

Credits

Author: Paris Smaragdis
MIT, Cambridge
1995

vdelayx

vdelayx — A variable delay opcode with high quality interpolation.

Description

A variable delay opcode with high quality interpolation.

Syntax

```
aout vdelayx ain, adl, imd, iws [, ist]
```

Initialization

aout -- output audio signal

ain -- input audio signal

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist (optional) -- skip initialization if not zero

Performance

This opcode uses high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

  where a is the output gain, and dt is the change of delay time per seconds.
- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayxq, *vdelayxs*, *vdelayxw*, *vdelayxwq*, *vdelayxws*

vdelayxq

vdelayxq — A 4-channel variable delay opcode with high quality interpolation.

Description

A 4-channel variable delay opcode with high quality interpolation.

Syntax

```
aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]
```

Initialization

aout1, aout2, aout3, aout4 -- output audio signals.

ain1, ain2, ain3, ain4 -- input audio signals.

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist (optional) -- skip initialization if not zero

Performance

This opcode uses high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

The multichannel opcodes (eg. *vdelayxq*) allow delaying 2 or 4 variables at once (stereo or quad signals); this is much more efficient than using separate opcodes for each channel.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

  where a is the output gain, and dt is the change of delay time per seconds.
- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayx, *vdelayxs*, *vdelayxw*, *vdelayxwq*, *vdelayxws*

vdelayxs

vdelayxs — A stereo variable delay opcode with high quality interpolation.

Description

A stereo variable delay opcode with high quality interpolation.

Syntax

```
aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]
```

Initialization

aout1, aout2 -- output audio signals

ain1, ain2 -- input audio signals

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist -- skip initialization if not zero

Performance

This opcode uses high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

The multichannel opcodes (eg. *vdelayxq*) allow delaying 2 or 4 variables at once (stereo or quad signals); this is much more efficient than using separate opcodes for each channel.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

  where a is the output gain, and dt is the change of delay time per seconds.
- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayx, *vdelayxq*, *vdelayxw*, *vdelayxwq*, *vdelayxws*

vdelayxw

vdelayxw — Variable delay opcodes with high quality interpolation.

Description

Variable delay opcodes with high quality interpolation.

Syntax

```
aout vdelayxw ain, adl, imd, iws [, ist]
```

Initialization

aout -- output audio signal

ain -- input audio signal

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist -- skip initialization if not zero

Performance

These opcodes use high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

The vdelayxw opcodes change the position of the write tap in the delay line (unlike all other delay ugens that move the read tap), and are most useful for implementing Doppler effects where the position of the listener is fixed, and the sound source is moving.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

  where a is the output gain, and dt is the change of delay time per seconds.

- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayx, *vdelayxq*, *vdelayxs*, *vdelayxwq*, *vdelayxws*

vdelayxwq

vdelayxwq — Variable delay opcodes with high quality interpolation.

Description

Variable delay opcodes with high quality interpolation.

Syntax

```
aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \
      imd, iws [, ist]
```

Initialization

ain1, ain2, ain3, ain4 -- input audio signals

aout1, aout2, aout3, aout4 -- output audio signals

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist -- skip initialization if not zero

Performance

These opcodes use high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

The vdelayxw opcodes change the position of the write tap in the delay line (unlike all other delay ugens that move the read tap), and are most useful for implementing Doppler effects where the position of the listener is fixed, and the sound source is moving.

The multichannel opcodes (eg. *vdelayxq*) allow delaying 2 or 4 variables at once (stereo or quad signals); this is much more efficient than using separate opcodes for each channel.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

where a is the output gain, and dt is the change of delay time per seconds.

- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayx, *vdelayxq*, *vdelayxs*, *vdelayxw*, *vdelayxws*

vdelayxws

vdelayxws — Variable delay opcodes with high quality interpolation.

Description

Variable delay opcodes with high quality interpolation.

Syntax

```
aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```

Initialization

ain1, ain2 -- input audio signals

aout1, aout2 -- output audio signals

adl -- delay time in seconds

imd -- max. delay time (seconds)

iws -- interpolation window size (see below)

ist -- skip initialization if not zero

Performance

These opcodes use high quality (and slow) interpolation, that is much more accurate than the currently available linear and cubic interpolation. The *iws* parameter sets the number of input samples used for calculating one output sample (allowed values are any integer multiply of 4 in the range 4 - 1024); higher values mean better quality and slower speed.

The vdelayxw opcodes change the position of the write tap in the delay line (unlike all other delay ugens that move the read tap), and are most useful for implementing Doppler effects where the position of the listener is fixed, and the sound source is moving.

The multichannel opcodes (eg. *vdelayx*) allow delaying 2 or 4 variables at once (stereo or quad signals); this is much more efficient than using separate opcodes for each channel.

Notes

- Delay time is measured in seconds (unlike in vdelay and vdelay3), and must be a-rate.
- The minimum allowed delay is iws/2 samples.
- Using the same variables as input and output is allowed in these opcodes.
- In vdelayxw*, changing the delay time has some effects on output volume:

  a = 1 / (1 + dt)

where a is the output gain, and dt is the change of delay time per seconds.

- These opcodes are best used in the double-precision version of Csound.

See Also

vdelayx, *vdelayxq*, *vdelayxs*, *vdelayxw*, *vdelayxwq*

vdivv

vdivv — Performs division between two vectorial control signals

Description

Performs division between two vectorial control signals

Syntax

```
vdivv  ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

ksrcoffset - index offset for the source (ifn2) table (Default=0)

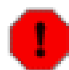
kverbose - Selects whether or not warnings are printed (Default=0)

vdivv divides two vectorial control signals, that is, each element of ifn1 is divided by the corresponding element of ifn2. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 0). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination elements will be set to 0).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are divided). There's an i-rate ver-

sion of this opcode called *vdivv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vdivv opcode. It uses the file *vdivv.csd* [examples/vdivv.csd].

Exemple 537. Example of the vdivv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vdivv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
```

```
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2


i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vdivv_i

vdivv_i — Performs division between two vectorial control signals at init time.

Description

Performs division between two vectorial control signals at init time.

Syntax

```
vdivv_i  ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table (Default=0)

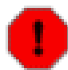
isrcoffset - index offset for the source (ifn2) table (Default=0)

Performance

vdivv_i divides two vectorial control signals, that is, each element of ifn1 is divided by the corresponding element of ifn2. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vdivv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vdelayk

vdelayk — k-rate variable time delay.

Description

Variable delay applied to a k-rate signal

Syntax

```
kout vdelayk  iksig, kdel, imaxdel [, iskip, imode]
```

Initialization

imaxdel - maximum value of delay in seconds.

iskip (optional) - Skip initialization if present and non zero.

imode (optional) - if non-zero it suppresses linear interpolation. While, normally, interpolation increases the quality of a signal, it should be suppressed if using vdelay with discrete control signals, such as, for example, trigger signals.

Performance

kout - delayed output signal

ksig - input signal

kdel - delay time in seconds can be varied at k-rate

vdelayk is similar to *vdelay*, but works at k-rate. It is designed to delay control signals, to be used, for example, in algorithmic composition.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vecdelay

vecdelay — Vectorial Control-rate Delay Paths

Description

Generate a sort of 'vectorial' delay

Syntax

```
vecdelay  ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]
```

Initialization

ifn - number of the table containing the output vector

ifnIn - number of the table containing the input vector

ifnDel - number of the table containing a vector whose elements contain delay values in seconds

ielements - number of elements of the two vectors

imaxdel - Maximum value of delay in seconds.

iskip (optional) - initial disposition of delay-loop data space (see reson). The default value is 0.

Performance

vecdelay is similar to *vdelay*, but it works at k-rate and, instead of delaying a single signal, it delays a vector. *ifnIn* is the input vector of signals, *ifn* is the output vector of signals, and *ifnDel* is a vector containing delay times for each element, expressed in seconds. Elements of *ifnDel* can be updated at k-rate. Each single delay can be different from that of the other elements, and can vary at k-rate. imaxdel sets the maximum delay allowed for all elements of *ifnDel*.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

veloc

veloc — Donne la vélocité d'un évènement MIDI.

Description

Donne la vélocité d'un évènement MIDI.

Syntaxe

```
ival veloc [ilow] [, ihigh]
```

Initialisation

ilow, ihigh -- Limites basse et haute pour le mappage

Exécution

Donne la valeur de l'octet MIDI (0 - 127) pour la vélocité de l'évènement courant.

Exemples

Voici un exemple de l'opcode veloc. Il utilise le fichier *veloc.csd* [examples/veloc.csd].

Exemple 538. Exemple le l'opcode veloc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac           -iadc     -d          -M0  ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o veloc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 veloc

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
```

```
e

</CsScore>
</CsoundSynthesizer>
```

Voir aussi

aftouch, *ampmidi*, *cpsmidi*, *cpsmidib*, *midictrl*, *notnum*, *octmidi*, *octmidib*, *pchbend*, *pchmidi*, *pchmidib*

Crédits

Auteur : Barry L. Vercoe - Mike Berry
MIT - Mills
Mai 1997

Exemple écrit par Kevin Conder.

vexp

vexp — Performs power-of operations between a vector and a scalar

Description

Performs power-of operations between a vector and a scalar

Syntax

```
vexp  ifn, kval, kelements [, kdstoffset] [, kverbose]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

Performance

kval - scalar operand to be processed

kelements - number of elements of the vector

kdstoffset - index offset for the destination table (Optional, default = 0)

kverbose - Selects whether or not warnings are printed (Default=0)

vexp rises *kval* to each element contained in a vector from table *ifn*,starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is processed every control period. Use with care or you will end up with very large (or small) numbers (or use *vexp_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

Examples

Here is an example of the vexp opcode. It uses the file *vexp.csd* [examples/vexp.csd].

Exemple 539. Example of the vexp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp ifn1, ival, ielements, idstoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
```

```
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vexp_i

vexp_i — Performs power-of operations between a vector and a scalar

Description

Performs power-of operations between a vector and a scalar

Syntax

```
vexp_i  ifn, ival, ielements[, idstoffset]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

ielements - number of elements of the vector

ival - scalar value to be added

idstoffset - index offset for the destination table

Performance

vexp_i rises *kval* to each element contained in a vector from table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vexp*.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Examples

Here is an example of the vexp_i opcode. It uses the file *vexp_i.csd* [examples/vexp_i.csd].

Exemple 540. Example of the vexp_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd, *vmult_i*, *vpow_i* and *vexp_i*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vexpseg

vexpseg — Vectorial envelope generator

Description

Generate exponential vectorial segments

Syntax

```
vexpseg   ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

Initialization

ifnout - number of table hosting output vectorial signal

ifn1 - starting vector

ifn2,ifn3,etc. - vector after idurx seconds

idur1 - duration in seconds of first segment.

dur2, idur3, etc. - duration in seconds of subsequent segments.

ielements - number of elements of vectors.

Performance

These opcodes are similar to linseg and expseg, but operate with vectorial signals instead of with scalar signals.

Output is a vectorial control signal hosted by ifnout (that must be previously allocated), while each break-point of the envelope is actually a vector of values. All break-points must contain the same number of elements (ielements).

All these operators are designed to be used together with other opcodes that operate with vectorial signals such as bmscan, vcella, adsynt, adsynt2 etc.

Example

Here is an example of the vexpseg opcode. It uses the files *vexpseg.csd* [examples/vexpseg.csd].

Exemple 541. Example of the vexpseg opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>


sr=44100
```

```
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vexpseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
        kval table kcount, gitable2
        printk 0,kval
        kcount = kcount +1
else
turnoff
endif

endin


</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vexpv

vexpv — Performs exponential operations between two vectorial control signals

Description

Performs exponential operations between two vectorial control signals

Syntax

```
vexpv  ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

ksrcoffset - index offset for the source (ifn2) table (Default=0)

kverbose - Selects whether or not warnings are printed (Default=0)

vexpv elevates each element of ifn2 to the corresponding element of ifn1. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 1). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination elements will be set to 1).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are processed). There's an i-rate version of this opcode called *vexpv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vexpv opcode. It uses the file *vexpv.csd* [examples/vexpv.csd].

Exemple 542. Example of the vexpv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vexpv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 0 16 1

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.002 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vexpv_i

vexpv_i — Performs exponential operations between two vectorial control signals at init time.

Description

Performs exponential operations between two vectorial control signals at init time.

Syntax

```
vexpv_i  ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table (Default=0)

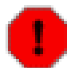
isrcoffset - index offset for the source (ifn2) table (Default=0)

Performance

vexpv_i elevates each element of ifn2 to the corresponding element of ifn1. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will be set to 1). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector elements will be set to 1).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vexpv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vibes

vibes — Modèle physique de la frappe d'un bloc de métal.

Description

La sortie audio est un son de métal frappé comme sur un vibraphone. La méthode est un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

Syntaxe

```
ares vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
```

Initialisation

ihrd -- la dureté de la baguette utilisé pour frapper. Compris entre 0 et 1. 0,5 est une valeur adaptée.

ipos -- l'endroit où le bloc est frappé, compris entre 0 et 1.

imp -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

ivfn -- forme du vibrato, habituellement une table sinus, créée par une fonction

idec -- durée avant la fin de la note lorqu'il y a une atténuation

idoubles (facultatif) -- pourcentage de frappes doubles. La valeur par défaut est de 40%.

itriples (facultatif) -- pourcentage de frappes triples. La valeur par défaut est de 20%.

Exécution

kamp -- Amplitude de la note.

kfreq -- Fréquence de la note.

kvibf -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de de 0 à 12.

kvamp -- Amplitude du vibrato.

Exemples

Voici un exemple de l'opcode vibes. Il utilise les fichiers *vibes.csd* [examples/vibes.csd] et *marmstk1.wav* [examples/marmstk1.wav].

Exemple 543. Exemple de l'opcode vibes.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibes.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; kamp = 20000
  ; kfreq = 440
  ; ihrd = 0.5
  ; ipos = 0.561
  ; imp = 1
  ; kvibf = 6.0
  ; kvamp = 0.05
  ; ivibfn = 2
  ; idec = 0.1

  a1 vibes 20000, 440, 0.5, 0.561, 1, 6.0, 0.05, 2, 0.1

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, the "marmstk1.wav" audio file.
f 1 0 256 1 "marmstk1.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for four seconds.
i 1 0 4
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

marimba

Crédits

Auteur : John ffitch (d'après Perry Cook)
Université de Bath, Codemist Ltd.
Bath, UK

Nouveau dans la version 3.47 de Csound

vibr

vibr — Vibrato contrôlable par l'utilisateur, d'usage plus facile.

Description

Vibrato contrôlable par l'utilisateur, d'usage plus facile.

Syntaxe

```
kout vibr kAverageAmp, kAverageFreq, ifn
```

Initialisation

ifn -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

Exécution

kAverageAmp -- Valeur d'amplitude moyenne du vibrato

kAverageFreq -- Valeur de fréquence moyenne du vibrato (en cps)

vibr est une version de *vibrato* d'usage plus facile. Il a le même moteur de génération que *vibrato*, mais les paramètres correspondant aux arguments d'entrée manquants sont codés en dur sur des valeurs par défaut.

Exemples

Voici un exemple de l'opcode vibr. Il utilise le fichier *vibr.csd* [examples/vibr.csd].

Exemple 544. Exemple de l'opcode vibr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create a vibrato waveform.
  kaverageamp init 7500
  kaveragefreq init 5
```

```
  ifn = 1
  kvamp vibr kaverageamp, kaveragefreq, ifn

  ; Generate a tone including the vibrato.
  a1 oscili 10000+kvamp, 440, 2

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vibrato.
f 1 0 256 10 1
; Table #1, a sine wave for the oscillator.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

jitter, *jitter2*, *vibrato*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

vibrato

vibrato — Génère un vibrato naturel contrôlable par l'utilisateur.

Description

Génère un vibrato naturel contrôlable par l'utilisateur.

Syntaxe

```
kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \
      kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \
      kcpsMaxRate, ifn [, iphs]
```

Initialisation

ifn -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

iphs -- (facultatif) Phase initiale de la table, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

Exécution

kAverageAmp -- Valeur de l'amplitude moyenne du vibrato

kAverageFreq -- Valeur de la fréquence moyenne du vibrato (en cps)

kRandAmountAmp -- Importance de la déviation aléatoire de l'amplitude

kRandAmountFreq -- Importance de la déviation aléatoire de la fréquence

kAmpMinRate -- Fréquence minimale des segments de déviation aléatoire de l'amplitude (en cps)

kAmpMaxRate -- Fréquence maximale des segments de déviation aléatoire de l'amplitude (en cps)

kcpsMinRate -- Fréquence minimale des segments de déviation aléatoire de la fréquence (en cps)

kcpsMaxRate -- Fréquence maximale des segments de déviation aléatoire de la fréquence (en cps)

vibrato produit un vibrato naturel contrôlable par l'utilisateur. Le concept consiste à varier aléatoirement la fréquence et l'amplitude de l'oscillateur générant le vibrato, afin de simuler les irrégularités d'un vibrato réel.

Afin d'avoir un contrôle total de ces variations aléatoires, plusieurs arguments sont présents en entrée. Les variations aléatoires sont obtenues à partir de deux suites séparées de segments, la première contrôlant les déviations d'amplitude, la seconde les déviations de fréquence. La durée moyenne de chaque segment dans chaque suite peut être raccourcie ou allongée par les arguments *kAmpMinRate*, *kAmpMaxRate*, *kcpsMinRate*, *kcpsMaxRate*, et les déviations par rapport aux valeurs d'amplitude et de fréquence moyennes peuvent être ajustées indépendamment au moyen de *kRandAmountAmp* et de *kRandAmountFreq*.

Exemples

Voici un exemple de l'opcode vibrato. Il utilise le fichier *vibrato.csd* [examples/vibrato.csd].

Exemple 545. Example of the vibrato opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibrato.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create a vibrato waveform.
  kaverageamp init 2500
  kaveragefreq init 6
  krandamountamp init 0.3
  krandamountfreq init 0.5
  kampminrate init 3
  kampmaxrate init 5
  kcpsminrate init 3
  kcpsmaxrate init 5
  ifn = 1
  kvamp vibrato kaverageamp, kaveragefreq, krandamountamp, \
                krandamountfreq, kampminrate, kampmaxrate, \
                kcpsminrate, kcpsmaxrate, ifn

  ; Generate a tone including the vibrato.
  a1 oscili 10000+kvamp, 440, 2

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vibrato.
f 1 0 256 10 1
; Table #1, a sine wave for the oscillator.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

jitter, *jitter2*, *vibr*

Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

vincr

vincr — Accumulates audio signals.

Description

vincr increments one audio variable with another signal, i.e. it accumulates output.

Syntax

```
vincr accum, aincr
```

Performance

accum -- audio-rate accumulator variable to be incremented

aincr -- incrementing signal

vincr (variable increment) and *clear* are intended to be used together. *vincr* stores the result of the sum of two audio variables into the first variable itself (which is intended to be used as an accumulator in polyphony). The accumulator is typically a global variable that is used to combine signals from several sources (different instruments or instrument instances) for further processing (for example, via a global effect that reads the accumulator) or for outputting the combined signal by some means other than one of the *out* opcodes (eg. via the *fout* opcode). After the accumulator is used, the accumulator variable should be set to zero by means of the *clear* opcode (or it will explode).

Examples

See the *fout* opcode for an example.

See Also

clear

Credits

Author: Gabriel Maldonado
Italy
1999

New in Csound version 3.56

vlimit

vlimit — Limiting and Wrapping Vectorial Signals

Description

Limits elements of vectorial control signals.

Syntax

```
vlimit  ifn, kmin, kmax, ielements
```

Initialization

ifn - number of the table hosting the vector to be processed

ielements - number of elements of the vector

Performance

kmin - minimum threshold value

kmax - maximum threshold value

vlimit set lower and upper limits on each element of the vector they process.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vlinseg

vlinseg — Vectorial envelope generator

Description

Generate linear vectorial segments

Syntax

```
vlinseg   ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

Initialization

ifnout - number of table hosting output vectorial signal

ifn1 - starting vector

ifn2,ifn3,etc. - vector after idurx seconds

idur1 - duration in seconds of first segment.

dur2, idur3, etc. - duration in seconds of subsequent segments.

ielements - number of elements of vectors.

Performance

These opcodes are similar to linseg and expseg, but operate with vectorial signals instead of with scalar signals.

Output is a vectorial control signal hosted by ifnout (that must be previously allocated), while each break-point of the envelope is actually a vector of values. All break-points must contain the same number of elements (ielements).

All these operators are designed to be used together with other opcodes that operate with vectorial signals such as bmscan, vcella, adsynt, adsynt2 etc.

Example

Here is an example of the vlinseg opcode. It uses the files *vlinseg.csd* [examples/vlinseg.csd].

Exemple 546. Example of the vlinseg opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>


sr=44100
```

```
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vlinseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
        kval table kcount, gitable2
        printk 0,kval
        kcount = kcount +1
else
turnoff
endif

endin


</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vlowres

vlowres — A bank of filters in which the cutoff frequency can be separated under user control.

Description

A bank of filters in which the cutoff frequency can be separated under user control

Syntax

```
ares vlowres asig, kfco, kres, iord, ksep
```

Initialization

iord -- total number of filters (1 to 10)

Performance

asig -- input signal

kfco -- frequency cutoff (not in Hz)

ksep -- frequency cutoff separation for each filter

vlowres (variable resonant lowpass filter) allows a variable response curve in resonant filters. It can be thought of as a bank of lowpass resonant filters, each with the same resonance, serially connected. The frequency cutoff of each filter can vary with the *kcfo* and *ksep* parameters.

Examples

Here is an example of the vlowres opcode. It uses the file *vlowres.csd* [examples/vlowres.csd], and *beats.wav* [examples/beats.wav].

Exemple 547. Example of the vlowres opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vlowres.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Use a nice sawtooth waveform.
  asig vco 32000, 220, 1

  ; Vary the cutoff frequency from 30 to 300 Hz.
  kfco line 30, p3, 300
  kres = 25
  iord = 2
  ksep = 20

  ; Apply the filters.
  avlr vlowres asig, kfco, kres, iord, ksep

  ; It gets loud, so clip the output amplitude to 30,000.
  a1 clip avlr, 1, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Gabriel Maldonado
Italy

Example written by Kevin Conder.

New in Csound version 3.49

vmap

vmap — Maps elements from a vector according to indeces contained in another vector

Description

Maps elements from a vector onto another according to the indeces of a this vector

Syntax

```
vmap  ifn1, ifn2, ielements [,idstoffset, isrcoffset]
```

Initialization

ifn1 - number of the table where the vectorial signal will be copied, and which contains the mapping vector

ifn2 - number of the table hosting the vectorial signal to be copied

ielements - number of elements to process

idstoffset - index offset for destination table (ifn1)

isrcoffset - index offset for source table (ifn2)

Performance

vmap maps elements of *ifn2* according to the values of table *ifn1*. Elements of *ifn1* are treated as indexes of table *ifn2*, so element values of *ifn1* must not exceed the length of *ifn2* table otherwise a Csound will report an error. Elements of *ifn1* are treated as integers, so any fractional part will be truncated. There is no interpolation performed on this operation.

In practice, what happens is that the elements of *ifn1* are used as indeces to *ifn2*, and then are replaced by the corresponding elements from *ifn2*. *ifn1* must be different from *ifn2*, otherwise the results are unpredictable. Csound will produce an init error if they are not.

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Examples

Here is an example of the vmap opcode. It uses the file *vmap.csd* [examples/vmap.csd].

Exemple 548. Example of the vmap opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vmap.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
ksmps = 256
nchnls = 2
gisize = 64

gitable ftgen 0, 0, gisize, 10, 1  ;Table to be processed
gimap1 ftgen  0, 0, gisize, -7, gisize-1, gisize-1, 0 ; Mapping function to reverse table
gimap2 ftgen 0, 0, gisize, -5, 1, gisize-1, gisize-1  ; Mapping function for PWM
gimap3 ftgen 0, 0, gisize, -7, 1, (gisize/2)-1, gisize-1, 1, 1, (gisize/2)-1, gisize-1  ; Double freque

instr 1  ;Hear an oscillator using gitable
asig oscil 10000, 440, gitable
outs asig,asig
endin

instr 2  ;Reverse the table (no sound change, except for a single click
vmap gimap1, gitable, gisize
vcopy_i gitable, gimap1, gisize
turnoff
endin

instr 3  ;Non-interpolated PWM (or phase waveshaping)
vmap gimap2, gitable, gisize
vcopy_i gitable, gimap2, gisize
turnoff
endin

instr 4  ;Double frequency
vmap gimap3, gitable, gisize
vcopy_i gitable, gimap3, gisize
turnoff
endin


</CsInstruments>
<CsScore>
i 1 0 8

i 2 2 1
i 3 4 1
i 4 6 1

e
</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vmirror

vmirror — Limiting and Wrapping Vectorial Signals

Description

'Reflects' elements of vectorial control signals on thresholds.

Syntax

```
vmirror  ifn, kmin, kmax, ielements
```

Initialization

ifn - number of the table hosting the vector to be processed

ielements - number of elements of the vector

Performance

kmin - minimum threshold value

kmax - maximum threshold value

vmirror 'reflects' each element of corresponding vector if it exceeds low or high thresholds.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vmult

vmult — Multiplies a vector in a table by a scalar value.

Description

Multiplies a vector in a table by a scalar value.

Syntax

```
vmult  ifn, kval, kelements [, kdstoffset] [, kverbose]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

Performance

kval - scalar value to be multiplied

kelements - number of elements of the vector

kdstoffset - index offset for the destination table (Optional, default = 0)

kverbose - Selects whether or not warnings are printed (Default=0)

vmult multiplies each element of the vector contained in the table *ifn* by *kval*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is multiplied every control period. Use with care or you will end up with very large numbers (or use *vmult_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

See also

vadd_i, *vadd*, *vmult_i*, *vpow* and *vexp*.

Example

Here is an example of the vmult opcode. It uses the file *vmult-2.csd* [examples/vmult-2.csd].

Exemple 549. Example of the vmult opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult ifn1, ival, ielements, idstoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
```

```
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

Here is another example of the vmult opcode. It uses the file *vmult.csd* [examples/vmult.csd].

Exemple 550. Example of the vmult opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


sr=44100
kr=4410
ksmps=10
nchnls=2

        instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
        endin

        instr 2
vcopy 2, 1, 40000 ;copy vector from sample to empty table
vmult 5, 10000, 262144 ;scale noise to make it audible
vcopy 1, 5, 40000 ;put noise into sample
turnoff
        endin

        instr 3
vcopy 1, 2, 40000 ;put original information back in
turnoff
        endin

</CsInstruments>
<CsScore>
f1   0 262144   -1 "beats.aiff" 0 4 0
f2   0 262144   2  0

f5   0 262144   21  3 30000

i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

See also

vadd_i, *vmult*, *vpow* and *vexp*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vmult_i

vmult_i — Multiplies a vector in a table by a scalar value.

Description

Multiplies a vector in a table by a scalar value.

Syntax

```
vmult_i  ifn, ival, ielements [, idstoffset]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

ival - scalar value to be multiplied

ielements - number of elements of the vector

idstoffset - index offset for the destination table

Performance

vmult_i multiplies each element of the vector contained in the table *ifn* by *ival*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vmult*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Examples

Here is an example of the vmult_i opcode. It uses the file *vmult_i.csd* [examples/vmult_i.csd].

Exemple 551. Example of the vmult_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd, *vadd*, *vmult*, *vpow* and *vexp*.

See also

vadd_i, *vmult*, *vpow_i* and *vexp_i*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vmultv

vmultv — Performs mutiplication between two vectorial control signals

Description

Performs mutiplication between two vectorial control signals

Syntax

```
vmultv  ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

ksrcoffset - index offset for the source (ifn2) table (Default=0)

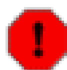
kverbose - Selects whether or not warnings are printed (Default=0)

vmultv multiplies two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The Result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are multiplied). There's an i-rate

version of this opcode called *vmultv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vmultv opcode. It uses the file *vmultv.csd* [examples/vmultv.csd].

Exemple 552. Example of the vmultv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vmultv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin
```

```
</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vmultv_i

vmultv_i — Performs mutiplication between two vectorial control signals at init time.

Description

Performs mutiplication between two vectorial control signals at init time.

Syntax

```
vmultv_i  ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table (Default=0)

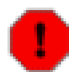
isrcoffset - index offset for the source (ifn2) table (Default=0)

Performance

vmultv_i multiplies two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vmultv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

voice

voice — Simulation d'une voix humaine.

Description

Simulation d'une voix humaine.

Syntaxe

```
ares voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn
```

Initialisation

ifn, *ivfn* -- numéros des deux tables contenant la forme d'onde de la porteuse et la forme d'onde du vibrato. Les fichiers *impuls20.aiff* [examples/impuls20.aiff], *ahh.aiff* [examples/ahh.aiff], *eee.aiff* [examples/eee.aiff] ou *ooo.aiff* [examples/ooo.aiff] conviennent pour la première, et la deuxième peut contenir une sinusoïde. Ces fichiers sont disponibles à *ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/*.

Exécution

kamp -- Amplitude de la note.

kfreq -- Frequency de la note. Elle peut varier pendant l'exécution.

kphoneme -- un entier compris entre 0 et 16, pour choisir les formants des sons :

- « eee », « ihh », « ehh », « aaa »,
- « ahh », « aww », « ohh », « uhh »,
- « uuu », « ooo », « rrr », « lll »,
- « mmm », « nnn », « nng », « ngg ».

Actuellement les phonèmes

- « fff », « sss », « thh », « shh »,
- « xxx », « hee », « hoo », « hah »,
- « bbb », « ddd », « jjj », « ggg »,
- « vvv », « zzz », « thz », « zhh »

ne sont pas disponibles (!)

kform -- gain pour le phonème. Des valeurs entre 0,0 et 1,2 sont recommandées.

kvibf -- fréquence du vibrato en Hertz. On suggère des valeurs entre 0 et 12

kvamp -- amplitude du vibrato

Exemples

Voici un exemple de l'opcode voice. Il utilise les fichiers *voice.csd* [examples/voice.csd] et *impuls20.aiff* [examples/impuls20.aiff].

Exemple 553. Exemple de l'opcode voice.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o voice.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 3
  kfreq = 0.8
  kphoneme = 6
  kform = 0.488
  kvibf = 0.04
  kvamp = 1
  ifn = 1
  ivfn = 2

  av voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

  ; It tends to get loud, so clip voice's amplitude at 30,000.
  a1 clip av, 2, 30000
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, an audio file for the carrier waveform.
f 1 0 256 1 "impuls20.aiff" 0 0 0
; Table #2, a sine wave for the vibrato waveform.
f 2 0 256 10 1

; Play Instrument #1 for a half-second.
i 1 0 0.5
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch (d'après Perry Cook)

Université de Bath, Codemist Ltd.
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

vosim

vosim — Simple vocal simulation based on glottal pulses with formant characteristics.

Description

This opcode produces a simple vocal simulation based on glottal pulses with formant characteristics. Output is a series of sound events, where each each event is composed of a burst of squared sine pulses followed by silence. The VOSIM (VOcal SIMulation) synthesis method was developed by Kaegi and Tempelaars in the 1970's.

Syntax

```
ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]
```

Intialization

ifn - a sound table, normally containing half a period of a sinewave, squared (see notes below).

iskip - (optional) Skip initialization, for tied notes.

Performance

ar - output signal. Note that the output is usually unipolar - positive only.

kamp - output amplitude, the peak amplitude of the first pulse in each burst.

kFund - fundamental pitch, in Herz. Each event is 1/kFund seconds long.

kForm - formant center frequency. Length of each pulse in the burst is 1/kForm seconds.

*kDeca*y - a dampening factor from pulse to pulse. This is subtracted from amplitude on each new pulse.

*kPulseCoun*t - number of pulses in the burst part of each event.

*kPulseFacto*r - the pulse width is multiplied by this value at each new pulse. This results in formant sweeping. If factor is < 1.0, the formant sweeps up, if > 1.0 each new pulse is longer, so the formant sweeps down. The final pitch of the formant is kForm * pow(kPulseFactor, kPulseCount)

The output of *vosim* is a series of sound events, where each each event is composed of a burst of squared sine pulses followed by silence. The total duration of the events determines fundamental frequency. The length of each single pulse in the squared-sine bursts produce a formant frequency band. The width of the formant is determined by rate of silence to pulses (see below). The final result is also shaped by the dampening factor from pulse to pulse.

A small practical problem in using this opcode is that no GEN function will create a squared sine wave out of the box. Something like the following can be used to create the appropriate table from the score.

```
; use GEN09 to create half a sine in table 17

f 17 time size 9  0.5  1 0

; run instr 101 on table 17 for a single init-pass

i 101 0 0 17
```

It can also be done with an instrument writing to an f-table in the orchestra:

```
        ; square each point in table #p4. This should be run as init-only, just once in the performance
instr 101

    index tableng p4

    index = index - 1  ; start from last point
loop:

    ival table index, p4

    ival = ival * ival

    tableiw ival, index, p4

    index = index - 1

    if index < 0 igoto endloop
            igoto loop
endloop:
endin
```

Parameter Limits

The count of pulses multiplied by pulse width should fit in the event length (1/*kFund*). If this is not fulfilled, the algorithm does not break, we just do not start any pulses that would outlast the event. This might introduce a silence at end of event even if none was intended. In consequence, *kForm* should be higher than *kFund*, otherwise only silence is output.

Vosim was created to emulate voice sounds using a model of glottal pulse. Rich sounds can be created by combining several instances of *vosim* with different parameters. One drawback is that the signal is not band-limited. But as the authors point out, attenuation of high-pitch components is -60 dB at 6 times the fundamental frequency. The signal can also be changed by changing the source signal in the lookup table. The technique has historical interest, and can produce rich sound very cheaply (each sample requires only a table lookup and a single multiplication for attenuation).

As stated, formant bandwidth depends on the ratio between pulse burst and silence in an event. But this is not an independent parameter: The fundamental decides event length, and formant center defines the pulse length. It is therefore impossible to guarantee a specific burst/silence ratio, since the burst length has to be an integer multiple of pulse length. The decay of pulses can be used to smooth the transition from N to N+/-1 pulses, but there will still be steps in the spectral profile of output. The example code below shows one approach to this.

All input parameters are k-rate. The input parameters are only used to set up each new event (or grain). Event amplitude is fixed for each event at initialization. In normal parameter ranges, when *ksmps* <500, the k-rate parameters are updated more often than events are created. In any case, no wide-band noise will be injected in the system due to k-rate inputs being updated less often than they are read, but some other artefacts could be created.

The opcode should behave reasonably in the face of all user inputs. Some details:

a. kFund < 0: This is forced to positive - no point in "reversed" events.

b. kFund == 0: This leads to "infinite" length event, ie a pulse burst followed by very long indefinite silence.

c. kForm == 0: This leads to infinite length pulse, so no pulses are generated (i.e. silence).

d. kForm < 0: Table is read backward. If table is symmetric, kform and -kform should give bit-identical outputs.

e. kPulseFactor == 0: Second pulse onwards is zero. See (c).

f. kPulseFactor < 0: Pulses alternately read table forward and reversed.

With asymmetric pulse table there may be some use for negative kForm or negative kPulseFactor.

Examples

Here is an example of the vosim opcode. It uses the file *vosim.csd* [examples/vosim.csd].

Exemple 554. Example of the vosim opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vosim.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 100
nchnls = 1

;################################################
; By Rasmus Ekman 2008

; Square each point in table #p4. This should only be run once in the performance.
instr 10

        index tableng p4
        index = index - 1   ; start from last point
loop:
        ival table index, p4
        ival = ival * ival
        tableiw ival, index, p4
        index = index - 1
        if index < 0 igoto endloop
                igoto loop
endloop:
endin

;################################################

; Main vosim instrument. Sweeps from a fund1/form1 to fund2/form2,
; trying for narrowest formant bandwidth (still quite wide by the looks of it)
; p4:     amp
; p5, p6: fund beg-end
; p7, p8: form beg-end
; p9:     amp decay (ignored)
; p10:    pulse count (ignored - calc internally)
; p11:    pulse length mod
; p12:    skip (for tied events)
; p13:    don't fade out (if followed by tied note)
instr 1
    kamp  init  p4
    ; freq start, end
    kfund  line  p5, p3, p6
    ; formant start, end
    kform  line  p7, p3, p8

        ; Try for constant ratio burst/silence, and narrowest formant bandwidth
        kPulseCount  = (kform / kfund)  ;init p10
        ; Attempt to smooth steps between format bandwidths,
        ; increasing decay before we are forced to a lower pulse count
```

```
        kDecay = kPulseCount/(kform % kfund)  ; init p9
        if (kDecay * kPulseCount) > kamp then
                kDecay = kamp / kPulseCount
        endif
        kDecay = 0.3 * kDecay

        kPulseFactor init p11

;  ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]
    ar1 vosim         kamp, kfund, kform, kDecay, kPulseCount, kPulseFactor, 17, p12

    ; scale amplitude for 16-bit files, with quick fade out
    amp init 20000
    if (p13 != 0) goto nofade
        amp linseg 20000, p3-.02, 20000, .02, 0
nofade:
        out ar1 * amp
endin


</CsInstruments>
<CsScore>

f1       0  32768    9  1    1  0    ; sine wave
f17      0  32768    9  0.5  1  0    ; half sine wave
i10 0 0 17 ; init run only, square table 17

; Vosim score

; Picking some formants from the table in Csound manual

;      p4=amp  fund     form      decay pulses pulsemod [skip] nofade
; tenor a -> e
i1 0  .5  .5   280 240  650  400   .03   5      1
i1 .  .   .3   .   .    1080 1700  .03   5      .
i1 .  .   .2   .   .    2650 2600  .03   5      .
i1 .  .   .15  .   .    2900 3200  .03   5      .

; tenor a -> o
i1 0.6 .2  .5  300 210  650  400   .03   5      1      0      1
i1 .   .   .3  .   .    1080 800   .03   5      .      .      .
i1 .   .   .2  .   .    2650 2600  .03   5      .      .      .
i1 .   .   .15 .   .    2900 2800  .03   5      .      .      .
; tenor o -> aah
i1 .8  .3  .5  210 180  400  650   .03   5      1      1      1
i1 .   .   .3  .   .    800  1080  .03   5      .      .      .
i1 .   .   .2  .   .    2600 2650  .03   5      .      .      .
i1 .   .   .15 .   .    2800 2900  .03   5      .      .      .
; tenor aa -> i
i1 1.1 .2  .5  180 250  650  290   .03   5      1      1      1
i1 .   .   .3  .   .    1080 1870  .03   5      .      .      .
i1 .   .   .2  .   .    2650 2800  .03   5      .      .      .
i1 .   .   .15 .   .    2900 3250  .03   5      .      .      .
; tenor i -> u
i1 1.3 .3  .5  250 270  290  350   .03   5      1      1      0
i1 .   .   .3  .   .    1870 600   .03   5      .      .      .
i1 .   .   .2  .   .    2800 2700  .03   5      .      .      .
i1 .   .   .15 .   .    3250 2900  .03   5      .      .      .

e

</CsScore>
</CsoundSynthesizer>
```

See also

fof, *fof2*

Credits

Author: Rasmus Ekman
March 2008

vphaseseg

vphaseseg — Allows one-dimensional HVS (Hyper-Vectorial Synthesis).

Description

vphaseseg allows one-dimensional HVS (Hyper-Vectorial Synthesis).

Syntax

```
vphaseseg kphase, ioutab, ielems, itab1,idist1,itab2 \
      [,idist2,itab3, ... ,idistN-1,itabN]
```

Initialization

ioutab - number of output table.

ielem - number of elements to process

itab1,...,itabN - breakpoint table numbers

idist1,...,idistN-1 - distances between breakpoints in percentage values

Performance

kphase - phase pointer

vphaseseg returns the coordinates of section points of an N-dimensional space path. The coordinates of section points are stored into an output table. The number of dimensions of the N-dimensional space is determined by the *ielem* argument that is equal to N and can be set to any number. To define the path, user have to provide a set of points of the N-dimensional space, called break-points. Coordinates of each break-point must be contained by a different table. The number of coordinates to insert in each break-point table must obviously equal to *ielem* argument. There can be any number of break-point tables filled by the user.

Hyper-Vectorial Synthesis actually deals with two kinds of spaces. The first space is the N-dimensional space in which the path is defined, this space is called time-variant parameter space (or SPACE A). The path belonging to this space is covered by moving a point into the second space that normally has a number of dimensions smaller than the first. Actually, the point in motion is the projection of corresponding point of the N-dimensional space (could also be considered a section of the path). The second space is called user-pointer-motion space (or SPACE B) and, in the case of *vphaseseg* opcode, has only ONE DIMENSION. Space B is covered by means of *kphase* argument (that is a sort of path pointer), and its range is 0 to 1. The output corresponding to current pointer value is stored in *ioutab* table, whose data can be afterwards used to control any syntesis parameters.

In *vphaseseg*, each break-point is separated from the other by a distance expressed in percentage, where all the path length is equal to the sum of all distances. So distances between breakpoints can be different, differently from kinds of HVS in which space B has more than one dimension, in these cases distance between break-points MUST be THE SAME for all intervals.

See Also

hvs1, *hvs2*, *hvs3*

Credits

Author: Gabriel Maldonado

New in version 5.06

vport

vport — Vectorial Control-rate Delay Paths

Description

Generate a sort of 'vectorial' portamento

Syntax

```
vport ifn, khtime, ielements [, ifnInit]
```

Initialization

ifn - number of the table containing the output vector

ielements - number of elements of the two vectors

ifnInit (optional) - number of the table containing a vector whose elements contain intial portamento values.

Performance

vport is similar to *port*, but operates with vectorial signals, istead of with scalar signals. Each vector element is treated as an indipendent control signal. Input vector input and output vectors are placed in the same table and output vector overrides input vector. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vpow

vpow — Raises each element of a vector to a scalar power

Description

Raises each element of a vector to a scalar power

Syntax

```
vpow  ifn, kval, kelements [, kdstoffset] [, kverbose]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

Performance

kval - scalar value to which the elements of ifn will be raised

kelements - number of elements of the vector

kdstoffset - index offset for the destination table (Optional, default = 0)

kverbose - Selects whether or not warnings are printed (Default=0)

vpow raises each element of the vector contained in the table *ifn* to the power of *kval*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is processed every control period. Use with care or you will end up with very large (or small) numbers (or use *vpow_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

Examples

Here is an example of the vpow opcode. It uses the file *vpow.csd* [examples/vpow.csd].

Exemple 555. Example of the vpow opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow ifn1, ival, ielements, idstoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
```

```
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd_i, *vmult*, *vpow* and *vexp*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vpow_i

vpow_i — Raises each element of a vector to a scalar power

Description

Raises each element of a vector to a scalar power

Syntax

```
vpow_i  ifn, ival, ielements [, idstoffset]
```

Initialization

ifn - number of the table hosting the vectorial signal to be processed

ielements - number of elements of the vector

ival - scalar value to which the elements of ifn will be raised

idstoffset - index offset for the destination table

Performance

vpow_i elevates each element of the vector contained in the table *ifn* to the power of *ival*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vpow*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the intial vectors. If you want to keep initial vector, use *vcopy* or *vcopy_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

Examples

Here is an example of the vpow_i opcode. It uses the file *vpow_i.csd* [examples/vpow_i.csd].

Exemple 556. Example of the vpow_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

See also

vadd_i, *vmult_i*, *vpow* and *vexp_i*.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vpowv

vpowv — Performs power-of operations between two vectorial control signals

Description

Performs power-of operations between two vectorial control signals

Syntax

```
vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

ksrcoffset - index offset for the source (ifn2) table (Default=0)

kverbose - Selects whether or not warnings are printed (Default=0)

vpowv raises each element of ifn1 to the corresponding element of ifn2. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are processed). There's an i-rate version of this opcode called *vpowv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vpowv opcode. It uses the file *vpowv.csd* [examples/vpowv.csd].

Exemple 557. Example of the vpowv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vpowv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
        endin


</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vpowv_i

vpowv_i — Performs power-of operations between two vectorial control signals at init time.

Description

Performs power-of operations between two vectorial control signals at init time.

Syntax

```
vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table

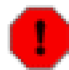
isrcoffset - index offset for the source (ifn2) table

Performance

vpowv_i raises each element of ifn1 to the corresponding element of ifn2. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vpowv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vpvoc

vpvoc — Implements signal reconstruction using an fft-based phase vocoder and an extra envelope.

Description

Implements signal reconstruction using an fft-based phase vocoder and an extra envelope.

Syntax

```
ares vpvoc ktimpnt, kfmod, ifile [, ispecwp] [, ifn]
```

Initialization

ifile -- the pvoc number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

ispecwp (optional, default=0) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

ifn (optional, default=0) -- optional function table containing control information for vpvoc. If *ifn* = 0, control is derived internally from a previous *tableseg* or *tablexseg* unit. Default is 0. (New in Csound version 3.59)

Performance

ktimpnt -- The passage of time, in seconds, through the analysis file. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

kfmod -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

This implementation of *pvoc* was orignally written by Dan Ellis. It is based in part on the system of Mark Dolson, but the pre-analysis concept is new. The spectral extraction and amplitude gating (new in Csound version 3.56) were added by Richard Karpen based on functions in SoundHack by Tom Erbe.

vpvoc is identical to *pvoc* except that it takes the result of a previous *tableseg* or *tablexseg* and uses the resulting function table (passed internally to the *vpvoc*), as an envelope over the magnitudes of the analysis data channels. Optionally, a table specified by *ifn* may be used.

The result is spectral enveloping. The function size used in the *tableseg* should be *framesize/2,* where framesize is the number of bins in the phase vocoder analysis file that is being used by the *vpvoc*. Each location in the table will be used to scale a single analysis bin. By using different functions for *ifn1*, *ifn2*, etc.. in the *tableseg*, the spectral envelope becomes a dynamically changing one. See also *tableseg* and *tablexseg*.

Examples

The following example, using *vpvoc*, shows the use of functions such as

```
f 1 0 256 5 .001 128 1 128 .001
f 2 0 256  5 1 128 .001 128 1
f 3 0 256  7 1 256 1
```

to scale the amplitudes of the separate analysis bins.

```
ktime   line            0, p3,3 ; time pointer, in seconds, into file
        tablexseg       1, p3*.5, 2, p3*.5, 3
apv     vpvoc           ktime,1, "pvoc.file"
```

The result would be a time-varying « spectral envelope » applied to the phase vocoder analysis data. Since this amplifies or attenuates the amount of signal at the frequencies that are paired with the amplitudes which are scaled by these functions, it has the effect of applying very accurate filters to the signal. In this example the first table would have the effect of a band-pass filter, gradually be band-rejected over half the note's duration, and then go towards no modification of the magnitudes over the second half.

See Also

pvoc

Credits

Authors: Dan Ellis and Richard Karpen
Seattle, WA USA
1997

New in version 3.44

vrandh

vrandh — Generates a vector of random numbers stored into a table, holding the values for a period of time.

Description

Generates a vector of random numbers stored into a table, holding the values for a period of time. Generates a sort of 'vectorial band-limited noise'.

Syntax

```
vrandh  ifn,  krange, kcps, ielements [, idstoffset] [, iseed]
      [, isize] [, ioffset]
```

Initialization

ifn - number of the table where the vectorial signal will be generated

ielements - number of elements of the vector

idstoffset - (optional, default=0) -- index offset for the destination table

iseed (optional, default=0.5) -- seed value for the recursive pseudo-random formula. A value between 0 and +1 will produce an initial output of *kamp* * *iseed.* A negative value will cause seed re-initialization to be skipped. A value greater than 1 will seed from system time, this is the best option to generate a different random sequence for each run.

isize (optional, default=0) -- if zero, a 16 bit number is generated. If non-zero, a 31-bit random number is generated. Default is 0.

ioffset - (optional, default=0) -- a base value added to the random result.

Performance

krange - range of random elements (from -krange to krange)

kcps - rate of generated elements in cycles per seconds

This opcode is similar to *randh*, but operates on vectors instead of with scalar values.

Though the argument *isize* defaults to 0, thus using a 16-bit random number generator, using the newer 31-bit algorithm is recommended, as this will produce a random sequence with a longer period (more random numbers before the sequence starts repeating).

The output is a vector contained in *ifn* (that must be previously allocated).

All these operators are designed to be used together with other opocdes that operate with vector such as bmscan, *adsynt* etc.

Note: bmscan not yet available on Canonical Csound

Examples

Here is an example of the vrandh opcode. It uses the file *vrandh.csd* [examples/vrandh.csd].

Exemple 558. Example of the vrandh opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vranh.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Andres Cabrera

sr=44100
ksmps=128
nchnls=2

gitab ftgen 0, 0, 16, -7, 0, 128, 0


instr 1
  krange init p4
  kcps init p5
  ioffset init p6

  kav1 init 0
  kav2 init 0
  kcount init 0

  ;       table   krange  kcps  ielements   idstoffset  iseed  isize ioffset
  vrandh  gitab,  krange, kcps,     3,          3,          2,    0,    ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
;              krange  kcps    ioffset
i 1 0          5 100 1 300
i 1 5          5 300 1 400
i 1 10          5 100 2 1000
i 1 15          5 400 4 1000
i 1 20          5 1000 8 2000
i 1 25          5 250 16 300
e

</CsScore>
</CsoundSynthesizer>
```

See also

vrandi, *randh*

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vrandi

vrandi — Generate a sort of 'vectorial band-limited noise'

Description

Generate a sort of 'vectorial band-limited noise'

Syntax

```
vrandi  ifn,  krange, kcps, ielements [, idstoffset] [, iseed]
      [, isize] [, ioffset]
```

Initialization

ifn - number of the table where the vectorial signal will be generated

ielements - number of elements to process

idstoffset - (optional, default=0) -- index offset for the destination table

iseed (optional, default=0.5) -- seed value for the recursive pseudo-random formula. A value between 0 and +1 will produce an initial output of *kamp * iseed.* A negative value will cause seed re-initialization to be skipped. A value greater than 1 will seed from system time, this is the best option to generate a different random sequence for each run.

isize (optional, default=0) -- if zero, a 16 bit number is generated. If non-zero, a 31-bit random number is generated. Default is 0.

ioffset - (optional, default=0) -- a base value added to the random result.

Performance

krange - range of random elements (from -krange to krange)

kcps - rate of generated elements in cycles per seconds

This opcode is similar to *randi*, but operates on vectors instead of with scalar values.

Though argument *isize* defaults to 0, thus using a 16-bit random number generator, using the newer 31-bit algorithm is recommended, as this will produce a random sequence with a longer period (more random numbers before the sequence starts repeating).

The output is a vector contained in *ifn* (that must be previously allocated).

All these operators are designed to be used together with other opocdes that operate with vector such as bmscan, *adsynt* etc.

Note: bmscan not yet available on Canonical Csound

Examples

Here is an example of the vrandi opcode. It uses the file *vrandi.csd* [examples/vrandi.csd].

Exemple 559. Example of the vrandi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vrandi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera

gitab ftgen 0, 0, 16, -7, 0, 128, 0

instr 1
  krange init p4
  kcps init p5
  ioffset init p6
  ;       table   krange  kcps  ielements   idstoffset  iseed  isize ioffset
  vrandi  gitab,  krange, kcps,     3,         3,          2,   1,    ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>

f 1 0 2048 10 1

;             krange  kcps    ioffset
i 1 0         5 100 1 300
i 1 5         5 5 1 400
i 1 10         5 100 2 1000
i 1 15         5 400 4 1000
i 1 20         5 1000 8 2000
i 1 20         5 300 32 350

e

</CsScore>
</CsoundSynthesizer>
```

See also

vrandh, *randi*

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vstaudio, vstaudiog

vstaudio — VST audio output.

Syntax

```
aout1,aout2 vstaudio instance, [ain1, ain2]
```

```
aout1,aout2 vstaudiog instance, [ain1, ain2]
```

Description

vstaudio and *vstaudiog* are used for sending and receiving audio from a VST plugin.

vstaudio is used within an instrument definition that contains a *vstmidiout* or *vstnote* opcode. It outputs audio for only that one instrument. Any audio remaining in the plugin after the end of the note, for example a reverb tail, will be cut off and should be dealt with using a damping envelope.

vstaudiog (vstaudio global) is used in a separate instrument to process audio from any number of VST notes or MIDI events that share the same VST plugin instance (instance). The *vstaudiog* instrument must be numbered higher than all the instruments receiving notes or MIDI data, and the note controlling the vstplug instrument must have an indefinite duration, or at least a duration as long as the VST plugin is active.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

Performance

aout1, aout2 - the audio output received from the plugin.

ain1, ain2 - the audio input sent to the plugin.

Examples

See *vstmidiout* and *vstparamset* for examples.

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstbankload

vstbankload — Loads parameter banks to a VST plugin.

Syntax

```
vstbankload instance, ipath
```

Description

vstbankload is used for loading parameter banks to a VST plugin.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

ipath - the full pathname of the parameter bank (`.fxb` file).

Examples

Exemple 560. Example for vstbankload

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 4
vstbankload gihandle1,"c:/vstplugins/cheeze/chengo'scheese.fxb"
vstinfo gihandle1
endin

/* sco */
i 3 0 21
i4 1 1 57 32
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstedit

vstedit — Opens the GUI editor widow for a VST plugin.

Syntax

```
vstedit instance
```

Description

vstedit opens the custom GUI editor widow for a VST plugin. Note that not all VST plugins have custom GUI editors.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstinit

vstinit — Load a VST plugin into memory for use with the other vst4cs opcodes.

Syntax

```
instance vstinit ilibrarypath [,iverbose]
```

Description

vstinit is used to load a VST plugin into memory for use with the other vst4cs opcodes. Both VST effects and instruments (synthesizers) can be used.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

ilibrarypath - the full path to the vst plugin shared library (dll, on Windows). Remember to use '/' instead of '\' as separator.

iverbose - show plugin information and parameters when loading.

Examples

Exemple 561. Loading a VST Plugin

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 1
      giHandle2 vstinit "c:/vstplugins/crazy diamonds.dll",1
endin



/* sco */
i 1 0 1
e
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstinfo

vstinfo — Displays the parameters and the programs of a VST plugin.

Syntax

```
vstinfo instance
```

Description

vstinfo displays the parameters and the programs of a VST plugin.

Note: The *verbose* flag in *vstinit* gives the same information as *vstinfo*. *vstinfo* is useful after loading parameter banks, or when the plugin changes paramters dynamically.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

Examples

Exemple 562. Example for vstinfo

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 4
vstbankload gihandle1,"c:/vstplugins/cheeze/chengo'scheese.fxb"
vstinfo gihandle1
endin


/* sco */
i 3 0 21
i4 1 1 57 32
e
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstmidiout

vstmidiout — Sends MIDI information to a VST plugin.

Syntax

```
vstmidiout instance, kstatus, kchan, kdata1, kdata2
```

Description

vstmidiout is used for sending MIDI information to a VST plugin.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

Performance

kstatus - the type of midi message to be sent. Currently noteon (144), note off (128), Control Change (176), Program change (192), Aftertouch (208) and Pitch Bend (224) are supported.

kchan - the MIDI channel transmitted on.

kdata1, kdata2 - the MIDI data pair, which varies depending on kstatus. e.g. note/velocity for note on and note off, Controller number/value for control change.

Examples

Exemple 563. Example for vstmidiout

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 3
ain1 = 0
ab1, ab2 vstaudio gihandle1, ain1, ain1
outs ab1, ab2
endin
instr 4
vstmidiout gihandle1,144,1,p4,p5
endin
```

```
/* sco */
i 3 0 21
i4 1 1 57 32
i4 3 1 60 100
i4 5 1 62 100
i4 7 1 64 100
i4 9 1 65 100
i4 11 1 67 100
i4 13 1 69 100
i4 15 3 71 100
```

```
i4 18 3 72 100
e
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstnote

vstnote — Sends a MIDI note with definite duration to a VST plugin.

Syntax

```
vstnote instance, kchan, knote, kveloc, kdur
```

Description

vstnote sends a MIDI note with definite duration to a VST plugin.

Initialization

instance - the number which identifies the plugin generated by *vstinit*.

Performance

kchan - The midi channel to trasnmit the note on.

knote - The midi note number to send.

kveloc - The midi note's velocity.

kdur - The midi note's duration in seconds.

Note: Be sure the instrument containing vstnote is not finished before the duration of the note, otherwise you'll have a 'hung' note.

Examples

Exemple 564. Example for vstnote

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle5 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 3
ain1 = 0
ga1, ga2 vstplugg gihandle5, ain1, ain1
endin
instr 4
vstnote giHandle5, 1, p4, p5, p3
endin
instr 10
outs ga1, ga2
endin


/* sco */
i 3 0 21
i 10 0 21
i4 1 3 57 55
```

```
i4 3 3 60 100
i4 5 3 62 100
i4 7 3 64 100
i4 9 2 65 100
i4 11 1 67 100
i4 13 1 69 100
i4 15 3 71 100
i4 18 3 72 100
```

See Also

vstinit

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstparamset,vstparamget

vstparamset — Used for parameter comunication to and from a VST plugin.

Syntax

```
vstparamset instance, kparam, kvalue
```

```
kvalue vstparamget instance, kparam
```

Description

vstparamset and *vstparamget* are used for parameter comunication to and from a VST plugin.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

Performance

kparam - The number of the parameter to set or get.

kvalue - the value to set, or the the value returned by the plugin.

Parameters vary according to the plugin. To find out what parameters are available, use the verbose option when loading the plugin with vstinit.

Examples

Exemple 565. Example of vstparamset

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 3
ain1 = 0
ab1, ab2 vstaudio gihandle1, ain1, ain1
outs ab1, ab2
endin
instr 4
vstmidiout gihandle1,144,1,p4,p5
kline line 0,p3,1
vstparamset gihandle1, 3, kline
endin
```

```
/* sco */
i 3 0 21
i4 1 1 57 32
i4 3 1 60 100
i4 5 1 62 100
i4 7 1 64 100
```

```
i4 9 1 65 100
i4 11 1 67 100
i4 13 1 69 100
i4 15 3 71 100
i4 18 3 72 100
e
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vstprogset

vstprogset — Loads parameter banks to a VST plugin.

Syntax

```
vstprogset instance, kprogram
```

Description

vstprogset sets one of the programs in an `.fxb` bank.

Initialization

instance - the number which identifies the plugin, to be passed to other vst4cs opcodes.

kprogram - the number of the program to set.

Examples

Exemple 566. Usage of vstprogset

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
giHandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 4
vstbankload gihandle1,"c:/vstplugins/cheeze/chengo'scheese.fxb"
vstprogset gihandle1, 4
vstinfo gihandle1
endin


/* sco */
i 3 0 21
i4 1 1 57 32
e
```

Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

vsubv

vsubv — Performs subtraction between two vectorial control signals

Description

Performs subtraction between two vectorial control signals

Syntax

```
vsubv  ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

Performance

kelements - number of elements of the two vectors

kdstoffset - index offset for the destination (ifn1) table (Default=0)

ksrcoffset - index offset for the source (ifn2) table (Default=0)

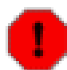
kverbose - Selects whether or not warnings are printed (Default=0)

vsubv subtracts two vectorial control signals, that is, each element of ifn2 is subrtacted from the corresponding element of ifn1. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are subtracted). There's an i-rate

version of this opcode called *vsubv_i*.

Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
    instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
    endin
```

All these operators (*vaddv*,*vsubv*,*vmultv*,*vdivv*,*vpowv*,*vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Examples

Here is an example of the vsubv opcode. It uses the file *vsubv.csd* [examples/vsubv.csd].

Exemple 567. Example of the vsubv opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>


sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vsubv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
```

```
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2


i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vsubv_i

vsubv_i — Performs subtraction between two vectorial control signals at init time.

Description

Performs subtraction between two vectorial control signals at init time.

Syntax

```
vsubv_i  ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

Initialization

ifn1 - number of the table hosting the first vector to be processed

ifn2 - number of the table hosting the second vector to be processed

ielements - number of elements of the two vectors

idstoffset - index offset for the destination (ifn1) table (Default=0)

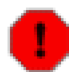
isrcoffset - index offset for the source (ifn2) table (Default=0)

Performance

vsubv_i subtracts two vectorial control signals, that is, each element of ifn2 is subrtacted from the corresponding element of ifn1. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vsubv*.

All these operators (*vaddv_i*,*vsubv_i*,*vmultv_i*,*vdivv_i*,*vpowv_i*,*vexpv_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

vtable1k

vtable1k — Read a vector (several scalars simultaneously) from a table.

Description

This opcode reads vectors from tables at k-rate.

Syntax

```
vtable1k  kfn,kout1 [, kout2, kout3, .... , koutN ]
```

Performance

kfn - table number

kout1...koutN - output vector elements

vtable1k is a reduced version of *vtablek*, it only allows to access the first vector (it is equivalent to vtablek with kndx = zero, but a bit faster). It is useful to easily and quickly convert a set of values stored in a table into a set of k-rate variables to be used in normal opocodes, instead of using individual *table* opcodes for each value.

Note

vtable1k is an unusual opcode as it produces its output on the right side arguments of the opcode.

Examples

Here is an example of the vtable1k opcode. It uses the files *vtable1k.csd* [examples/vtable1k.csd].

Exemple 568. Example of the vtable1k opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

giElem init 13
giOutTab ftgen 1,0,128, 2,         0
giFreqTab ftgen 2,0,128,-7,         1,giElem, giElem+1
giSine ftgen 3,0,256,10, 1

        FLpanel "This Panel contains a Slider Bank",500,400
        FLslidBnk "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, 360, 600, 100, 10
        FLpanel_end
```

```
        FLrun

        instr 1

kout1 init 0
kout2 init 0
kout3 init 0
kout4 init 0
kout5 init 0
kout6 init 0
kout7 init 0
kout8 init 0

vtable1k  giOutTab, kout1 , kout2, kout3, kout4, kout5 , kout6, kout7, kout8
kmodindex1=         2 * db(kout1 * 80 )
kmodindex2= 2 * db(kout2 * 80 )
kmodindex3= 2 * db(kout3 * 80 )
kamp = 50 * db(kout4 * 70 )
kfreq1 = 1.1 * octave(kout5 * 10)
kfreq2 = 1.1 * octave(kout6 * 10)
kfreq3 = 1.1 * octave(kout7 * 10)
kfreq4 = 30 * octave(kout8 * 8)

amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,       kfreq4+amod1+amod2+amod3, giSine


        outs aout, aout
        endin


</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>
```

See Also

vtablek

Credits

Written by Gabriel Maldonado.

New in Csound 5.06

vtablei

vtablei — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables.

Syntax

```
vtablei  indx, ifn, interp, ixmode, iout1 [, iout2, iout3, .... , ioutN ]
```

Initialization

indx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

ifn - table number

iout1...ioutN - output vector elements

ixmode - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

interp - vtable (vector table) family of opcodes allows the user to switch beetween interpolated or non-interpolated output by means of the interp argument.

Performance

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (iout1 , iout2, iout3, ... ioutN).

vtable (vector table) family of opcodes allows the user to switch beetween interpolated or non-interpolated output by means of the interp argument.

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtable, in order to correct eventual out-of-range values.

Note

Notice that *vtablei*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

Examples

Here is an example of the vtablei opcode. It uses the files *vtablei.csd* [examples/vtablei.csd]

Exemple 569. Example of the vtablei opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =        44100
kr      =        100
ksmps   =        441
nchnls  =        2

gindx init 0

       instr    1
kindex init 0
ktrig metro 0.5
if ktrig = 0 goto noevent
event "i", 2, 0, 0.5, kindex
kindex = kindex + 1
noevent:

        endin

        instr 2
iout1 init 0
iout2 init 0
iout3 init 0
iout4 init 0
indx = p4
vtablei  indx, 1, 1, 0, iout1,iout2, iout3, iout4
print iout1, iout2, iout3, iout4
turnoff
        endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20

</CsScore>
</CsoundSynthesizer>
```

See also

vtablea, *vtablek*, *vtabi*, *vtablewi*, *vtabwi*,

Credits

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vtablek

vtablek — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables at k-rate.

Syntax

```
vtablek  kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]
```

Initialization

ixmode - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

kinterp - switch beetween interpolated or non-interpolated output. 0 -> non-interpolation , non-zero -> interpolation activated

Performance

kndx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

kfn - table number

kout1...koutN - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (kout1 , kout2, kout3, ... koutN).

vtablek allows the user to switch beetween interpolated or non-interpolated output at k-rate by means of kinterp argument.

vtablek allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtable, in order to correct eventual out-of-range values.

Note

Notice that *vtablek*'s output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output argu-

ments such as *fin* or *trigseq*).

Examples

Here is an example of the vtablek opcode. It uses the files *vtablek.csd* [examples/vtablek.csd].

Exemple 570. Example of the vtablek opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =        44100
kr      =        100
ksmps   =        441
nchnls =         2

gkindx init -1

       instr    1
kindex init 0
ktrig metro 0.5
if ktrig = 0 goto noevent
gkindx = gkindx + 1
noevent:

        endin

        instr 2
kout1 init 0
kout2 init 0
kout3 init 0
kout4 init 0
vtablek  gkindx, 1, 1, 0, kout1,kout2, kout3, kout4
printk2 kout1
printk2 kout2
printk2 kout3
printk2 kout4
        endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20
i 2 0 20
</CsScore>
</CsoundSynthesizer>
```

See also

vtablea, *vtablei*, *vtabk*, *vtablewk*, *vtabwk*,

Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vtablea

vtablea — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables at a-rate.

Syntax

```
vtablea  andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]
```

Initialization

ixmode - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

Performance

andx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

kfn - table number

kinterp - switch beetween interpolated or non-interpolated output. 0 -> non-interpolation , non-zero -> interpolation activated

aout1...aoutN - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (aout1 , aout2, aout3, ... aoutN).

vtablea allows the user to switch beetween interpolated or non-interpolated output at k-rate by means of kinterp argument.

vtablea allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtablea, in order to correct eventual out-of-range values.

Note

Notice that *vtablea*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output

arguments such as *fin* or *trigseq*).

See also

vtablek, *vtablei*, *vtaba*, *vtablewa*, *vtabwa*,

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtablewi

vtablewi — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at init time.

Syntax

```
vtablewi  indx, ifn, ixmode, inarg1 [, inarg2, inarg3 , .... , inargN ]
```

Initialization

indx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

ifn - table number

ixmode - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

inarg1...inargN - output vector elements

Performance

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (inarg1 , inarg2, inarg3, ... inargN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewi*, in order to correct eventual out-of-range values.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtablewk

vtablewk — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at k-rate.

Syntax

```
vtablewk  kndx, kfn, ixmode, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
```

Initialization

ixmode - index data mode. The default value is 0. == 0 index is treated as a raw table location, == 1 index is normalized (0 to 1).

Performance

kndx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

kfn - table number

kinarg1...kinargN - output vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (kinarg1 , kinarg2, kinarg3, ... kinargN).

vtablewk allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewk*, in order to correct eventual out-of-range values.

Examples

Here is an example of the vtablewk opcode. It uses the files *vtablewk.csd* [examples/vtablewk.csd].

Exemple 571. Example of the vtablewk opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -b441 -B441
</CsOptions>
<CsInstruments>
```

```
sr=44100
kr=4410
ksmps=10
nchnls=2

        instr 1
vcopy
ar random 0, 1
vtablewa ar
out ar,ar
        endin


</CsInstruments>
<CsScore>
f1   0 262144   -1 "beats.wav" 0 4 0
f2   0 262144   2  0


i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vtablewa

vtablewa — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at a-rate.

Syntax

```
vtablewa  andx, kfn, ixmode, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
```

Initialization

ixmode - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

Performance

andx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

kfn - table number

ainarg1...ainargN - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (ainarg1 , ainarg2, ainarg3, ... ainargN).

vtablewa allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewa*, in order to correct eventual out-of-range values.

Examples

Here is an example of the vtablewa opcode. It uses the files *vtablewa.csd* [examples/vtablewa.csd].

Exemple 572. Example of the vtablek opcode.

```
<CsoundSynthesizer>
<CsOptions>
```

```
;-ovtablewa.wav -W -b441 -B441
-odac -b441 -B441
</CsOptions>
<CsInstruments>


sr=44100
kr=441
ksmps=100
nchnls=2

        instr 1
ilen = ftlen (1)

knew1 oscil 10000, 440, 3
knew2 oscil 15000, 440, 3, 0.5
kindex phasor 0.3
asig oscil 1, sr/ilen , 1
vtablewk kindex*ilen, 1, 0, knew1, knew2
out asig,asig
        endin


</CsInstruments>
<CsScore>
f1  0 262144   -1 "beats.wav" 0 4 0
f2  0 262144   2  0
f3  0 1024  10 1

i1 0 10
</CsScore>
</CsoundSynthesizer>
```

Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

vtabi

vtabi — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables.

Syntax

```
vtabi  indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]
```

Initialization

indx - Index into f-table, either a positive number range matching the table length

ifn - table number

iout1...ioutN - output vector elements

Performance

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (iout1 , iout2, iout3, ... ioutN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtable, in order to correct eventual out-of-range values.

Notice that vtabi output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as fin or trigseq).

The **vtab** family is similar to **vtable**, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.

Note

Notice that *vtabi*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

Examples

For an example of the vtabi opcode usage, see *vtablei*.

See also

vtabk, *vtaba*, *vtablei*, *vtablewi*, *vtabwi*,

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtabk

vtabk — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables at k-rate.

Syntax

```
vtabk  kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]
```

Initialization

ifn - table number

Performance

kndx - Index into f-table, either a positive number range matching the table length

kout1...koutN - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.) . The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (kout1 , kout2, kout3, ... koutN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtable, in order to correct eventual out-of-range values.

Notice that *vtabk* output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as fin or trigseq).

The **vtab** family is similar to **vtable**, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.

Note

Notice that *vtabk*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

Examples

For an example of the vtabk opcode usage, see *vtablek*.

See also

vtabi, *vtaba*, *vtablek*, *vtablewk*, *vtabwk*,

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtaba

vtaba — Read vectors (from tables -or arrays of vectors).

Description

This opcode reads vectors from tables at a-rate.

Syntax

```
vtaba  andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]
```

Initialization

ifn - table number

Performance

andx - Index into f-table, either a positive number range matching the table length

aout1...aoutN - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (aout1 , aout2, aout3, ... aoutN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtaba, in order to correct eventual out-of-range values.

Notice that **vtaba** output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as fin or trigseq).

The **vtab** family is similar to the **vtable** family, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.

Note

Notice that *vtaba*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

Examples

The usage of *vtaba* is similar to *vtablek*.

See also

vtabk, *vtabi*, *vtablea*, *vtablewa*, *vtabwa*,

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtabwi

vtabwi — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at init time.

Syntax

```
vtabwi  indx, ifn, inarg1 [, inarg2, inarg3 , .... , inargN ]
```

Initialization

indx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

ifn - table number

inarg1...inargN - output vector elements

Performance

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (inarg1 , inarg2, inarg3, ... inargN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwi*, in order to correct eventual out-of-range values.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtabwk

vtabwk — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at a-rate.

Syntax

```
vtabwk  kndx, ifn, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
```

Initialization

ifn - table number

Performance

kndx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0). *kinarg1...kinargN* - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.) . The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (kinarg1 , kinarg2, kinarg3, ... kinargN).

vtabwk allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwk*, in order to correct eventual out-of-range values.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vtabwa

vtabwa — Write vectors (to tables -or arrays of vectors).

Description

This opcode writes vectors to tables at a-rate.

Syntax

```
vtabwa  andx, ifn, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
```

Initialization

ifn - table number

Performance

andx - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

ainarg1...ainargN - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (ainarg1 , ainarg2, ainarg3, ... ainargN).

vtabwa allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occurr), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwa*, in order to correct eventual out-of-range values.

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

vwrap

vwrap — Limiting and Wrapping Vectorial Signals

Description

Wraps elements of vectorial control signals.

Syntax

```
vwrap  ifn, kmin, kmax, ielements
```

Initialization

ifn - number of the table hosting the vector to be processed

ielements - number of elements of the vector

Performance

kmin - minimum threshold value

kmax - maximum threshold value

vwrap wraps around each element of corresponding vector if it exceeds low or high thresholds.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

Note: bmscan not yet available on Canonical Csound

Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

waveset

waveset — A simple time stretch by repeating cycles.

Description

A simple time stretch by repeating cycles.

Syntax

```
ares waveset ain, krep [, ilen]
```

Initialization

ilen (optional, default=0) -- the length (in samples) of the audio signal. If *ilen* is set to 0, it defaults to half the given note length (p3).

Performance

ain -- the input audio signal.

krep -- the number of times the cycle is repeated.

The input is read and each complete cycle (two zero-crossings) is repeated krep times.

There is an internal buffer as the output is clearly slower that the input. Some care is taken if the buffer is too short, but there may be strange effects.

Examples

Here is an example of the waveset opcode. It uses the file *waveset.csd* [examples/waveset.csd], and *beats.wav* [examples/beats.wav].

Exemple 573. Example of the waveset opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o waveset.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1 - play an audio file.
instr 1
  asig soundin "beats.wav"
  out asig
endin


; Instrument #2 - stretch the audio file with waveset.
instr 2
  asig soundin "beats.wav"
  a1 waveset asig, 2

  out a1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for four seconds.
i 2 3 4
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch
February 2001

Example written by Kevin Conder.

New in version 4.11

weibull

weibull — Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement).

Description

Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement). C'est un générateur de bruit de classe x.

Syntaxe

```
ares weibull ksigma, ktau
```

```
ires weibull ksigma, ktau
```

```
kres weibull ksigma, ktau
```

Exécution

ksigma -- contrôle l'échelle de l'étalement de la distribution.

ktau -- s'il est supérieur à un, les nombres proches de *ksigma* sont favorisés. S'il est inférieur à un, les petites valeurs sont favorisées. S'il est égal à 1, la distribution est exponentielle. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

Exemples

Voici un exemple de l'opcode weibull. Il utilise le fichier *weibull.csd* [examples/weibull.csd].

Exemple 574. Exemple de l'opcode weibull.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o weibull.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number in a Weibull distribution.
  ; ksigma = 1
  ; ktau = 1

  i1 weibull 1, 1

  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 1.834
```

Voir Aussi

seed, *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Exemple écrit par Kevin Conder.

wgbow

wgbow — Creates a tone similar to a bowed string.

Description

Audio output is a tone similar to a bowed string, using a physical model developed from Perry Cook, but re-coded for Csound.

Syntax

```
ares wgbow kamp, kfreq, kpres, krat, kvibf, kvamp, ifn [, iminfreq]
```

Initialization

ifn -- table of shape of vibrato, usually a sine table, created by a function

iminfreq (optional) -- lowest frequency at which the instrument will play. If it is omitted it is taken to be the same as the initial *kfreq*. If *iminfreq* is negative, initialization will be skipped.

Performance

A note is played on a string-like instrument, with the arguments as below.

kamp -- amplitude of note.

kfreq -- frequency of note played.

kpres -- a parameter controlling the pressure of the bow on the string. Values should be about 3. The useful range is approximately 1 to 5.

krat -- the position of the bow along the string. Usual playing is about 0.127236. The suggested range is 0.025 to 0.23.

kvibf -- frequency of vibrato in Hertz. Suggested range is 0 to 12

kvamp -- amplitude of the vibrato

Examples

Here is an example of the wgbow opcode. It uses the file *wgbow.csd* [examples/wgbow.csd].

Exemple 575. Example of the wgbow opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
```

```
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgbow.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 31129.60
  kfreq = 440
  kpres = 3.0
  krat = 0.127236
  kvibf = 6.12723
  ifn = 1

  ; Create an amplitude envelope for the vibrato.
  kv linseg 0, 0.5, 0, 1, 1, p3-0.5, 1
  kvamp = kv * 0.01

  a1 wgbow kamp, kfreq, kpres, krat, kvibf, kvamp, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 3.47

wgbowedbar

wgbowedbar — A physical model of a bowed bar.

Description

A physical model of a bowed bar, belonging to the Perry Cook family of waveguide instruments.

Syntax

```
ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvel] \
      [, ibowpos] [, ilow]
```

Initialization

iconst (optional, default=0) -- an integration constant. Default is zero.

itvel (optional, default=0) -- either 0 or 1. When *ktvel* = 0, the bow velocity follows an ADSR style trajectory. When *ktvel* = 1, the value of the bow velocity decays in an exponentially.

ibowpos (optional, default=0) -- the position on the bow, which affects the bow velocity trajectory.

ilow (optional, default=0) -- lowest frequency required

Performance

kamp -- amplitude of signal

kfreq -- frequency of signal

kpos -- position of the bow on the bar, in the range 0 to 1

kbowpres -- pressure of the bow (as in *wgbowed*)

kgain -- gain of filter. A value of about 0.809 is suggested.

Examples

Here is an example of the wgbowedbar opcode. It uses the file *wgbowedbar.csd* [examples/wgbowedbar.csd].

Exemple 576. Example of the wgbowedbar opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o wgbowedbar.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

  instr 1
; pos       =         [0, 1]
; bowpress =          [1, 10]
; gain      =         [0.8, 1]
; intr      =         [0,1]
; trackvel =          [0, 1]
; bowpos    =         [0, 1]

  kb       line 0.5, p3, 0.1
  kp       line 0.6, p3, 0.7
  kc       line 1, p3, 1

  a1       wgbowedbar p4, cpspch(p5), kb, kp, 0.995, p6, 0

           out a1
           endin


</CsInstruments>
<CsScore>

  i1       0   3 32000 7.00 0
  e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 4.07

wgbrass

wgbrass — Creates a tone related to a brass instrument.

Description

Audio output is a tone related to a brass instrument, using a physical model developed from Perry Cook, but re-coded for Csound.

Syntax

```
ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn [, iminfreq]
```

Initialization

iatt -- time taken to reach full pressure

ifn -- table of shape of vibrato, usually a sine table, created by a function

iminfreq -- lowest frequency at which the instrument will play. If it is omitted it is taken to be the same as the initial *kfreq*. If *iminfreq* is negative, initialization will be skipped.

Performance

A note is played on a brass-like instrument, with the arguments as below.

kamp -- Amplitude of note.

kfreq -- Frequency of note played.

ktens -- lip tension of the player. Suggested value is about 0.4

kvibf -- frequency of vibrato in Hertz. Suggested range is 0 to 12

kvamp -- amplitude of the vibrato

NOTE

This is rather poor, and at present uncontrolled. Needs revision, and possibly more parameters.

Examples

Here is an example of the wgbrass opcode. It uses the file *wgbrass.csd* [examples/wgbrass.csd].

Exemple 577. Example of the wgbrass opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgbrass.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 31129.60
  kfreq = 440
  ktens = 0.4
  iatt = 0.1
  kvibf = 6.137
  ifn = 1

  ; Create an amplitude envelope for the vibrato.
  kvamp line 0, p3, 0.5

  a1 wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 3.47

wgclar

wgclar — Creates a tone similar to a clarinet.

Description

Audio output is a tone similar to a clarinet, using a physical model developed from Perry Cook, but re-coded for Csound.

Syntax

```
ares wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq]
```

Initialization

iatt -- time in seconds to reach full blowing pressure. 0.1 seems to correspond to reasonable playing. A longer time gives a definite initial wind sound.

idetk -- time in seconds taken to stop blowing. 0.1 is a smooth ending

ifn -- table of shape of vibrato, usually a sine table, created by a function

iminfreq (optional) -- lowest frequency at which the instrument will play. If it is omitted it is taken to be the same as the initial *kfreq*. If *iminfreq* is negative, initialization will be skipped.

Performance

A note is played on a clarinet-like instrument, with the arguments as below.

kamp -- Amplitude of note.

kfreq -- Frequency of note played.

kstiff -- a stiffness parameter for the reed. Values should be negative, and about -0.3. The useful range is approximately -0.44 to -0.18.

kngain -- amplitude of the noise component, about 0 to 0.5

kvibf -- frequency of vibrato in Hertz. Suggested range is 0 to 12

kvamp -- amplitude of the vibrato

Examples

Here is an example of the wgclar opcode. It uses the file *wgclar.csd* [examples/wgclar.csd].

Exemple 578. Example of the wgclar opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgclar.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp init 31129.60
  kfreq = 440
  kstiff = -0.3
  iatt = 0.1
  idetk = 0.1
  kngain = 0.2
  kvibf = 5.735
  kvamp = 0.1
  ifn = 1

  a1 wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 3.47

wgflute

wgflute — Creates a tone similar to a flute.

Description

Audio output is a tone similar to a flute, using a physical model developed from Perry Cook, but re-coded for Csound.

Syntax

```
ares wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq] [, ijetrf] [, iendrf]
```

Initialization

iatt -- time in seconds to reach full blowing pressure. 0.1 seems to correspond to reasonable playing.

idetk -- time in seconds taken to stop blowing. 0.1 is a smooth ending

ifn -- table of shape of vibrato, usually a sine table, created by a function

iminfreq (optional) -- lowest frequency at which the instrument will play. If it is omitted it is taken to be the same as the initial kfreq. If *iminfreq* is negative, initialization will be skipped.

ijetrf (optional, default=0.5) -- amount of reflection in the breath jet that powers the flute. Default value is 0.5.

iendrf (optional, default=0.5) -- reflection coefficient of the breath jet. Default value is 0.5. Both *ijetrf* and *iendrf* are used in the calculation of the pressure differential.

Performance

kamp -- Amplitude of note.

kfreq -- Frequency of note played. While it can be varied in performance, I have not tried it.

kjet -- a parameter controlling the air jet. Values should be positive, and about 0.3. The useful range is approximately 0.08 to 0.56.

kngain -- amplitude of the noise component, about 0 to 0.5

kvibf -- frequency of vibrato in Hertz. Suggested range is 0 to 12

kvamp -- amplitude of the vibrato

Examples

Here is an example of the wgflute opcode. It uses the file *wgflute.csd* [examples/wgflute.csd].

Exemple 579. Example of the wgflute opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgflute.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 31129.60
  kfreq = 440
  kjet = 0.32
  iatt = 0.1
  idetk = 0.1
  kngain = 0.15
  kvibf = 5.925
  kvamp = 0.05
  ifn = 1

  a1 wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 3.47

wgpluck

wgpluck — A high fidelity simulation of a plucked string.

Description

A high fidelity simulation of a plucked string, using interpolating delay-lines.

Syntax

```
ares wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite
```

Initialization

icps -- frequency of plucked string

iamp -- amplitude of string pluck

iplk -- point along the string, where it is plucked, in the range of 0 to 1. 0 = no pluck

idamp -- damping of the note. This controls the overall decay of the string. The greater the value of idamp1, the faster the decay. Negative values will cause an increase in output over time.

ifilt -- control the attenuation of the filter at the bridge. Higher values cause the higher harmonics to decay faster.

Performance

kpick -- proportion of the way along the point to sample the output.

axcite -- a signal which excites the string.

A string of frequency *icps* is plucked with amplitude *iamp* at point *iplk*. The decay of the virtual string is controlled by *idamp* and *ifilt* which simulate the bridge. The oscillation is sampled at the point *kpick*, and excited by the signal *axcite*.

Examples

The following example produces a moderately long note with rapidly decaying upper partials. It uses the file *wgpluck.csd* [examples/wgpluck.csd].

Exemple 580. An example of the wgpluck opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o wgpluck.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 10
  ifilt = 1000

  axcite oscil 1, 1, 1
  apluck wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

  out apluck
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

The following example produces a shorter, brighter note. It uses the file *wgpluck_brighter.csd* [examples/wgpluck_brighter.csd].

Exemple 581. An example of the wgpluck opcode with a shorter, brighter note.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgpluck_brighter.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 30
  ifilt = 10

  axcite oscil 1, 1, 1
  apluck wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

  out apluck
```

```
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Michael A. Casey
M.I.T.
Cambridge, Mass.
1997

New in Version 3.47

wgpluck2

wgpluck2 — Physical model of the plucked string.

Description

wgpluck2 is an implementation of the physical model of the plucked string, with control over the pluck point, the pickup point and the filter. Based on the Karplus-Strong algorithm.

Syntax

```
ares wgpluck2 iplk, kamp, icps, kpick, krefl
```

Initialization

iplk -- The point of pluck is *iplk*, which is a fraction of the way up the string (0 to 1). A pluck point of zero means no initial pluck.

icps -- The string plays at *icps* pitch.

Performance

kamp -- Amplitude of note.

kpick -- Proportion of the way along the string to sample the output.

krefl -- the coefficient of reflection, indicating the lossiness and the rate of decay. It must be strictly between 0 and 1 (it will complain about both 0 and 1).

Examples

Here is an example of the wgpluck2 opcode. It uses the file *wgpluck2.csd* [examples/wgpluck2.csd].

Exemple 582. Example of the wgpluck2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgpluck2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  iplk = 0.75
  kamp = 30000
  icps = 220
  kpick = 0.75
  krefl = 0.5

  apluck wgpluck2 iplk, kamp, icps, kpick, krefl

  out apluck
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

repluck

Credits

Author: John ffitch (after Perry Cook)
University of Bath, Codemist Ltd.
Bath, UK

New in Csound version 3.47

wguide1

wguide1 — A simple waveguide model consisting of one delay-line and one first-order lowpass filter.

Description

A simple waveguide model consisting of one delay-line and one first-order lowpass filter.

Syntax

```
ares wguide1 asig, xfreq, kcutoff, kfeedback
```

Performance

asig -- the input of excitation noise.

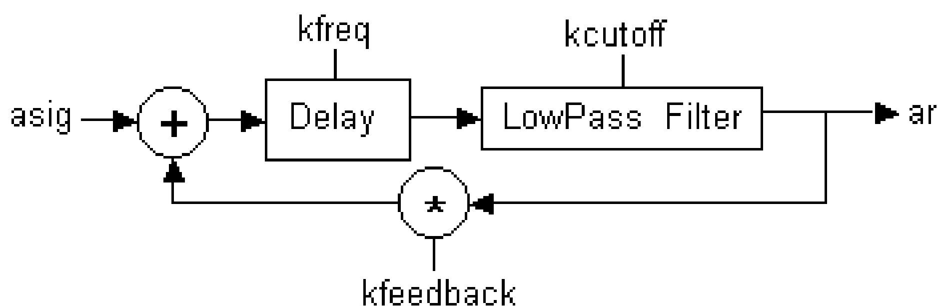
xfreq -- the frequency (i.e. the inverse of delay time) Changed to x-rate in Csound version 3.59.

kcutoff -- the filter cutoff frequency in Hz.

kfeedback -- the feedback factor.

wguide1 is the most elemental waveguide model, consisting of one delay-line and one first-order low-pass filter.

Implementing waveguide algorithms as opcodes, instead of orc instruments, allows the user to set *kr* different than *sr*, allowing better performance particulary when using real-time.



wguide1.

Examples

Here is an example of the wguide1 opcode. It uses the file *wguide1.csd* [examples/wguide1.csd].

Exemple 583. Example of the wguide1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wguide1.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple noise waveform.
instr 1
  ; Generate some noise.
  asig noise 20000, 0.5

  out asig
endin

; Instrument #2 - a waveguide example.
instr 2
  ; Generate some noise.
  asig noise 20000, 0.5

  ; Run it through a wave-guide model.
  kfreq init 200
  kcutoff init 3000
  kfeedback init 0.8
  awg1 wguide1 asig, kfreq, kcutoff, kfeedback

  out awg1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e


</CsScore>
</CsoundSynthesizer>
```

See Also

wguide2

Credits

Author: Gabriel Maldonado
Italy
October 1998

Example written by Kevin Conder.

New in Csound version 3.49

wguide2

wguide2 — A model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters.

Description

A model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters.

Syntax

```
ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \
      kfeedback1, kfeedback2
```

Performance

asig -- the input of excitation noise

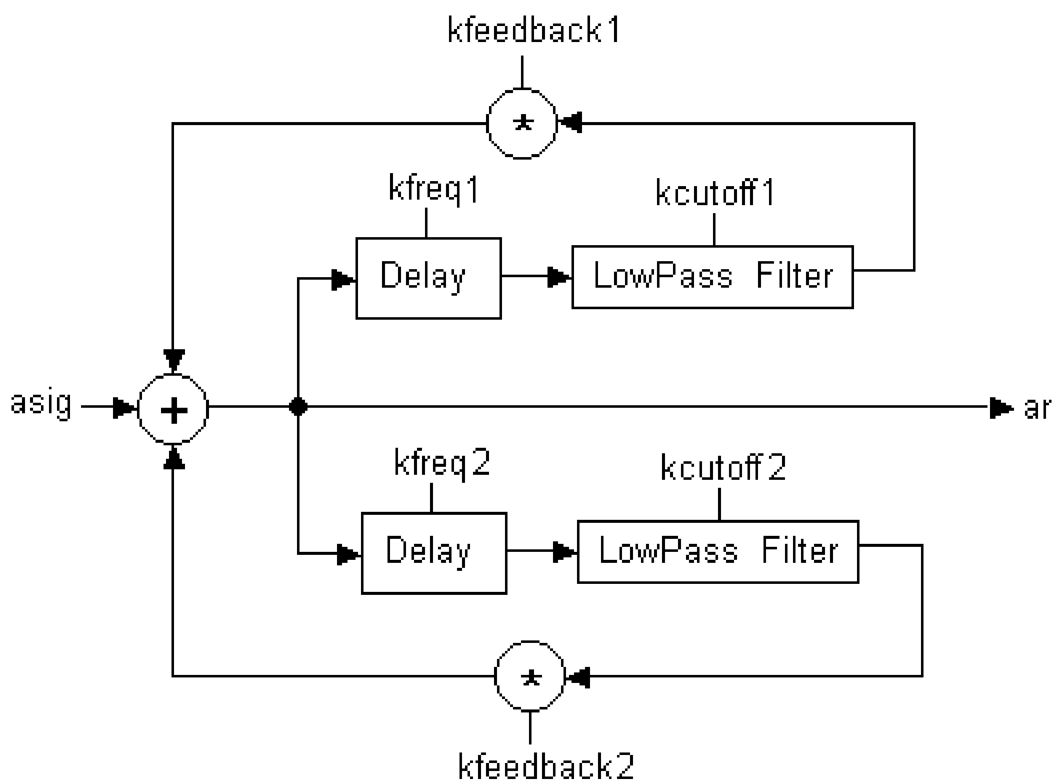
xfreq1, xfreq2 -- the frequency (i.e. the inverse of delay time) Changed to x-rate in Csound version 3.59.

kcutoff1, kcutoff2 -- the filter cutoff frequency in Hz.

kfeedback1, kfeedback2 -- the feedback factor

wguide2 is a model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters. The two feedback lines are mixed and sent to the delay again each cycle.

Implementing waveguide algorithms as opcodes, instead of orc instruments, allows the user to set *kr* different than *sr*, allowing better performance particulary when using real-time.



wguide2.

Note

As a rule of thumb, to avoid making *wguide2* unstable, the sum of the two feedback values should be below 0.5.

Examples

Here is an example of the wguide2 opcode. It uses the file *wguide2.csd* [examples/wguide2.csd].

Exemple 584. Example of the wguide1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wguide1.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
nchnls = 2
instr 1
```

```
  afrq  line    50, 10, 100
  asig  oscil   3000, afrq, 1
  aenv  expon   1,10,0.000001
  aexc  =       aenv*asig
  ares  wguide2 aexc, 500, 1200, 777, 1500, 0.2, 0.25
        out ares,asig
endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

See Also

wguide1

Credits

Author: Gabriel Maldonado
Italy
October 1998

New in Csound version 3.49

Example written by John ffitch

wrap

wrap — Wraps-around the signal that exceeds the low and high thresholds.

Description

Wraps-around the signal that exceeds the low and high thresholds.

Syntax

```
ares wrap asig, klow, khigh
```

```
ires wrap isig, ilow, ihigh
```

```
kres wrap ksig, klow, khigh
```

Initialization

isig -- input signal

ilow -- low threshold

ihigh -- high threshold

Performance

xsig -- input signal

klow -- low threshold

khigh -- high threshold

wrap wraps-around the signal that exceeds the low and high thresholds.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals. *wrap* is also useful for wrap-around of table data when the maximum index is not a power of two (see *table* and *tablei*). Another use of *wrap* is in cyclical event repeating, with arbitrary cycle length.

See Also

limit, *mirror*

Credits

Author: Gabriel Maldonado
Italy

New in Csound version 3.49

wterrain

wterrain — A simple wave-terrain synthesis opcode.

Description

A simple wave-terrain synthesis opcode.

Syntax

```
aout wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \
      itabx, itaby
```

Initialization

itabx, itaby -- The two tables that define the terrain.

Performance

The output is the result of drawing an ellipse with axes *k_xradius* and *k_yradius* centered at (*k_xcenter*, *k_ycenter*), and traversing it at frequency *kpch*.

Examples

Here is an example of the wterrain opcode. It uses the file *wterrain.csd* [examples/wterrain.csd].

Exemple 585. Example of the wterrain opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wterrain.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
kdclk   linseg  0, 0.01, 1, p3-0.02, 1, 0.01, 0
kcx     line    0.1, p3, 1.9
krx     linseg  0.1, p3/2, 0.5, p3/2, 0.1
kpch    line    cpspch(p4), p3, p5 * cpspch(p4)
a1      wterrain    10000, kpch, kcx, kcx, -krx, krx, p6, p7
a1      dcblock a1
        out     a1*kdclk
endin
```

```
</CsInstruments>
<CsScore>

f1      0       8192    10      1 0 0.33 0 0.2 0 0.14 0 0.11
f2      0       4096    10      1

i1      0       4       7.00 1 1 1
i1      4       4       6.07 1 1 2
i1      8       8       6.00 1 2 2
e


</CsScore>
</CsoundSynthesizer>
```

Credits

Author: Matthew Gillard
New in version 4.19

xadsr

xadsr — Calcule l'enveloppe ADSR classique.

Description

Calcule l'enveloppe ADSR classique.

Syntaxe

```
ares xadsr iatt, idec, islev, irel [, idel]
```

```
kres xadsr iatt, idec, islev, irel [, idel]
```

Initialisation

iatt -- durée de l'attaque (attack)

idec -- durée de la première chute (decay)

islev -- niveau d'entretien (sustain)

irel -- durée de la chute (release)

idel -- délai de niveau zéro avant le démarrage de l'enveloppe

Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

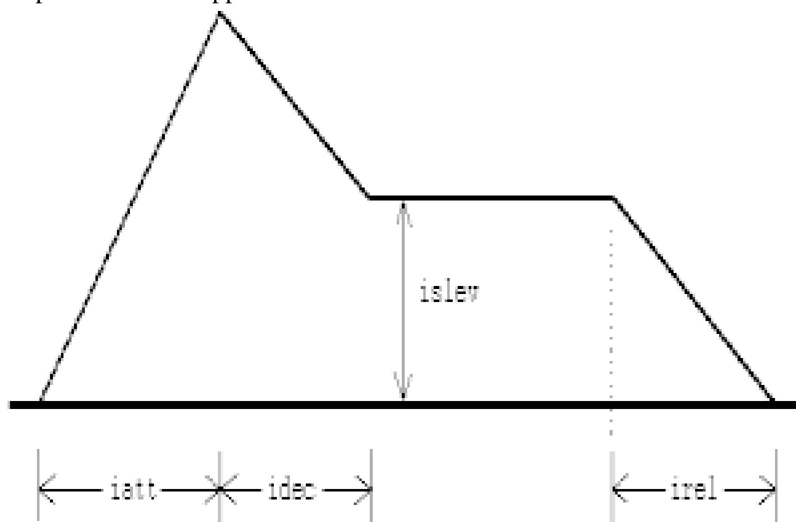


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *xadsr* n'est pas adapté au traitement des évènements MIDI, pour lesquels il faut plutôt utiliser *mxadsr*.

L'opcode *xadsr* est identique à *adsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

xadsr est nouveau dans la version 3.51 de Csound.

Voir Aussi

adsr, *madsr*, *mxadsr*

Crédits

Auteur : John ffitch

xin

xin — Passes variables from a user-defined opcode block,

Description

The *xin* and *xout* opcodes copy variables to and from the opcode definition, allowing communication with the calling instrument.

The types of input and output variables are defined by the parameters *intypes* and *outtypes*.

Notes

- *xin* and *xout* should be called only once, and *xin* should precede *xout*, otherwise an init error and deactivation of the current instrument may occur.
- These opcodes actually run only at i-time. Performance time copying is done by the user opcode call. This means that skipping *xin* or *xout* with *kgoto* has no effect, while skipping with *igoto* affects both init and performance time operation.

Syntax

```
xinarg1 [, xinarg2] ... [xinargN] xin
```

Performance

xinarg1, *xinarg2*, ... - input arguments. The number and type of variables must agree with the user-defined opcode's *intypes* declaration. However, *xin* does not check for incorrect use of init-time and control-rate variables.

The syntax of a user-defined opcode block is as follows:

```
opcode  name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
[setksmps  iksmps]
... the rest of the instrument's code.
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

The new opcode can then be used with the usual syntax:

```
[xinarg1] [, xinarg2] ... [xinargN]  name  [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

Examples

See the example for the *opcode* opcode.

See Also

endop, *opcode*, *setksmps*, *xout*

Credits

Author: Istvan Varga, 2002; based on code by Matt J. Ingalls

New in version 4.22

xout

xout — Retrieves variables from a user-defined opcode block,

Description

The *xin* and *xout* opcodes copy variables to and from the opcode definition, allowing communication with the calling instrument.

The types of input and output variables are defined by the parameters *intypes* and *outtypes*.

Notes

- *xin* and *xout* should be called only once, and *xin* should precede *xout*, otherwise an init error and deactivation of the current instrument may occur.
- These opcodes actually run only at i-time. Performance time copying is done by the user opcode call. This means that skipping *xin* or *xout* with *kgoto* has no effect, while skipping with *igoto* affects both init and performance time operation.

Syntax

```
xout xoutarg1 [, xoutarg2] ... [, xoutargN]
```

Performance

xoutarg1, *xoutarg2*, ... - output arguments. The number and type of variables must agree with the user-defined opcode's *outtypes* declaration. However, *xout* does not check for incorrect use of init-time and control-rate variables.

The syntax of a user-defined opcode block is as follows:

```
opcode  name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
[setksmps  iksmps]
... the rest of the instrument's code.
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

The new opcode can then be used with the usual syntax:

```
[xinarg1] [, xinarg2] ... [xinargN]  name  [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

Examples

See the example for the *opcode* opcode.

See Also

endop, *opcode*, *setksmps*, *xin*

Credits

Author: Istvan Varga, 2002; based on code by Matt J. Ingalls

New in version 4.22

xscanmap

xscanmap — Allows the position and velocity of a node in a scanned process to be read.

Description

Allows the position and velocity of a node in a scanned process to be read.

Syntax

```
kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]
```

Initialization

iscan -- which scan process to read

iwhich (optional) -- which node to sense. The default is 0.

Performance

kamp -- amount to amplify the *kpos* value.

kvamp -- amount to amplify the *kvel* value.

The internal state of a node is read. This includes its position and velocity. They are amplified by the *kamp* and *kvamp* values.

Credits

Author: John ffitch

New in version 4.20

xscansmap

xscansmap — Allows the position and velocity of a node in a scanned process to be read.

Description

Allows the position and velocity of a node in a scanned process to be read.

Syntax

```
xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]
```

Initialization

iscan -- which scan process to read

iwhich (optional) -- which node to sense. The default is 0.

Performance

kpos -- the node's postion.

kvel -- the node's velocity.

kamp -- amount to amplify the *kpos* value.

kvamp -- amount to amplify the *kvel* value.

The internal state of a node is read. This includes its position and velocity. They are amplified by the *kamp* and *kvamp* values.

Credits

New in version 4.21

November 2002. Thanks to Rasmus Ekman for pointing this opcode out.

xscans

xscans — Fast scanned synthesis waveform and the wavetable generator.

Description

Experimental version of *scans*. Allows much larger matrices and is faster and smaller but removes some (unused?) flexibility. If liked, it will replace the older opcode as it is syntax compatible but extended.

Syntax

```
ares xscans kamp, kfreq, ifntraj, id [, iorder]
```

Initialization

ifntraj -- table containing the scanning trajectory. This is a series of numbers that contains addresses of masses. The order of these addresses is used as the scan path. It should not contain values greater than the number of masses, or negative numbers. See the *introduction to the scanned synthesis section*.

id -- If positive, the ID of the opcode. This will be used to point the scanning opcode to the proper waveform maker. If this value is negative, the absolute of this value is the wavetable on which to write the waveshape. That wavetable can be used later from an other opcode to generate sound. The initial contents of this table will be destroyed.

iorder (optional, default=0) -- order of interpolation used internally. It can take any value in the range 1 to 4, and defaults to 4, which is quartic interpolation. The setting of 2 is quadratic and 1 is linear. The higher numbers are slower, but not necessarily better.

Performance

kamp -- output amplitude. Note that the resulting amplitude is also dependent on instantaneous value in the wavetable. This number is effectively the scaling factor of the wavetable.

kfreq -- frequency of the scan rate

Matrix Format

The new matrix format is a list of connections, one per line linking point x to point y. There is no weight given to the link; it is assumed to be unity. The list is proceeded by the line <MATRIX> and ends with a </MATRIX> line

For example, a circular string of 8 would be coded as

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
3 2
3 4
4 3
4 5
5 4
```

```
5 6
6 5
6 7
7 6
0 7
</MATRIX>
```

Credits

Written by John ffitch.

New in version 4.20

Examples

For an example, see the documentation on *scans*.

See Also

scans, *xscanu*

xscanu

xscanu — Compute the waveform and the wavetable for use in scanned synthesis.

Description

Experimental version of *scanu*. Allows much larger matrices and is faster and smaller but removes some (unused?) flexibility. If liked, it will replace the older opcode as it is syntax compatible but extended.

Syntax

```
xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, iright, kpos, kstrngth, ain, idisp, id
```

Initialization

init -- the initial position of the masses. If this is a negative number, then the absolute of init signifies the table to use as a hammer shape. If init > 0, the length of it should be the same as the intended mass number, otherwise it can be anything.

irate -- update rate.

ifnvel -- the ftable that contains the initial velocity for each mass. It should have the same size as the intended mass number.

ifnmass -- ftable that contains the mass of each mass. It should have the same size as the intended mass number.

ifnstif --

- *either* an ftable that contains the spring stiffness of each connection. It should have the same size as the square of the intended mass number. The data ordering is a row after row dump of the connection matrix of the system.
- *or* a string giving the name of a file in the MATRIX format

ifncentr -- ftable that contains the centering force of each mass. It should have the same size as the intended mass number.

ifndamp -- the ftable that contains the damping factor of each mass. It should have the same size as the intended mass number.

ileft -- If init < 0, the position of the left hammer (ileft = 0 is hit at leftmost, ileft = 1 is hit at rightmost).

iright -- If init < 0, the position of the right hammer (iright = 0 is hit at leftmost, iright = 1 is hit at rightmost).

idisp -- If 0, no display of the masses is provided.

id -- If positive, the ID of the opcode. This will be used to point the scanning opcode to the proper waveform maker. If this value is negative, the absolute of this value is the wavetable on which to write the waveshape. That wavetable can be used later from an other opcode to generate sound. The initial

contents of this table will be destroyed.

Performance

kmass -- scales the masses

kstif -- scales the spring stiffness

kcentr -- scales the centering force

kdamp -- scales the damping

kpos -- position of an active hammer along the string (kpos = 0 is leftmost, kpos = 1 is rightmost). The shape of the hammer is determined by init and the power it pushes with is kstrngth.

kstrngth -- power that the active hammer uses

ain -- audio input that adds to the velocity of the masses. Amplitude should not be too great.

Matrix Format

The new matrix format is a list of connections, one per line linking point x to point y. There is no weight given to the link; it is assumed to be unity. The list is proceeded by the line <MATRIX> and ends with a </MATRIX> line

For example, a circular string of 8 would be coded as

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
3 2
3 4
4 3
4 5
5 4
5 6
6 5
6 7
7 6
0 7
</MATRIX>
```

Credits

Written by John ffitch.

New in version 4.20

Examples

For an example, see the documentation on *scans*.

See Also

scanu, *xscans*

xtratim

xtratim — Extend the duration of real-time generated events.

Description

Extend the duration of real-time generated events and handle their extra life (Usually for usage along with *release* instead of *linenr*, *linsegr*, etc).

Syntax

```
xtratim iextradur
```

Initialization

iextradur -- additional duration of current instrument instance

Performance

xtratim extends current MIDI-activated note duration by *iextradur* seconds after the corresponding noteoff message has deactivated the current note itself. It is usually used in conjunction with *release*. This opcode has no output arguments.

This opcode is useful for implementing complex release-oriented envelopes, whose duration is not known when the envelope starts (e.g. for real-time MIDI generated events).

Examples

Here is a simple example of the xtratim opcode. It uses the file *xtratim.csd* [examples/xtratim.csd].

Exemple 586. Example of the xtratim opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example shows how to generate a release segment for an ADSR envelope after a MIDI noteoff is received, extending the duration with *xtratim* and using *release* to check whether the note is on the release phase.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent  MIDI in
-odac           -iadc     -d       -M0  ;;;realtime I/O
</CsOptions>
<CsInstruments>
;Simple usage of the xtratim opcode

instr 1

  inum notnum
  icps cpsmidi
  iamp ampmidi 4000
 ;
```

```
 ;------- complex envelope block ------
  xtratim 1 ;extra-time, i.e. release dur
  krel init 0
  krel release ;outputs release-stage flag (0 or 1 values)
  if (krel == 1) kgoto rel ;if in release-stage goto release section
 ;
 ;************ attack and sustain section ***********
  kmp1 linseg 0, .03, 1, .05, 1, .07, 0, .08, .5, 4, 1, 50, 1
  kmp = kmp1*iamp
   kgoto done
 ;
 ;--------- release section --------
   rel:
  kmp2 linseg 1, .3, .2, .7, 0
  kmp = kmp1*kmp2*iamp
  done:
 ;------
  a1 oscili kmp, icps, 1
  out a1
 endin

</CsInstruments>
<CsScore>
f 0 3600 ;dummy table to wait for realtime MIDI events
e
</CsScore>
</CsoundSynthesizer>
```

Here is a more elaborate example of the xtratim opcode. It uses the file *xtratim-2.csd* [examples/xtratim-2.csd].

Exemple 587. More complex example of the xtratim opcode.

This example shows how to generate a release segment for an ADSR envelope after a MIDI noteoff is received, extending the duration with *xtratim* and using *release* to check whether the note is on the release phase. Two envelopes are generated simultaneously for the left and right channels.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    Silent  MIDI in
-odac           -idac     -d       -M0  ;;;realtime I/O
</CsOptions>
<CsInstruments>
;xtratim example by Jonathan Murphy Dec. 2006

  sr     =  44100
  ksmps     =  32
  nchnls    =  2

            ; sine wave for oscillators
gisin     ftgen     1, 0, 4096, 10, 1
            ; set volume initially to midpoint
            ctrlinit  1, 7,64

;;; simple two oscil, two envelope synth
    instr 1

            ; frequency
  kcps     cpsmidib
            ; initial velocity (noteon)
  ivel     veloc

            ; master volume
  kamp     ctrl7     1, 7, 0, 127
  kamp     =  kamp * ivel

            ; parameters for aenv1
  iatt1     =  0.03
  idec1     =  1
  isus1     =  0.25
  irel1     =  1
```

```
              ; parameters for aenv2
  iatt2       =  0.06
  idec2       =  2
  isus2       =  0.5
  irel2       =  2

              ; extra (release) time allocated
              xtratim   (irel1>irel2 ? irel1 : irel2)
              ; krel is used to trigger envelope release
  krel       init       0
  krel       release
              ; if noteoff received, krel == 1, otherwise krel == 0
if (krel == 1) kgoto rel

              ; attack, decay, sustain segments
  atmp1       linseg    0, iatt1, 1, idec1, isus1 , 1, isus1
  atmp2       linseg    0, iatt2, 1, idec2, isus2 , 1, isus2
  aenv1       =  atmp1
  aenv2       =  atmp2
              kgoto     done

              ; release segment
rel:
  atmp3       linseg    1, irel1, 0, 1, 0
  atmp4       linseg    1, irel2, 0, 1, 0
  aenv1       =  atmp1 * atmp3  ;to go from the current value (in case
  aenv2       =  atmp2 * atmp4  ;the attack hasn't finished) to the release.

              ; control oscillator amplitude using envelopes
done:
  aosc1       oscil     aenv1, kcps, 1
  aosc2       oscil     aenv2, kcps * 1.5, 1
  aosc1       =  aosc1 * kamp
  aosc2       =  aosc2 * kamp

              ; send aosc1 to left channel, aosc2 to right,
              ; release times are noticably different

              outs      aosc1, aosc2

    endin

</CsInstruments>
<CsScore>

f 0 3600 ;dummy table to wait for realtime MIDI events

</CsScore>
</CsoundSynthesizer>
```

See Also

linenr, *release*

Credits

Author: Gabriel Maldonado

Italy

Examples by Gabriel Maldonado and Jonathan Murphy

New in Csound version 3.47

xyin

xyin — Sense the cursor position in an output window

Description

Sense the cursor position in an output window. When *xyin* is called the position of the mouse within the output window is used to reply to the request. This simple mechanism does mean that only one *xyin* can be used accurately at once. The position of the mouse is reported in the output window.

Syntax

```
kx, ky xyin iprd, ixmin, ixmax, iymin, iymax [, ixinit] [, iyinit]
```

Initialization

iprd -- period of cursor sensing (in seconds). Typically .1 seconds.

xmin, xmax, ymin, ymax -- edge values for the x-y coordinates of a cursor in the input window.

ixinit, iyinit (optional) -- initial x-y coordinates reported; the default values are 0,0. If these values are not within the given min-max range, they will be coerced into that range.

Performance

xyin samples the cursor x-y position in an input window every *iprd* seconds. Output values are repeated (not interpolated) at the k-rate, and remain fixed until a new change is registered in the window. There may be any number of input windows. This unit is useful for real-time control, but continuous motion should be avoided if *iprd* is unusually small.

Note

Depending on your platform and distribution, you might need to enable displays using the --*displays* command line flag.

Examples

Here is an example of the xyin opcode. It uses the file *xyin.csd* [examples/xyin.csd].

Exemple 588. Example of the xyin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     --displays ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o xyin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Print and capture values every 0.1 seconds.
  iprd = 0.1
  ; The x values are from 1 to 30.
  ixmin = 1
  ixmax = 30
  ; The y values are from 1 to 30.
  iymin = 1
  iymax = 30
  ; The initial values for X and Y are both 15.
  ixinit = 15
  iyinit = 15

  ; Get the values kx and ky using the xyin opcode.
  kx, ky xyin iprd, ixmin, ixmax, iymin, iymax, ixinit, iyinit

  ; Print out the values of kx and ky.
  printks "kx=%f, ky=%f\\n", iprd, kx, ky

  ; Play an oscillator, use the x values for amplitude and
  ; the y values for frequency.
  kamp = kx * 1000
  kcps = ky * 220
  a1 oscil kamp, kcps, 1

  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 30 seconds.
i 1 0 30
e


</CsScore>
</CsoundSynthesizer>
```

As the values of kx and ky change, they will be printed out like this:

```
kx=8.612036, ky=22.677933
kx=10.765685, ky=15.644135
```

Credits

Example written by Kevin Conder.

zacl

zacl — Efface une ou plusieurs variables dans l'espace za.

Description

Efface une ou plusieurs variables dans l'espace za.

Syntaxe

```
zacl kfirst, klast
```

Exécution

kfirst -- Première position zk ou za de l'intervalle à effacer.

klast -- Dernière position zk ou za de l'intervalle à effacer.

zacl efface une ou plusieurs variables dans l'espace za. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-a à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

Exemples

Voici un exemple de l'opcode zacl. Il utilise le fichier *zacl.csd* [examples/zacl.csd].

Exemple 589. Exemple de l'opcode zacl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zacl.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Generate a simple sine waveform.
  asin oscil 20000, 440, 1

  ; Send the sine waveform to za variable #1.
  zaw asin, 1
```

```
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Read za variable #1.
  a1 zar 1

  ; Generate the audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zacl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zamod, *zar*, *zaw*, *zawm*, *ziw*, *ziwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zakinit

zakinit — Etablit l'espace zak.

Description

Etablit l'espace zak. Ne doit être appelé qu'une seule fois.

Syntaxe

```
zakinit isizea, isizek
```

Initialisation

isizea -- le nombre de positions de taux audio pour les patch de taux-a. Chaque position est un tableau de longueur ksmps.

isizek -- le nombre de positions à réserver pour les nombres en virgule flottante dans l'espace zk. On peut lire et écrire dans celles-ci au taux-i et au taux-k.

Exécution

Il y a au moins une position d'allouée pour chaque espace za et zk. Il peut y avoir des milliers ou des dizaines de milliers de positions za et zk, mais la plupart des pièces n'en nécessitent probablement que quelques douzaines pour patcher les signaux. Ces positions de patch sont référencées par un numéro dans les autres opcodes zak.

Pour n'exécuter *zakinit* qu'une seule fois, on le place en dehors de toute définition d'instrument, dans l'en-tête de l'orchestre, après *sr*, *kr*, *ksmps*, et *nchnls*.

Note

Les canaux zak se comptent à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

Exemples

Voici un exemple de l'opcode zakinit. Il utilise le fichier *zakinit.csd* [examples/zakinit.csd].

Exemple 590. Exemple de l'opcode zakinit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o zakinit.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 4410
nchnls = 1

; Initialize the ZAK space.
; Create 3 a-rate variables and 5 k-rate variables.
zakinit 2, 3

instr 1 ;a simple waveform.
        ; Generate a simple sine waveform.
        asin oscil 20000, 440, 1

        ; Send the sine waveform to za variable #1.
        zaw asin, 1
endin

instr 2  ;generates audio output.
        ; Read za variable #1.
        a1 zar 1

        ; Generate audio output.
        out a1

        ; Clear the za variables, get them ready for
        ; another pass.
        zacl 0, 3
endin

instr 3  ;increments k-type channels
         k0 zkr 0
         k1 zkr 1
         k2 zkr 2

        zkw k0+1, 0
        zkw k1+5, 1
        zkw k2+10, 2
endin

instr 4 ;displays values from k-type channels
        k0 zkr 0
        k1 zkr 1
        k2 zkr 2

; The total count for k0 is 30, since there are 10
; control blocks per second and intruments 3 and 4
; are on for 3 seconds.
        printf "k0 = %i\n",k0, k0
        printf "k1 = %i\n",k1, k1
        printf "k2 = %i\n",k2, k2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 1
i 2 0 1

i 3 0 3
i 4 0 3
e

</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Robin Whittle
Australie

Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zamod

zamod — Module un signal de taux-a par un autre.

Description

Module un signal de taux-a par un autre.

Syntaxe

```
ares zamod asig, kzamod
```

Exécution

asig -- Le signal d'entrée

kzamod -- Contrôle quelle variable za sera utilisée pour la modulation. Une valeur positive indique une modulation additive, une valeur négative indique une modulation multiplicative. Une valeur de 0 ne fait aucun changement à *asig*.

zamod Module un signal de taux-a par un autre, qui provient d'une variable za. La position de la variable modulante est contrôlée par la variable de taux-i ou de taux-k *kzamod*. Ceci est la version de taux-a de *zkmod*.

Exemples

Voici un exemple de l'opcode zamod. Il utilise le fichier *zamod.csd* [examples/zamod.csd].

Exemple 591. Exemple de l'opcode zamod.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zamod.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a simple waveform.
instr 1
  ; Vary an a-rate signal linearly from 20,000 to 0.
  asig line 20000, p3, 0
```

```
  ; Send the signal to za variable #1.
  zaw asig, 1
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Generate a simple sine wave.
  asin oscil 1, 440, 1

  ; Modify the sine wave, multiply its amplitude by
  ; za variable #1.
  a1 zamod asin, -1

  ; Generate the audio output.
  out a1

  ; Clear the za variables, prepare them for
  ; another pass.
  zacl 0, 2
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zacl, *ziw*, *ziwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zar

zir — Lecture à partir d'une position dans l'espace za au taux-a.

Description

Lecture à partir d'une position dans l'espace za au taux-a.

Syntaxe

```
ares zar kndx
```

Exécution

kndx -- pointe sur la position za à lire.

zar lit la suite de nombres décimaux à *kndx* dans l'espace za, qui sont les ksmps nombres décimaux de taux-a à traiter dans un cycle-k.

Exemples

Voici un exemple du opcode zar. Il utilise le *zar.csd* [examples/zar.csd].

Exemple 592. Exemple de l'opcode zar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zar.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Generate a simple sine waveform.
  asin oscil 20000, 440, 1

  ; Send the sine waveform to za variable #1.
  zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
```

```
  ; Read za variable #1.
  a1 zar 1

  ; Generate audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zacl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zarg, *zir*, *zkr*

Crédits

Auteur : Robin Whittle
Autralie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zarg

zarg — Lecture à partir d'une position dans l'espace za au taux-a avec application d'un gain.

Description

Lecture à partir d'une position dans l'espace za au taux-a avec application d'un gain.

Syntaxe

```
ares zarg kndx, kgain
```

Initialisation

kndx -- pointe sur la position za à lire.

kgain -- Multiplicateur pour le signal taux-a.

Exécution

zarg lit la suite de nombres décimaux à *kndx* dans l'espace za, qui sont les ksmps nombres décimaux de taux-a à traiter dans un cycle-k. *zarg* multiplie aussi le signal de taux-a par la valeur de taux-k *kgain*.

Exemples

Voici un exemple de l'opcode zarg. Il utilise le fichier *zarg.csd* [examples/zarg.csd].

Exemple 593. Exemple de l'opcode zarg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zarg.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Generate a simple sine waveform, with an amplitude
  ; between 0 and 1.
  asin oscil 1, 440, 1
```

```
  ; Send the sine waveform to za variable #1.
  zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Read za variable #1, multiply its amplitude by 20,000.
  a1 zarg 1, 20000

  ; Generate audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zacl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zar, *zir*, *zkr*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zaw

zaw — Ecrit dans une variable za au taux-a sans mixage.

Description

Ecrit dans une variable za au taux-a sans mixage.

Syntaxe

```
zaw asig, kndx
```

Exécution

asig -- Valeur à écrire dans la position za.

kndx -- Pointe sur la position zk ou za vers laquelle écrire.

zaw écrit *asig* dans la variable za specifiée par *kndx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

Exemples

Voici un exemple de l'opcode zaw. Il utilise le fichier *zaw.csd* [examples/zaw.csd].

Exemple 594. Exemple de l'opcode zaw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zaw.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Generate a simple sine waveform.
  asin oscil 20000, 440, 1

  ; Send the sine waveform to za variable #1.
```

```
  zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Read za variable #1.
  a1 zar 1

  ; Generate the audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zacl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zawm, *ziw*, *ziwm*, *zkw*, *zkwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zawm

zawm — Ecrit dans une variable za au taux-a avec mixage.

Description

Ecrit dans une variable za au taux-a avec mixage.

Syntaxe

```
zawm asig, kndx [, imix]
```

Initialisation

imix (facultatif, par defaut=1) -- indique si le mixage sera fait.

Exécution

asig -- valeur à écrire dans l'espace za.

kndx -- Pointe sur la position zk ou za vers laquelle écrire.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée et il n'y a aucune écriture.

zawm est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est specifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toute autre valeur entraînera un mixage.

Avertissement : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zacl* pour effacer ces variables.

Exemples

Voici un exemple de l'opcode zawm. Il utilise le fichier *zawm.csd* [examples/zawm.csd].

Exemple 595. Exemple de l'opcode zawm.

Voir les sections *Audio en Temps R#el* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zawm.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
  ; Generate a simple sine waveform.
  asin oscil 15000, 440, 1

  ; Mix the sine waveform with za variable #1.
  zawm asin, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
  ; Generate another waveform with a different frequency.
  asin oscil 15000, 880, 1

  ; Mix this sine waveform with za variable #1.
  zawm asin, 1
endin

; Instrument #3 -- generates audio output.
instr 3
  ; Read za variable #1, containing both waveforms.
  a1 zar 1

  ; Generate the audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zacl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zaw, *ziw*, *ziwm*, *zkw*, *zkwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zfilter2

zfilter2 — Performs filtering using a transposed form-II digital filter lattice with radial pole-shearing and angular pole-warping.

Description

General purpose custom filter with time-varying pole control. The filter coefficients implement the following difference equation:

(1)*y(n) = b0*x[n] + b1*x[n-1] +...+ bM*x[n-M] - a1*y[n-1] -...- aN*y[n-N]

the system function for which is represented by:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + ... + bM*Z^{-M}}{1 + a1*Z^{-1} + ... + aN*Z^{-N}}$$

Syntax

```
ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \
      ia1,ia2, ..., iaN
```

Initialization

At initialization the number of zeros and poles of the filter are specified along with the corresponding zero and pole coefficients. The coefficients must be obtained by an external filter-design application such as Matlab and specified directly or loaded into a table via *GEN01*. With *zfilter2*, the roots of the characteristic polynomials are solved at initialization so that the pole-control operations can be implemented efficiently.

Performance

The *filter2* opcodes perform filtering using a transposed form-II digital filter lattice with no time-varying control. *zfilter2* uses the additional operations of radial pole-shearing and angular pole-warping in the Z plane.

Pole shearing increases the magnitude of poles along radial lines in the Z-plane. This has the affect of altering filter ring times. The k-rate variable *kdamp* is the damping parameter. Positive values (0.01 to 0.99) increase the ring-time of the filter (hi-Q), negative values (-0.01 to -0.99) decrease the ring-time of the filter, (lo-Q).

Pole warping changes the frequency of poles by moving them along angular paths in the Z plane. This operation leaves the shape of the magnitude response unchanged but alters the frequencies by a constant factor (preserving 0 and p). The k-rate variable *kfreq* determines the frequency warp factor. Positive values (0.01 to 0.99) increase frequencies toward p and negative values (-0.01 to -0.99) decrease frequencies toward 0.

Since *filter2* implements generalized recursive filters, it can be used to specify a large range of general DSP algorithms. For example, a digital waveguide can be implemented for musical instrument modeling using a pair of *delayr* and *delayw* opcodes in conjunction with the *filter2* opcode.

Examples

A controllable second-order IIR filter operating on an a-rate signal:

```
a1 zfilter2 asig, kdamp, kfreq, 1, 2, 1, ia1, ia2 ; controllable a-rate   ; IIR filter
```

See Also

filter2

Credits

Author: Michael A. Casey
M.I.T.
Cambridge, Mass.
1997

New in Version 3.47

zir

zir — Lecture à partir d'une position dans un espace zk au taux-i.

Description

Lecture à partir d'une position dans un espace zk au taux-i.

Syntaxe

```
ir zir indx
```

Initialisation

indx -- pointe vers la position zk à lire.

Exécution

zir lit le signal à la position *indx* dans l'espace zk.

Exemples

Voici un exemple de l'opcode zir. Il utilise le fichier *zir.csd* [examples/zir.csd].

Exemple 596. Exemple de l'opcode zir.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zir.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
  ; Set the zk variable #1 to 32.594.
  ziw 32.594, 1
endin

; Instrument #2 -- prints out zk variable #1.
instr 2
  ; Read the zk variable #1 at i-rate.
```

```
  i1 zir 1

  ; Print out the value of zk variable #1.
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zar, *zarg*, *zkr*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

ziw

ziw — Ecrit dans une variable zk au taux-i sans mixage.

Description

Ecrit dans une variable zk au taux-i sans mixage.

Syntaxe

```
ziw isig, indx
```

Initialisation

isig -- Initialise la valeur de la position zk.

indx -- Pointe sur la position zk ou za vers laquelle écrire.

Exécution

ziw écrit *isig* dans la variable zk spécifié par *indx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

Exemples

Voici un exemple de l'opcode ziw. Il utilise le fichier *ziw.csd* [examples/ziw.csd].

Exemple 597. Exemple de l'opcode ziw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ziw.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
```

```
  ; Set zk variable #1 to 64.182.
  ziw 64.182, 1
endin

; Instrument #2 -- prints out zk variable #1.
instr 2
  ; Read zk variable #1 at i-rate.
  i1 zir 1

  ; Print out the value of zk variable #1.
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zaw, *zawm*, *ziwm*, *zkw*, *zkwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

ziwm

ziwm — Ecrit dans une variable zk au taux-i avec mixage.

Description

Ecrit dans une variable zk au taux-i avec mixage.

Syntaxe

```
ziwm isig, indx [, imix]
```

Initialisation

isig -- initialise la valeur à la position zk.

indx -- pointe sur la position zk vers laquelle écrire.

imix (facultatif, par défault=1) -- indique si le mixage doit avoir lieu.

Exécution

ziwm est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw* et *zaw*. Toute autre valeur entraînera un mixage.

Attention : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm* et *zawm*, il faut faire attention à ce que les variables qui recoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche serait d'établir certains intervalles de variables zk et za à utiliser pour le mixage, puis d'utiliser *zkcl* ou *zacl* pour effacer ces intervalles.

Exemples

Voici un exemple de l'opcode ziwm. Il utilise le fichier *ziwm.csd* [examples/ziwm.csd].

Exemple 598. Exemple de l'opcode ziwm.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ziwm.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
  ; Add 20.5 to zk variable #1.
  ziwm 20.5, 1
endin

; Instrument #2 -- another simple instrument.
instr 2
  ; Add 15.25 to zk variable #1.
  ziwm 15.25, 1
endin

; Instrument #3 -- prints out zk variable #1.
instr 3
  ; Read zk variable #1 at i-rate.
  i1 zir 1

  ; Print out the value of zk variable #1.
  ; It should be 35.75 (20.5 + 15.25)
  print i1
endin


</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

zaw, *zawm*, *ziw*, *zkw*, *zkwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zkcl

zkcl — Efface une ou plusieurs variable dans l'espace zk.

Description

Efface une ou plusieurs variable dans l'espace zk.

Syntaxe

```
zkcl kfirst, klast
```

Exécution

ksig -- Le signal d'entrée

kfirst -- Première position zk ou za de l'intervalle à effacer.

klast -- Dernière position zk ou za de l'intervalle à effacer.

zkcl efface une ou plusieurs variables dans l'espace zk. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-k à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

Exemples

Voici un exemple de l'opcode zkcl. Il utilise le fichier *zkcl.csd* [examples/zkcl.csd].

Exemple 599. Exemple de l'opcode zkcl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkcl.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Linearly vary a k-rate signal from 220 to 1760.
  kline line 220, p3, 1760
```

```
  ; Add the linear signal to zk variable #1.
  zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Read zk variable #1.
  kfreq zkr 1

  ; Use the value of zk variable #1 to vary
  ; the frequency of a sine waveform.
  a1 oscil 20000, kfreq, 1

  ; Generate the audio output.
  out a1

  ; Clear the zk variables, get them ready for
  ; another pass.
  zkcl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
; Play Instrument #2 for three seconds.
i 2 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zacl, *zkwm*, *zkw*, *zkmod*, *zkr*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zkmod

zkmod — Facilite la modulation d'un signal par un autre.

Description

Facilite la modulation d'un signal par un autre.

Syntaxe

```
kres zkmod ksig, kzkmod
```

Exécution

ksig -- Le signal d'entrée

kzkmod -- contrôle quelle variable zk est utilisée pour la modulation. Une valeur positive signifie une modulation additive, une valeur négative une modulation multiplicative. La valeur 0 ne fait aucun changement à *ksig*. *kzkmod* peut être de taux-i ou de taux-k.

zkmod Facilite la modulation d'un signal par un autre, le signal de modulation provenant d'une variable zk. La modulation spécifiée peut être additive ou multiplicative.

Exemples

Voici un exemple de l'opcode zkmod. Il utlise le fichier *zkmod.csd* [examples/zkmod.csd].

Exemple 600. Exemple de l'opcode zkmod.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkmod.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a signal with jitter.
instr 1
  ; Generate a k-rate signal goes from 30 to 2,000.
  kline line 30, p3, 2000

  ; Add the signal into zk variable #1.
```

```
  zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Create a k-rate signal modulated the jitter opcode.
  kamp init 20
  kcpsmin init 40
  kcpsmax init 60
  kjtr jitter kamp, kcpsmin, kcpsmax

  ; Get the frequency values from zk variable #1.
  kfreq zkr 1
  ; Add the the frequency values in zk variable #1 to
  ; the jitter signal.
  kjfreq zkmod kjtr, 1

  ; Use a simple sine waveform for the left speaker.
  aleft oscil 20000, kfreq, 1
  ; Use a sine waveform with jitter for the right speaker.
  aright oscil 20000, kjfreq, 1

  ; Generate the audio output.
  outs aleft, aright

  ; Clear the zk variables, prepare them for
  ; another pass.
  zkcl 0, 2
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zamod, *zkcl*, *zkr*, *zkwm*, *zkw*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zkr

zkr — Lecture à partir d'une position dans l'espace zk au taux-k.

Description

Lecture à partir d'une position dans l'espace zk au taux-k.

Syntaxe

```
kres zkr kndx
```

Initialisation

kndx -- pointe sur la position za à lire.

Exécution

zkr lit la suite de nombres décimaux à *kndx* dans l'espace zk.

Exemples

Voici un exemple de l'opcode zkr. Il utilise le fichier *zkr.csd* [examples/zkr.csd].

Exemple 601. Exemple de l'opcode zkr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkr.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Linearly vary a k-rate signal from 440 to 880.
  kline line 440, p3, 880

  ; Add the linear signal to zk variable #1.
  zkw kline, 1
endin
```

```
; Instrument #2 -- generates audio output.
instr 2
  ; Read zk variable #1.
  kfreq zkr 1

  ; Use the value of zk variable #1 to vary
  ; the frequency of a sine waveform.
  a1 oscil 20000, kfreq, 1

  ; Generate the audio output.
  out a1

  ; Clear the zk variables, get them ready for
  ; another pass.
  zkcl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zar, *zarg*, *zir*, *zkcl*, *zkmod*, *zkwm*, *zkw*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zkw

zkw — Ecrit dans une variable zk au taux-k sans mixage.

Description

Ecrit dans une variable zk au taux-k sans mixage.

Syntaxe

```
zkw ksig, kndx
```

Exécution

ksig -- valeur à écrire dans la position zk.

kndx -- pointe sur la position zk ou za vers laquelle écrire.

zkw écrit *ksig* dans la variable zk spécifiée par *kndx*.

Exemples

Voici un exemple de l'opcode zkw. Il utilise le fichier *zkw.csd* [examples/zkw.csd].

Exemple 602. Exemple de l'opcode zkw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkw.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
  ; Linearly vary a k-rate signal from 100 to 1,000.
  kline line 100, p3, 1000

  ; Add the linear signal to zk variable #1.
  zkw kline, 1
endin

; Instrument #2 -- generates audio output.
```

```
instr 2
  ; Read zk variable #1.
  kfreq zkr 1

  ; Use the value of zk variable #1 to vary
  ; the frequency of a sine waveform.
  a1 oscil 20000, kfreq, 1

  ; Generate the audio output.
  out a1

  ; Clear the zk variables, get them ready for
  ; another pass.
  zkcl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir aussi

zaw, *zawm*, *ziw*, *ziwm*, *zkr*, *zkwm*

Crédits

Auteur : Robin Whittle
Australie
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

zkwm

zkwm — Ecrit dans une variable zk au taux-k avec mixage.

Description

Ecrit dans une variable zk au taux-k avec mixage.

Syntaxe

```
zkwm ksig, kndx [, imix]
```

Initialisation

imix (facultatif) -- indique si le mixage sera fait.

Ex#cution

ksig -- valeur à écrire dans l'espace zk.

kndx -- pointe sur la position zk ou za vers laquelle écrire.

zkwm est un opcode de mixage, il ajoute le signal à la valeur courante de la variable. Si aucun *imix* n'est specifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toutes autres valeurs entraînera un mixage.

Avertissement : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zacl* pour effacer ces variables.

Exemples

Voici un exemple de l'opcode zkwm. Il utilise le fichier *zkwm.csd* [examples/zkwm.csd].

Exemple 603. Exemple de l'opcode zkwm.

Voir les sections *Audio en Temps R#el* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in    No messages
-odac           -iadc     -d     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkwm.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
  ; Generate a k-rate signal.
  ; The signal goes from 30 to 20,000 then back to 30.
  kramp linseg 30, p3/2, 20000, p3/2, 30

  ; Mix the signal into the zk variable #1.
  zkwm kramp, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
  ; Generate another k-rate signal.
  ; This is a low frequency oscillator.
  klfo lfo 3500, 2

  ; Mix this signal into the zk variable #1.
  zkwm klfo, 1
endin

; Instrument #3 -- generates audio output.
instr 3
  ; Read zk variable #1, containing a mix of both signals.
  kamp zkr 1

  ; Create a sine waveform. Its amplitude will vary
  ; according to the values in zk variable #1.
  a1 oscil kamp, 880, 1

  ; Generate the audio output.
  out a1

  ; Clear the zk variable, get it ready for
  ; another pass.
  zkcl 0, 1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 5 seconds.
i 1 0 5
; Play Instrument #2 for 5 seconds.
i 2 0 5
; Play Instrument #3 for 5 seconds.
i 3 0 5
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

zaw, *zawm*, *ziw*, *ziwm*, *zkcl*, *zkw*, *zkr*

Crédits

Auteur : Robin Whittle
Australie

Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

Instructions de Partition et Routines GEN

Instructions de Partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section marquée
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des évènements de la partition
- *x* - Ignore le reste de la section courante
- *{* - Commence une boucle imbriquable, sans section
- *}* - Termine une boucle imbriquable, sans section

Instruction a (ou Instruction Avancer)

a — Avancer le temps de la partition de la quantité spécifiée.

Description

Provoque l'avancement du temps de la partition de la quantité spécifiée sans produire d'échantillons sonores.

Syntaxe

```
a p1  p2  p3
```

Exécution

p1   Non significatif. Habituellement zéro.
p2   Date en pulsations à laquelle l'avance doit commencer.
p3   Nombre de pulsations duquel il faut avancer sans produire de son.
p4 |
p5 | Non significatifs.
p6 |
.
.

Considérations Spéciales

Cette instruction permet d'avancer le compteur de pulsations dans une partition sans générer les échantillons sonores correspondants. On peut l'utiliser quand une section de la partition est incomplète (le début ou le milieu sont manquants) et que l'on ne souhaite pas générer et écouter une longue période de silence.

p2, date d'activation, et p3, nombre de pulsations, sont traités comme dans l'*instruction i*,en tenant compte du tri et des modifications par les *instructions t*.

Une *instruction a* sera insérée temporairement dans la partition par la fonction Score Extract lorsque l'extrait commence après le début de la Section. Ceci afin de conserver le compte de pulsations de la partition originale pour les messages de pic d'amplitude qui sont rapportés sur la console de l'utilisateur.

A chaque exécution d'un orchestre lorsqu'une *instruction a* est rencontrée, sa présence et son effet son rapportés sur la console de l'utilisateur.

Instruction b

b — Cette instruction réinitialise l'horloge.

Description

Cette instruction réinitialise l'horloge.

Syntaxe

```
b p1
```

Exécution

p1 -- Spécifie comment l'horloge doit être réglée.

Considérations Spéciales

p1 est le nombre de pulsations par lequel les valeurs p2 des *instructions i* suivantes sont modifiées. Si p1 est positif, l'horloge est avancée, et les notes suivantes apparaissent plus tard, le nombre de pulsations spécifié par p1 étant ajouté au p2 des notes. Si p1 est négatif, l'horloge est retardée, et les notes suivantes apparaissent plus tôt, le nombre de pulsations spécifié par p1 étant soustrait du p2 des notes. L'effet n'est pas cumulatif. L'horloge est réinitialisée avec chaque *instruction b*. Si p1 = 0, l'horloge revient à sa position initiale, et les notes suivantes apparaissent à leur position spécifiée en p2.

Exemples

```
i1      0       2
i1      10      888

b 5                             ; "avance" l'horloge
i2      1       1      440      ; date de début = 6
i2      2       1      480      ; date de début = 7

b -1                            ; retarde l'horloge
i3      3       2      3.1415   ; date de début = 2
i3      5.5     1      1.1111   ; date de début = 4.5

b 0                             ; réinitialise l'horloge à la normale
i4      10      200    7        ; date de début = 10
```

Crédits

Explication suggérée et exemple fourni par Paul Winkler. (Version 4.07 de Csound)

Instruction e

e — On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

Description

On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

Syntaxe

```
e [temps]
```

Exécution

Le premier p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la dernière section de la partition. Cette date doit être après le dernier évènement sinon elle n'aura pas d'effet. Les instruments "actifs en permanence" se termineront à cette date. Cette manière d'allonger la section est utile pour éviter les coupures prématurées de chute de réverbération ou d'autres effets.

Considérations Spéciales

L'*instruction e* est contextuellement identique à une *instruction s*. De plus, l'*instruction e* termine toute génération de signal (y compris une exécution indéfinie) et ferme tous les fichiers d'entrée et de sortie.

Si une *instruction e* intervient avant la fin de la partition, toutes les lignes suivantes de la partition seront ignorées.

Dans un fichier de partition pas encore trié, l'*instruction e* est facultative. Si un fichier de partition n'a pas d'*instruction e*, alors la fonction Sort en fournira une.

Instruction f (ou Instruction de Table de Fonction)

f — Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN.

Description

Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN pour utilisation par des instruments.

Syntaxe

```
f p1  p2  p3  p4  p5 ... PMAX
```

Exécution

p1 -- Numéro de table sous lequel la fonction mémorisée sera connue. Un nombre négatif signifie une demande de destruction de la table.

p2 -- Date d'activation de la génération de la fonction (ou de sa destruction) en pulsations.

p3 -- Taille de la table de la fonction (c'est-à-dire nombre de points). Doit être une puissance de 2, ou une puissance de 2 plus 1 si ce nombre est positif. La taille de table maximale est de 16777216 (2^{24}) points.

p4 -- Numéro de la routine GEN à appeler (voir *ROUTINES GEN*). Une valeur négative supprimera la normalisation.

p5 ... PMAX -- Paramètres dont la signification est déterminée par la routine GEN particulière.

Considérations Spéciales

Les tables de fonction sont des tableaux de valeurs en virgule flottante. On peut créer une simple onde sinusoïdale avec cette ligne :

f 1 0 1024 10 1

Cette table utilise *GEN10* pour son remplissage.

Historiquement, à cause des contraintes des anciennes plates-formes, Csound ne pouvait accepter que des tables dont la taille était une puissance de deux. Cette limitation a été levée dans les récentes versions, et l'on peut créer des tables de n'importe quelle taille. Cependant, pour créer une table dont la taille n'est pas une puissance de deux (ou une puissance de deux plus un), il faut spécifier la taille comme un nombre négatif.

Note

Il y a des opcodes qui n'accepteront pas des tables dont la taille n'est pas une puissance de deux, car ils comptent sur cela pour leur optimisation interne.

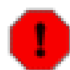
Pour les tableaux dont la longueur est une puissance de 2, l'allocation d'espace mémoire est toujours pré-

vue pour 2^n points plus un *point de garde*. La valeur du point de garde, utilisée pour la lecture avec interpolation, peut être fixée automatiquement selon le but de la table : si la *taille* est une puissance de 2 exacte, le point de garde sera une copie du premier point ; cela convient pour la *lecture cyclique avec interpolation* comme dans *oscili*, etc., et devrait même être utilisé pour la version sans interpolation *oscil* pour rester consistant. Si la *taille* est fixée à $2^n + 1$, le point de garde prolongera automatiquement le contour des valeurs de la table ; cela convient pour les fonctions à lecture non-cyclique comme dans *envplx*, *oscil1*, *oscil1i*, etc.

Les tables sont allouées dans la mémoire primaire, avec les données d'instrument. Le nombre maximum de tables était limité à 200. Ceci a changé et il n'est plus limité que par la quantité de mémoire disponible. (Actuellement il y a une limitation logicielle de 300, qui est augmentée automatiquement selon les besoins).

On peut supprimer une table de fonction existante par une *instruction f* contenant un p1 négatif et une date d'activation adéquate. Une table de fonction est également supprimée par la génération d'une autre table avec le même p1. Les fonctions ne sont pas automatiquement effacées à la fin d'une section de partition.

La date p2 est traitée de la même manière que dans l'*instruction i* en tenant compte du tri et des modifications par les *instructions t*. Si une *instruction f* et une *instruction i* ont le même p2, le tri donnera la priorité à l'*instruction f* afin que le table de fonction soit disponible pendant l'initialisation de la note.



Avertissement

Le nombre maximum de p-champs acceptés dans la partition est déterminé par PMAX (une variable de compilation). PMAX vaut actuellement 1000. Cela peut éliminer des valeurs entrées au moyen de *GEN02*. Pour contourner cette limitation, utiliser *GEN23* ou *GEN28* pour lire les valeurs à partir d'un fichier.

On peut utiliser une *instruction f 0* (avec zéro en p1 et p2 positif) pour créer une date sans action associée. De tels marqueurs temporels sont utiles pour remplir une section de partition (voir l'*instruction s*) et pour lancer une exécution de Csound à partir d'évènements en temps réel (par exemple en n'utilisant que des entrées MIDI sans évènements de partition). La durée indique le nombre de secondes de l'exécution de Csound. Si l'on veut que Csound tourne pendand 10 heures, on utilisera :

f0 36000

La manière la plus simple de remplir une table (f1) avec des 0 est :

f1 0 xx 2 0

where xx = table size.

La manière la plus simple de remplir une table (f1) avec n'importe quelle valeur unique est :

f1 0 xx -7 yy xx yy

où xx = taille de la table et yy = n'importe quelle valeur unique

Dans les deux exemple ci-dessus, la taille de la table (p3) doit être une puissance de 2 ou une puissance-de-2 + 1.

Voir aussi

ROUTINES GEN

Crédits

Mise à jour en août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 tables de fonction.

Instruction i (Instruction d'Instrument ou de Note)

i — Active un instrument à une date précise et pour une certaine durée.

Description

Cette instruction est nécessaire pour activer un instrument à une date précise et pour une certaine durée. Les valeurs des champs de paramètre sont passées à cet instrument avant son initialisation, et demeurent valides durant toute son exécution.

Syntaxe

```
i  p1  p2  p3  p4  ...
```

Initialisation

p1 -- Numéro d'instrument, habituellement un nombre entier non négatif. Une partie décimale facultative permet d'ajouter une étiquette indiquant des liaisons entre des notes particulières d'aggrégats consécutifs. Un p1 négatif (incluant une étiquette) peut être utilisé pour faire cesser une note « tenue » particulière.

p2 -- Date de début en unités arbitraires appelées pulsations.

p3 -- Durée en pulsations (habituellement positive). Une valeur négative démarre une note tenue (voir aussi *ihold*). On peut aussi utiliser une valeur négative pour les instruments 'toujours actifs' comme la réverbération. Ces notes ne sont pas terminées par des *instruction s*. Une valeur nulle provoquera une passe d'initialisation sans exécution (voir aussi *instr*).

p4 ... -- Paramètres dont la signification est déterminée par l'instrument.

Exécution

Une pulsation vaut une seconde, à moins qu'il n'y ait une *instruction t* dans cette section de la partition ou une *option -t* dans la ligne de commande.

Les dates de début ou d'action sont relatives au début d'une section (voir l'*instruction s*), qui reçoit la date 0.

Dans une section, les instructions de note peuvent être placées dans n'importe quel ordre. Avant d'être envoyées à l'orchestre, les instructions non triées de la partition doivent être traitées par la fonction Sort, qui les ordonnera par valeurs de p2 croissantes. Les notes ayant la même valeur en p2 seront triées par p1 croissants ; si elles ont le même p1, alors par p3 croissants.

Les notes peuvent être superposées, c'est-à-dire qu'un seul instrument peut jouer n'importe quel nombre de notes simultanément. (Les copies nécessaires de l'espace de données de l'instrument seront allouées dynamiquement par le chargeur de l'orchestre). Chaque note se termine normalement à la fin de sa durée en p3, ou à la réception d'un signal MIDI noteoff. Un instrument peut modifier sa propre durée en changeant la valeur de son p3 pendant l'initialisation de la note, ou en se prolongeant lui-même par l'action d'une unité *linenr* ou *xtratim*.

Un instrument peut être activé et réglé pour une durée indéfinie soit en lui donnant un p3 négatif soit en incluant un *ihold* dans le code de son temps-i. Si une note tenue est active, une *instruction i avec un p1*

correspondant ne provoquera pas une nouvelle allocation mais prendra l'espace de données de la note tenue. Les nouveaux p-champs (y compris p3) seront maintenant effectifs, et une passe de temps-i sera exécutée pendant laquelle les unités peuvent être soit initialisées à nouveau soit autorisées à continuer comme requis pour une note liée (voir *tigoto*). Une note tenue peut être suivie soit par une autre note tenue soit par une note de durée finie. Une note tenue continuera à être jouée au-delà des fins de section (voir l'*instruction s*). Elle est arrêtée seulement par un *turnoff* ou par une *instruction i* avec un p1 négatif correspondant ou par une *instruction e*.

Il est possible d'avoir plusieurs instances (habituellement, mais pas forcément, des notes de hauteurs différentes) du même instrument, tenues simultanément, via des valeurs négatives de p3. L'instrument peut ensuite recevoir de nouveaux paramètres de la partition. C'est utile pour éviter de longs *linseg* codés en dur, et peut être accompli en ajoutant une partie décimale au numéro de l'instrument.

Par exemple, pour tenir trois copies de l'instrument 10 dans un accord :

```
i10.1    0    -1    7.00
i10.2    0    -1    7.04
i10.3    0    -1    7.07
```

Les instructions *i* suivantes peuvent faire référence aux mêmes instances de note active, et si la définition de l'instrument est faite proprement, les nouveaux p-champs peuvent servir à changer le caractère des notes jouées. Par exemple, pour faire glisser l'accord précédent d'une octave vers le haut et le laisser résonner :

```
i10.1    1    1    8.00
i10.2    1    1    8.04
i10.3    1    1    8.07
```

Astuce

Pour la terminaison des notes, il faut tenir compte du fait que i 1.1 == i 1.10 et que i 1.1 != i 1.01. Le nombre maximum de positions décimales que l'on peut utiliser dépend de la précision avec laquelle Csound a été compilé (Voir *Csound Double (64 bit) ou Float (32 bit)*)

La définition de l'instrument doit prendre ceci en compte, cependant, spécialement si l'on veut éviter les clics (voir l'exemple ci-dessous).

Noter que la notation décimale du numéro d'intrument ne peut pas être utilisée en conjonction avec le MIDI en temps réel. Dans ce cas, l'instrument serait monodique tant qu'une note est tenue.

Les notes liées à des instances précédentes du même instrument, devraient éviter la plus grande partie de l'initialisation au moyen de *tigoto*, sauf pour les valeurs entrées dans la partition. Par exemple, tous les opcodes de lecture de table dans l'instrument, seront habituellement sautés en initialisation, car ils mémorisent en interne leur phase. Si celle-ci est brutalement modifiée, on entendra des clics en sortie.

Noter que plusieurs opcodes (comme *delay* et *reverb*) sont prévus pour une initialisation facultative. Pour utiliser cette possibilité, l'*opcode tival* est approprié. Ainsi, il n'y a pas besoin de les escamoter par un saut *tigoto*.

A partie de la version 3.53 de Csound, les chaînes sont reconnues dans les p-champs des opcodes qui les acceptent (*convolve, adsyn, diskin,* etc.). Il ne peut y avoir qu'une seule chaîne par ligne de la partition.

Considérations Spéciales

Le numéro d'instrument maximum était de 200. Cela a changé et il n'est plus limité que par la capacité mémoire (actuellement, il y a une limite logicielle de 200 ; celle-ci est étendue automatiquement si nécessaire).

Exemples

Voici un instrument capable de découvrir s'il est lié à une note précédente (*tival* retourne 1), et s'il doit être tenu (p3 négatif). L'attaque et la chute sont traitées en conséquence :

```
instr 10

  icps      init      cpspch(p4)            ; Reçoit la hauteur cible de l'évènement de partition
  iportime init       abs(p3)/7             ; La durée du portamento dépend de celle de la note
  iamp0     init      p5                    ; Fixe l'amplitude par défaut
  iamp1     init      p5
  iamp2     init      p5

  itie      tival                           ; Teste si cette note est liée,
  if itie   ==  1     igoto nofadein        ; si non alors entrée progressive
  iamp0     init      0

nofadein:
  if p3     < 0       igoto nofadeout       ; Teste si cette note est tenue,
  iamp2     init      0                     ; si non alors disparition progressive

nofadeout:
  ; Maintenant générer l'amplitude à partir des valeurs fixées :
  kamp      linseg    iamp0, .03, iamp1, abs(p3)-.03, iamp2

  ; Passe le reste de l'initialisation pour une note liée :
            tigoto    tieskip

  kcps      init      icps                  ; Initialise la hauteur pour une note non liée
  kcps      port      icps, iportime, icps  ; Glisse vers la hauteur cible

  kpw       oscil     .4, rnd(1), 1, rnd(.7) ; Un oscillateur simple en dent de scie
  ar        vco       kamp, kcps, 3, kpw+.5, 1, 1/icps

  ; (Utilisé pour tester - on peut fixer ipch à cpspch(p4+2)
  ;       et voir le spectre en sortie)
  ;       ar oscil kamp, kcps, 1

           out        ar

tieskip:                                   ; Passe certaines initialisations pour une note liée

endin
```

Une simple partition avec trois instances de l'instrument ci-dessus :

```
  f1   0 8192  10 1            ; Sinus

  i10.1     0     -1     7.00     10000
  i10.2     0     -1     7.04
  i10.3     0     -1     7.07
  i10.1     1     -1     8.00
  i10.2     1     -1     8.04
  i10.3     1     -1     8.07
  i10.1     2      1     7.11
  i10.2     2      1     8.04
  i10.3     2      1     8.07
  e
```

Crédits

Texte supplémentaire (Version 4.07 de Csound) expliquant les notes liées, publié par Rasmus Ekman d'après une note de David Kirsh, postée sur la liste de courrier électronique de Csound. Instrument en exemple par Rasmus Ekman.

Mise à jour Août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 instruments.

Instruction m (Instruction de Marquage)

m — Positionne une marque nommée dans la partition.

Description

Positionne une marque nommée dans la partition, qui peut être utilisée par une *instruction n*.

Syntaxe

```
m p1
```

Initialisation

p1 -- Nom de la marque.

Exécution

Peut être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Nouveau dans la version 3.48 de Csound

Instruction n

n — Répète une section.

Description

Répète une section depuis l'*instruction m* référencée.

Syntaxe

```
n p1
```

Initialisation

p1 -- Nom de la marque à répéter.

Exécution

Peut-être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

Credits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Nouveau dans la version 3.48 de Csound

Instruction q

q — Cette instruction peut être utilisée pour rendre un instrument silencieux.

Description

Cette instruction peut être utilisée pour rendre un instrument silencieux.

Syntaxe

```
q p1  p2  p3
```

Exécution

p1 -- Numéro de l'instrument à rendre muet/sonore.

p2 -- Date d'action en pulsations.

p3 -- Détermine si l'instrument doit être rendu silencieux ou sonore. La valeur 0 signifie silencieux, toute autre valeur signifie sonore.

Noter que ceci n'affecte pas les instruments déjà actifs à la date *p2*. Ça bloque toute tentative d'en démarrer un après cette date.

Instruction r (Instruction Répéter)

r — Débute une section répétée.

Description

Débute une section répétée, qui dure jusqu'à la prochaine instruction *s*, *r* ou *e*.

Syntaxe

```
r p1 p2
```

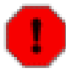
Initialisation

p1 -- Nombre de répétitions de la section demandé.

p2 -- Macro(nom) pour indexer chaque répétition (facultatif).

Exécution

Afin de rendre les sections plus souples qu'une simple édition, la macro nommée en p2 reçoit la valeur 1 à la première boucle dans la section, 2 à la seconde, 3 à la troisième, etc. On peut l'utiliser pour changer la valeur des p-champs, ou l'ignorer.



Avertissement

A cause de sérieux problèmes d'interaction avec l'expansion de macro, les sections doivent commencer et finir dans le même fichier, à l'extérieure de toute macro.

Exemples

Voici un exemple d'instruction r. Il utilise le fichier *r.sco* [examples/r.csd].

Exemple 604. Exemple d'instruction r.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o r.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; The score's p4 parameter has the number of repeats.
```

```
  kreps = p4
  ; The score's p5 parameter has our note's frequency.
  kcps = p5

  ; Print the number of repeats.
  printks "Repeated %i time(s).\\n", 1, kreps

  ; Generate a nice beep.
  a1 oscil 20000, kcps, 1
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; We'll repeat this section 6 times. Each time it
; is repeated, its macro REPS_MACRO is incremented.
r6 REPS_MACRO

; Play Instrument #1.
; p4 = the r statement's macro, REPS_MACRO.
; p5 = the frequency in cycles per second.
i 1 00.10 00.10 $REPS_MACRO 1760
i 1 00.30 00.10 $REPS_MACRO 880
i 1 00.50 00.10 $REPS_MACRO 440
i 1 00.70 00.10 $REPS_MACRO 220

; Marks the end of the section.
s

e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK
Avril 1998

Nouveau dans la version 3.48 de Csound

Exemple écrit par Kevin Conder

Instruction s

s — Marque le fin d'une section.

Description

L'*instruction s* marque le fin d'une section.

Syntaxe

```
s [temps]
```

Initialisation

Le premier p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la section. Cette date doit être après la fin du dernier évènement de la section sinon elle n'aura pas d'effet. On peut l'utiliser pour créer une pause avant le début de la section suivante ou pour permettre aux instruments "actifs en permanence" tels que les effets de jouer seuls pendant une certaine durée.

Exécution

Le tri des *instructions i*, des *instructions f* et des *instructions a* par date d'action est effectué section par section.

La modification temporelle par l'*instruction t* est faite section par section.

Toute les dates d'action à l'intérieur d'une section sont relatives à son début. Une instruction de section établit un nouveau temps relatif de 0, mais n'a pas d'autres effets de réinitialisation (par exemple les tables de fonction mémorisées sont préservées par delà les limites de section).

On considère qu'une section est complète lorsque toutes les dates d'action et toutes les durées finies ont été satisfaites. (C'est-à-dire que la "longueur" d'une section est déterminée par la dernière action apparue ou par l'arrêt du système). Une section peut être allongée par l'utilisation d'une *instruction f0* ou en fournissant la valeur de *p1*facultative à l'instruction *s*.

A la fin d'une section, le système provoque automatiquement le nettoyage des instruments inactifs et de leur espace de données.

Note

- Puisque les instructions de partition sont traitées section par section, la quantité de mémoire requise dépend du nombre maximum d'instructions de partition dans une section. L'allocation de mémoire est dynamique, et l'utilisateur sera informé chaque fois que des blocs de mémoire supplémentaires sont demandés pendant le traitement de la partition.
- Pour la dernière section d'une partition, l'*instruction s* est facultative ; l'*instruction e* peut être utilisée à la place.

Instruction t (Instruction de Tempo)

t — Fixe le tempo.

Description

Cette instruction fixe le tempo et spécifie les accelerando et les ritardando de la section courante. Ceci est réalisé en convertissant les pulsations en secondes.

Syntaxe

```
t  p1  p2  p3  p4 ... (illimité)
```

Initialisation

p1 -- Doit être zéro.

p2 -- Tempo initial en pulsations par minute.

p3, p5, p7,... -- Dates en pulsations (en ordre non décroissant).

p4, p6, p8,... -- Tempi pour les dates en pulsations référencées.

Exécution

Les dates et le Tempo pour chaque date sont donnés en couples ordonnés qui définissent des points sur un graphe « date, tempo ». (L'axe du temps est ici en pulsations et n'est donc pas nécessairement linéaire). Le taux de pulsations d'une section peut être pensé comme un mouvement d'un point à un autre de ce graphe : un mouvement entre deux points à la même hauteur signifie un tempo constant, tandis qu'un mouvement entre deux points de hauteurs différentes traduit un accelerando ou un ritardando selon le cas. Le graphe peut contenir des discontinuités : deux points ayant la même date mais des tempi différents provoqueront un changement de tempo instantané.

Le mouvement entre différents tempi sur des durées non nulles est inversement linéaire. Cela veut dire qu'un accelerando entre deux tempi M1 et M2 procède par interpolation linéaire des durées de chaque pulsation entre 60/M1 et 60/M2.

Le premier tempo doit être donné pour la pulsation 0.

Une fois assigné, un tempo sera effectif à partir de cette date à moins d'être influencé par un tempo suivant, ainsi, le dernier tempo spécifié sera actif jausqu'à la fin de la section.

Une *instruction t* ne s'applique que dans la section dans laquelle elle apparaît. Une seule *instruction t* est pertinente dans une section ; elle peut être placée n'importe où dans la section. Si une section de partition ne contient pas d'*instruction t*, les pulsations sont alors interprétées comme des secondes (c'est-à-dire avec une instruction *t 0 60* implicite).

Nota Bene. Si la commande de Csound comprend une *option -t*, le tempo interprété de toutes les *instruction t* de la partition sera remplaçé par le tempo de la ligne de commande.

Instruction v

v — Permet une modification temporelle variable localement des évènements de la partition.

Description

L'*instruction v* permet une modification temporelle variable localement des évènements de la partition.

Syntaxe

```
v p1
```

Initialisation

p1 -- facteur de modification temporelle (doit être positif).

Exécution

L'*instruction v* prend effet avec l'*instruction i* qui la suit, et reste effective jusqu'à la prochaine *instruction v*, *instruction s*, ou *instruction e*.

Exemples

La valeur de p1 est utilisée comme un coefficient multiplicatif de la date de début (p2) des *instructions i* suivantes.

```
i1   0 1  ; note1
v2
i1   1 1  ; note2
```

Dans cet exemple, la deuxième note apparaît deux pulsations après la première note, et elle est deux fois plus longue.

Bien que l'*instruction v* soit semblable à l'*instruction t*, l'*instruction v* agit localement. Cela veut dire que *v* n'affecte que les notes suivantes, et que son effet peut être annulé ou changé par une autre *instruction v*.

Les valeurs reportées ne sont pas affectées par l'*instruction v* (voir *Carry*).

```
i1   0 1  ; note1
v2
i1   1 .  ; note2
i1   2 .  ; note3
v1
i1   3 .  ; note4
i1   4 .  ; note5
e
```

Dans cet exemple, note3 et note5 sont jouées simulanément, tandis que note4 est jouée avant note3, c'est-à-dire à sa place initiale. Les durées sont inchangées.

```
i1    0 1
v2
i.    + .
i.    . .
```

Dans cet exemple, l'*instruction v* n'a aucun effet.

Instruction x

x — Ignore le reste de la section courante.

Description

On peut utiliser cette instruction pour ignorer le reste de la section courante.

Syntaxe

```
x valeurbidon
```

Initialisation

Tous les p-champs sont ignorés.

{ Statement

{ — Commence une boucle imbriquable, sans section.

Description

On peut utiliser les *instructions {* et *}* pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des évènements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

Syntaxe

```
{ p1 p2
```

Initialisation

p1 -- Nombre de répétitions de la boucle.

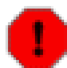
p2 -- Un nom de macro qui est automatiquement défini au début de la boucle et dont la valeur est incrémentée à chaque répétition (facultatif). La valeur initiale est zéro et la valeur finale est (p1 - 1).

Exécution

L'*instruction {* est utilisée conjointement avec l'*instruction }* pour définir des groupes d'évènements de partition qui se répètent. Une boucle de partition commence par l'*instruction {* qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'évènements (y compris des sauts de section) et il se termine par une *instruction }* ayant sa propre ligne. L'*instruction }* ne prend pas de paramètre.

Le terme "boucle" n'implique aucune sorte de succession temporelle pour les itérations de la boucle. Autrement dit, les valeurs p2 des évènements à l'intérieur de la boucle ne sont pas incrémentées automatiquement de la longueur de la boucle à chaque répétition. C'est un avantage car cela permet de définir facilement des groupes d'évènements simultanés. La macro de boucle peut être utilisée avec des *expressions de partition* pour incrémenter les dates de début d'évènements ou pour faire varier les évènements de toute autre manière désirée à chaque répétition. Noter que à la différence de l'*instruction r*, la valeur de la macro au premier passage dans la boucle est zéro (0), pas un (1). Ainsi la valeur finale est inférieure d'une unité au nombre de répétitions.

Les boucles de partition sont un outil très puissant. Bien que semblables à l'outil de répétition de section (l'*instruction r*), leur principal avantage est que les évènements de partition dans les itérations successives de la boucle ne sont pas séparés par une fin de section. Ainsi, il est possible de créer plusieurs boucles qui se chevauchent dans le temps. Les boucles peuvent aussi être imbriquées jusqu'à une profondeur de 39 niveaux.



Avertissement

En raison de sérieux problèmes d'interaction avec l'expension de macro, les boucles doivent commencer et se terminer dans le même fichier, et pas à l'intérieur d'une macro.

Exemples

Voici quelques exemples des *instructions { et }*.

Exemple 605. Répétition sequentielle d'une phrase de trois notes, quatre fois.

```
{ 4 CNT

i1  [0.00 + 0.75 * $CNT.]   0.2   220
i1  [0.25 + 0.75 * $CNT.]   .     440
i1  [0.50 + 0.75 * $CNT.]   .     880

}
```

interprété comme

```
i1  0.00   0.2   220
i1  0.25   .     440
i1  0.50   .     880

i1  0.75   0.2   220
i1  1.00   .     440
i1  1.25   .     880

i1  1.50   0.2   220
i1  1.75   .     440
i1  2.00   .     880

i1  2.25   0.2   220
i1  2.50   .     440
i1  2.75   .     880
```

Exemple 606. Création d'un groupe d'harmoniques simultanés.

Dans cet exemple, *p4* contient la fréquence de la note.

```
{ 8 PARTIAL
i1  0  1   [100 * ($PARTIAL. + 1)]
}
```

interprété comme

```
i1  0  1   100
i1  0  1   200
i1  0  1   300
i1  0  1   400
i1  0  1   500
i1  0  1   600
i1  0  1   700
i1  0  1   800
```

Voici un exemple complet des *instructions { et }*. Il utilise le fichier *leftbrace.csd* [examples/leftbrace.csd].

Exemple 607. Un exemple de boucles imbriquées pour créer plusieurs clusters inharmoniques de sinus.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
nchnls = 2

gaReverbSend init 0

; a simple sine wave partial
instr 1
    idur   =       p3
    iamp   =       p4
    ifreq  =       p5
    aenv   linseg  0.0, 0.1*idur, iamp, 0.6*idur, iamp, 0.3*idur, 0.0
    aosc   oscili  aenv, ifreq, 1
           vincr   gaReverbSend, aosc
endin

; global reverb instrument
instr 2
    al, ar reverbsc gaReverbSend, gaReverbSend, 0.85, 12000
           outs     gaReverbSend+al, gaReverbSend+ar
           clear    gaReverbSend
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

{ 4 CNT
  { 8 PARTIAL
      ;   start time     duration            amplitude          frequency

      i1  [0.5 * $CNT.]  [1 + ($CNT * 0.2)]  [500 + (~ * 200)]  [800 + (200 * $CNT.) + ($PARTIAL. * 20
  }
}

i2 0 6
e

</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

} Statement

} — Termine une boucle imbriquable, sans section.

Description

On peut utiliser les *instructions {* et *}* pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des évènements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

Syntaxe

```
}
```

Initialisation

Tous les p-champs sont ignorés.

Exécution

L'*instruction }* est utilisée conjointement avec l'*instruction {* pour définir des groupes d'évènements de partition qui se répètent. Une boucle de partition commence par l'*instruction {* qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'évènements (y compris des sauts de section) et il se termine par une *instruction }* ayant sa propre ligne. L'*instruction }* ne prend pas de paramètre.

Voir la documentation de l'*instruction {* pour plus de details.

Exemples

Voir les exemples de l'article sur l'*instruction {*.

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

Routines GEN

Les routines GEN sont utilisées comme générateurs de données pour les tables de fonction. Quand une table de fonction est créée au moyen de *l'instruction de partition f* la fonction GEN est donnée dans le quatrième argument. Un numéro de GEN négatif implique que la fonction ne sera pas normalisée et qu'elle gardera ses valeurs originales.

Générateurs Sinus/Cosinus :

- *GEN09* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.

- *GEN10* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN11* - Ensemble additif de partiels cosinus.
- *GEN19* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN30* - Génère des partiels harmoniques en analysant une table existante.
- *GEN33* - Génère des formes d'onde complexes en mélangeant des sinus.
- *GEN34* - Génère des formes d'onde complexes en mélangeant des sinus.

Générateurs par Morceaux de Ligne/Exponentielle

- *GEN05* - Construit des fonctions à partir de morceaux de courbes exponentielles.
- *GEN06* - Génère une fonction composée de morceaux de polynômes cubiques.
- *GEN07* - Construit des fonctions à partir de morceaux de lignes droites.
- *GEN08* - Génère une courbe spline cubique par morceaux.
- *GEN16* - Crée une table depuis une valeur initiale jusqu'à une valeur terminale.
- *GEN25* - Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).
- *GEN27* - Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

Routines GEN d'Accès Fichier :

- *GEN01* - Transfère des données d'un fichier son dans une table de fonction.
- *GEN23* - Lit des valeurs numériques à partir d'un fichier texte.
- *GEN28* - Lit un fichier texte qui contient une trajectoire paramétrée par le temps.

Routines GEN d'Accès à des Valeurs Numériques

- *GEN02* - Transfère les données des p-champs dans une table de fonction.
- *GEN17* - Crée une fonction en escalier à partir des paires x-y données.
- *GEN52* - Crée une table multi-canaux entrelacés à partir des tables source indiquées, dans le format attendu par l'opcode *ftconv*.

Routines GEN de Fonction Fenêtre

- *GEN20* - Génère les fonctions de différentes fenêtres.

Routines GEN de Fonction Aléatoire

- *GEN21* - Génère les tables de différentes distributions aléatoires.
- *GEN40* - Génère une distribution aléatoire à partir d'un histogramme.
- *GEN41* - Génère une liste aléatoire de paires numériques.
- *GEN42* - Génère une distribution aléatoire d'intervalles discrets de valeurs.
- *GEN43* - Charge un fichier PVOCEX contenant une analyse VP.

Routines GEN de Distorsion Linéaire

- *GEN03* - Génère une table de fonction en évaluant un polynôme.
- *GEN13* - Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.
- *GEN14* - Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.
- *GEN15* - Crée deux tables de fonctions polynomiales mémorisées.

Routines GEN de Dimensionnement de l'Amplitude

- *GEN04* - Génère une fonction de normalisation.
- *GEN12* - Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.
- *GEN24* - Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

Routines GEN de Mixage

- *GEN18* - Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.
- *GEN31* - Mélange n'importe quelle forme d'onde définie dans une table existante.
- *GEN32* - Mélange n'importe quelle forme d'onde, reéchantillonnée soit par TFR soit par interpolation linéaire.

Routines GEN de Hauteur et d'Accordage

- *GEN51* - Remplit une table avec une échelle micro-tonale entièrement personnalisée, à la manière des opcodes *cpstun*, *cpstuni* et *cpstmid*.

Routines GEN Nommées

On peut ajouter des routines GEN à Csound au moyen de plugins de fonction GEN. Il y a actuellement un seul plugin GEN qui fournit les fonctions exponentielle et tangente hyperbolique. Ces fonctions GEN ne sont pas appelées par un numéro, mais par un nom.

- "tanh" - remplit une table à partir d'une formule de tangente hyperbolique.
- "exp" - remplit une table à partir d'une formule de tangente hyperbolique.

GEN01

GEN01 — Transfère des données d'un fichier son dans une table de fonction.

Description

Ce sous-programme transfère des données d'un fichier son dans une table de fonction.

Syntaxe

```
f#  date  taille  1  codfic  decal  format  canal
```

Exécution

taille -- nombre de points dans la table. Ordinairement une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*) ; la taille de table maximale est de 16777216 (2^{24}) points. L'allocation de mémoire pour la table peut être *différée* en mettant ce paramètre à 0 ; la taille allouée est alors le nombre de points dans le fichier (probablement pas une puissance de 2), et la table n'est pas utilisable par les oscillateurs normaux, mais par l'unité *loscil*. Le fichier son peut aussi être mono ou stéréo.

codfic -- entier ou chaîne de caractères dénotant le nom du fichier son source. Un entier dénote le fichier *soundin.codfic* ; une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) enfin par SFDIR. Voir aussi *soundin*.

decal -- commence à lire à *decal* secondes dans le fichier.

canal -- numéro du canal à lire. 0 indique de lire tous les canaux.

format -- spécifie le format des données audio :

1 - 8-bit caractères signés	4 - 16-bit entiers courts
2 - 8-bit octets A-law	5 - 32-bit entiers longs
3 - 8-bit octets U-law	6 - 32-bit flottants

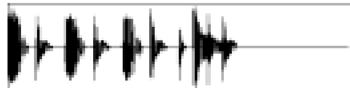
Si *format* = 0 le format des échantillons est lu dans l'en-tête du fichier son ou, par défaut depuis l'option *-o* de la ligne de commande de Csound.

Note

- La lecture s'arrête à la fin du fichier ou lorsque la table est pleine. Les cellules de la table non remplies contiendront des zéros.
- Si p4 est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.

Exemples

Voici un exemple simple de la routine GEN01. Il utilise les fichiers *gen01.csd* [examples/gen01.csd], et *beats.wav* [examples/beats.wav]. Il utilise le fichier audio "beats.wav" dont voici le graphe :



Graphe de la forme d'onde générée par GEN01.

Exemple 608. Un exemple simple de la routine GEN01.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen01.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
   kamp = 30000
   kcps = 1
   ifn = 1
   ibas = 1

   ; Play the audio sample stored in Table #1.
   a1 loscil kamp, kcps, ifn, ibas
   out a1
endin


</CsInstruments>
<CsScore>

; Table #1: read an audio file (using GEN01).
f 1 0 131072 1 "beats.wav" 0 4 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de la routine GEN01. Csound calculera automatiquement la taille de la table parce que nous l'avons fixée à 0. Cet exemple utilise les fichiers *gen01computed.csd* [examples/gen01computed.csd] et *beats.wav* [examples/beats.wav].

Exemple 609. Un exemple de la routine GEN01 avec une taille de table calculée.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen01computed.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
   kamp = 30000
   kcps = 1
   ifn = 1
   ibas = 1

   ; Play the audio sample stored in Table #1.
   a1 loscil kamp, kcps, ifn, ibas
   out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an audio file (using GEN01).
; Since our table size is 0, Csound will compute it.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Exemples écrits par Kevin Conder

Décembre 2002. Remerciements à Kanata Motohashi pour la correction des erreurs dans les exemples.

Septembre 2003. Remerciements au Dr. Richard Boulanger pour avoir signalé les références au format de fichier AIFF. GEN01 fonctionnne aussi avec des fichiers WAV.

GEN02

GEN02 — Transfère les données des p-champs dans une table de fonction.

Description

Ce sous-programme transfère les données des p-champs dans une table de fonction.

Syntaxe

```
f # date taille 2 v1 v2 v3 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La taille de table maximale est de 16777216 (2^{24}) points.

v1, v2, v3, etc. -- valeurs à copier directement dans l'espace de la table. Le nombre de valeurs est limité par la variable de compilation *PMAX*, qui contrôle le nombre maximum de p-champs (actuellement 1000). Les valeurs copiées peuvent comprendre le point de garde de la table ; les cellules de la table non remplies contiendront des zéros.

Note

Si p4 (le numéro de la routine GEN) est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération. On utilisera habituellement la valeur -2 avec cette fonction GEN, afin que les valeurs ne soient pas normalisées.

Exemples

Voici un exemple simple de la routine GEN02. Il utilise le fichier *gen02.csd* [examples/gen02.csd]. Il place 12 valeurs plus une valeur de garde explicite pour lecture cyclique dans une table de taille égale à la puissance de 2 supérieure la plus proche. La normalisation est empêchée. Voici le graphe :

Graphe de la forme d'onde générée par GEN02.

Exemple 610. Un exemple simple de la routine GEN02.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen02.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kamp tablei kndx, ifn, ixmode

  ; Create a sine wave, use the Table #1 values to control
  ; the amplitude. This creates a sound with a long attack.
  a1 oscil kamp*30000, 440, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an envelope with a long attack (using GEN02).
f 1 0 16 2 0 1 2 3 4 5 6 7 8 9 10 11 0
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN17

Crédits

Décembre 2002. Merci à Rasmus Ekman, pour avoir corrigé la limite de la variable *PMAX*.

GEN03

GEN03 — Génère une table de fonction en évaluant un polynôme.

Description

Ce sous-programme génère une table de fonction en évaluant un polynôme en x sur un intervalle fixe et avec des coefficients spécifiés.

Syntaxe

```
f # date taille 3 xval1 xval2 c0 c1 c2 ... cn
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1.

xval1, xval2 -- limites gauche et droite de l'intervalle x sur lequel le polynôme est défini (*xval1* < *xval2*). Celles-ci produiront la 1ère valeur stockée et la (puissance-de-2 plus 1)ème valeur stockée respectivement dans la table de la fonction générée.

c0, c1, c2, ..., cn -- coefficients du polynôme d'ordre n

$C_0 + C_1x + C_2x^2 + . . . + C_nx^n$

Les coefficients peuvent être des nombres réels positifs ou négatifs ; un zéro dénote un terme manquant dans le polynôme. La liste de coefficients commence en p7, avec une limite maximale actuelle de 144 termes.

Note

- Le segment défini [fn(*xval1*), fn(*xval2*)] est distribué également. Ainsi une table de 512 points sur l'interval [-1,1] aura son origine à la cellule 257 (au début de la seconde moitié). Si le point de garde est requis, les deux valeurs fn(-1) et fn(1) existeront dans la table.
- *GEN03* est utile en conjonction avec *table* ou *tablei* pour le waveshaping audio (modification du son par distortion non-linéaire). Les coefficients pour produire un formant particulier à partir d'un index de lecture sinusoïdal d'amplitude connue peuvent être déterminés avant le traitement en utilisant des algorithmes tels que les formules de Tchebychev. Voir aussi *GEN13*.

Exemples

Voici un exemple simple de la routine GEN03. Il utilise le fichier *gen03.csd* [examples/gen03.csd]. Il remplit une table avec une fonction polynomiale du 4ème ordre sur l'intervale des x allant de -1 à 1. L'origine sera à la position décalée 512. La fonction est post-normalisée. Voici le graphe :

Graphe de la forme d'onde générée par GEN03.

Exemple 611. Un exemple simple de la routine GEN03.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen03.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kamp table kndx, ifn, ixmode

  ; Create a sine wave, use the Table #1 values to control
  ; the amplitude.
  a1 oscil kamp*30000, 440, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN03).
f 1 0 1025 3 -1 1 5 4 3 2 2 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN13, *GEN14* et *GEN15*.

GEN04

GEN04 — Génère une fonction de normalisation.

Description

Ce sous-programme génère une fonction de normalisation en examinant le contenu d'une table existante.

Syntaxe

```
f  #  temps  taille  4  source#  modesource
```

Initialisation

taille -- nombre de points dans la table. Une puissance-de-2 plus 1. Ne doit pas dépasser (sauf de 1) la taille de la table source examinée ; limitée à exactement la moitié de cette taille si modesource est de type décalage (voir ci-dessous).

source # -- numéro de table de la fonction stockée à examiner.

modesource -- une valeur codée, spécifiant comment la table source doit être parcourue pour obtenir la fonction de normalisation. Zéro indique que la source doit être parcourue de gauche à droite. Une valeur non nulle indique que la source a une structure bipolaire ; la lecture commencera au point médian et progressera vers les extrémités, par paires de points équidistants du centre.

Note

- La fonction de normalisation dérive de la progression des maxima absolus de la table source parcourue. La nouvelle table est créée de gauche à droite, en stockant des valeurs égales à 1/(maximum absolu lu jusque là). Les valeurs stockées commenceront ainsi par 1/(première valeur lue), et deviendront progressivement plus petites lorsque de nouveaux maxima seront rencontrés. Pour une table source normalisée (valeurs <= 1), les valeurs dérivées descendront de 1/(première valeur lue) jusqu'à 1. Si la première valeur lue est zéro, son inverse sera fixé à 1.
- la fonction de normalisation générée par *GEN04* n'est pas elle-même normalisée.
- *GEN04* est utile pour modifier l'échelle d'un signal dérivé d'une table afin qu'il ait une amplitude de crête consistante. On l'utilise particulièrement en waveshaping quand la porteuse (ou fonction d'indexation) a une amplitude inférieure à la moitié de l'échelle complète.

Exemples

```
f   2   0   512   4   1   1
```

Création d'une fonction de normalisation à utiliser en connexion avec le table 1 de l'exemple *GEN03*. Un décalage bipolaire à point médian est spécifié.

GEN05

GEN05 — Construit des fonctions à partir de morceaux de courbes exponentielles.

Description

Construit des fonctions à partir de morceaux de courbes exponentielles.

Syntaxe

```
f # date taille 5 a n1 b n2 c ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

a, b, c, etc. -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, . . . Elle doivent être non nulles et de même signe.

n1, n2, etc. -- longueurs des morceaux (nombre de positions mémorisées), dans les p-champs de numéros pairs. Ne peuvent pas être négatives, mais un zéro est significatif pour spécifier des formes d'onde discontinues (comme dans l'exemple ci-dessous). La somme *n1* + *n2* + ... sera normalement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées.

Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la (n + 1)ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

Exemples

Voici un exemple simple de la routine GEN05. Il utilise le fichier *gen05.csd* [examples/gen05.csd]. Il créera une jolie enveloppe d'amplitude percussive. Voici le graphe :

Graphe de la forme d'onde générée par GEN05.

Exemple 612. Un exemple simple de la routine GEN05.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen05.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kamp table kndx, ifn, ixmode

  ; Create a sine wave, use the Table #1 values to control
  ; the amplitude. This creates a nice percussive sound.
  a1 oscil kamp*30000, 440, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a percussive envelope (using GEN05).
f 1 0 64 5 1 2 120 60 1 1 0.001 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN06, *GEN07* et *GEN08*

Crédits

Exemple écrit par Kevin Conder

GEN06

GEN06 — Génère une fonction composée de morceaux de polynômes cubiques.

Description

Ce sous-programme génèrera une fonction composée de morceaux de polynômes cubiques, couvrant les points spécifiés trois par trois.

Syntaxe

```
f  #   date   taille   6   a   n1   b   n2   c   n3   d ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

a, c, e, ... -- les maxima ou les minima locaux des morceaux successifs, dépendant de la relation de ces points avec les inflexions adjacentes. Peuvent être positifs ou négatifs.

b, d, f, ... -- ordonnées des points d'inflexion aux extrémités des segments curvilignes successif. Peuvent être positifs ou négatifs.

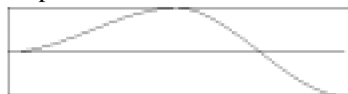
n1, n2, n3 ... -- nombre de valeurs stockées entre les points spécifiés. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des discontinuités. La somme *n1* + *n2* + ... sera normalement égale à *taille* pour les fonctions complètement spécifiées. (Pour des détails, voir *GEN05*).

Note

GEN06 construit une fonction stockée à partir de fonctions polynomiales cubiques. Les morceaux groupent les valeurs d'ordonnée par groupes de 3 : point d'inflexion, maximum/minimum, point d'inflexion. Le premier segment complet comprend *b*, *c*, *d* et il a pour longueur *n2* + *n3*, le suivant comprend *d*, *e*, *f* et il a pour longueur *n4* + *n5*, etc. Le premier morceau (*a*, *b* de longueur *n1*) est incomplet avec seulement une inflexion ; le dernier morceau peut être incomplet aussi. Bien que les points d'inflexion *b*, *d*, *f* ... figurent chacun dans deux segments (un à gauche et un à droite), les pentes des deux segments restent indépendantes à ce point commun (c'est-à-dire que la dérivée première sera probablement discontinue). Quand *a*, *c*, *e*... sont alternativement maximum et minimum, les jointures des inflexions seront relativement douces ; pour des maxima successifs ou des minima successifs les inflexions seront en peigne.

Exemples

Voici un exemple simple de la routine GEN06. Il utilise le fichier *gen06.csd* [examples/gen06.csd]. Il crée une courbe allant de 0 à 1 puis à -1, avec un minimum, un maximum et un minimum à ces valeurs respectives. Les inflexions sont à .5 et 0 et sont relativement douces. Voici son graphe :



Graphe de la forme d'onde générée par GEN06.

Exemple 613. Un exemple simple de la routine GEN06.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen06.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kval table kndx, ifn, ixmode

  ; Generate a sine waveform, use our Table #1 value to
  ; vary its frequency by 100 Hz from its base frequency.
  ibasefreq = 440
  kfreq = kval * 100
  a1 oscil 20000, ibasefreq + kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a curve (using GEN06).
f 1 0 65 6 0 16 0.5 16 1 16 0 16 -1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN05, *GEN07* et *GEN08*

GEN07

GEN07 — Construit des fonctions à partir de morceaux de lignes droites.

Description

Construit des fonctions à partir de morceaux de lignes droites.

Syntaxe

```
f  #    date    taille   7   a   n1   b   n2   c  ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

a, b, c, etc. -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, . . .

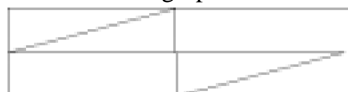
n1, n2, etc. -- longueur de segment (nombre de positions en mémoire), dans les p-champs de numéros pairs. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des formes d'onde discontinues (comme dans l'exemple ci-dessous). La somme *n1* + *n2* + ... sera normallement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées.

Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la (n + 1)ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

Exemples

Voici un exemple simple de la routine GEN07. Il utilise le fichier *gen07.csd* [examples/gen07.csd]. Il créera une période d'une onde en dent de scie dont la discontinuité se trouve au milieu de la fonction stockée. Voici le graphe :



Graphe de la forme d'onde générée par GEN07.

Exemple 614. Un exemple simple de la routine GEN07.

Voir les sections *Audio en Temps-Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen07.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the sine wave stored in Table #1.
  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a sawtooth wave (using GEN07).
f 1 0 256 7 0 128 1 0 -1 128 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN05, *GEN06* et *GEN08*

GEN08

GEN08 — Génère une courbe spline cubique par morceaux.

Description

Ce sous-programme génèrera une courbe spline cubique par morceaux, la plus lisse possible le long de tous les points spécifiés.

Syntaxe

```
f # date taille 8 a n1 b n2 c n3 d ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

a, b, c, etc. -- valeurs d'ordonnée de la fonction.

n1, n2, n3 ... -- longueur de chaque segment mesurée en valeurs mémorisées. Ne peuvent pas être nulles, mais peuvent être fractionnaires. Un segment particulier peut stocker ou non des valeurs ; les valeurs stockées seront générées à des points entiers à partir de début de la fonction. La somme *n1 + n2 + ...* sera normalement égale à *taille* pour les fonctions complètement spécifiées.

Note

- *GEN08* construit une table stockée à partir de morceaux d'une fonction polynomiale cubique. Chaque segment s'étend entre deux points spécifiés mais dépend aussi de leurs voisins de chaque côté. Les segments voisins coïncideront en valeur et en pente à leur point commun. (La pente commune est celle d'une parabole passant par ce point et ses deux voisins). La pente aux deux extrémités de la fonction est forcée à zéro (plate).

- *Conseil* : pour créer une discontinuité de pente ou de valeur dans la fonction stockée, disposer une série de points dans l'intervalle entre deux valeurs stockées ; faire de même pour une pente non nulle à l'une des extrémités.

Exemples

Voici un exemple simple de la routine GEN08. Il utilise le fichier *gen08.csd* [examples/gen08.csd]. Il créera une bosse lisse au milieu, légèrement négative des deux côtés, s'applatissant ensuite aux extrémités. Voici le graphe :

Graphe de la fonction générée par GEN08.

Exemple 615. Un exemple simple de la routine GEN08.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen08.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kval table kndx, ifn, ixmode

  ; Generate a sine waveform, use our Table #1 value to
  ; vary its frequency by 100 Hz from its base frequency.
  ibasefreq = 440
  kfreq = kval * 100
  a1 oscil 20000, ibasefreq + kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a curve with a smooth hump (using GEN08).
f 1 0 65 8 0 16 0 16 1 16 0 16 0
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN05, *GEN06* et *GEN07*

GEN09

GEN09 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

Description

Ce sous-programmme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 3 p-champs avec *GEN09*.

Syntaxe

```
f # date taille 9 pna ampa phsa pnb ampb phsb ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

pna, pnb, etc. -- numéro de partiel (par rapport à un fondamental qui occuperait *taille* positions par période) des sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

ampa, ampb, etc. -- amplitude des partiels *pna, pnb*, etc. Ce sont des amplitudes relatives, car la forme d'onde complexe peut être reproportionnée à posteriori. On peut utiliser des valeurs négatives pour signifier une opposition de phase (180 degrés).

phsa, phsb, etc. -- phase initiale des partiels *pna, pnb,* etc., exprimée en degrés (0-360).

Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproportionnée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

Exemples

Voici un exemple simple de la routine GEN09. Il utilise le fichier *gen09.csd* [examples/gen09.csd]. Il génèrera une onde cosinus, une onde sinusoïdale avec une phase initiale de 90 degrés. Voici son graphe :

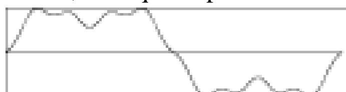
Graphe de la forme d'onde générée par GEN09.

Exemple 616. Un exemple simple de la routine GEN09.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen09.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the waveform stored in Table #1.
  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a cosine wave (using GEN09).
; This is a sine wave with an initial phase of 90 degrees.
f 1 0 16384 9 1 1 90

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de la routine GEN09. Il utilise le fichier *gen09square.csd* [examples/gen09square.csd]. Il combine les partiels 1, 3 et 9 avec les intensités relatives qu'ils ont dans une onde carrée, sauf que le partiel 9 est inversé. La fonction sera normalisée. Voici le graphe :



Graphe de la forme d'onde générée par GEN09.

Exemple 617. Une onde carrée générée par la routine GEN09.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc     ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o gen09square.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the waveform stored in Table #1.
  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: an approximation of a square wave (using GEN09).
f 1 0 16384 9 1 3 0 3 1 0 9 0.3333 180

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN10, *GEN19*

Crédits

L'exemple simple a été écrit par Kevin Conder.

GEN10

GEN10 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 1 p-champ avec *GEN10*.

Syntaxe

```
f # date taille 10 amp1 amp2 amp3 amp4 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

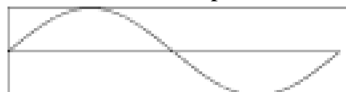
amp1, amp2, amp3, etc. -- amplitudes relatives des partiels harmoniques fixes de numéro 1, 2, 3, etc., commençant en p5. Les partiels non désirés recevront une amplitude nulle.

Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproportionnée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

Exemples

Voici un exemple de la routine GEN10. Il utilise le fichier *gen10.csd* [examples/gen10.csd]. Il génèrera une onde sinus simple. Voici son graphe :



Graphe de la forme d'onde générée par GEN10.

Exemple 618. Un exemple de la routine GEN10.

Voir les sections *Audio en Temps-Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen10.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the sine wave stored in Table #1.
  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple sine wave (using GEN10).
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN09, *GEN11* et *GEN19*.

Crédits

Exemple écrit par Kevin Conder

GEN11

GEN11 — Génère un ensemble additif de partiels cosinus.

Description

Ce sous-programme génère un ensemble additif de partiels cosinus, à la manière des générateurs de Csound *buzz* et *gbuzz*.

Syntaxe

```
f # date taille 11 nh [lh] [r]
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

nh -- nombre d'harmoniques demandés. Doit être positif.

lh(optional) -- harmonique présent le plus bas. Peut être positif, nul ou négatif. L'ensemble d'harmoniques peut démarrer à n'importe quel numéro d'harmonique et progresse vers le haut ; si *lh* est négatif, tous les harmoniques en dessous de zéro se réfléchiront autour de zéro pour produire des harmoniques positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux harmoniques positifs de l'ensemble. La valeur par défaut est 1.

r(facultatif) -- multiplicateur dans une série de coefficients d'amplitude. C'est une séries de puissances : si le *lh* ème harmonique a un coefficient d'amplitude de A le (*lh* + *n*)ème harmonique aura un coefficient de $A * r^n$, c'est-à-dire que les valeurs d'amplitudes suivent une courbe exponentielle. *r* peut être positif, nul ou négatif, et n'est pas restreint à des entiers. La valeur par défaut est 1.

Note

- Ce sous-programme est une version invariante dans le temps des générateurs de Csound *buzz* et *gbuzz*, et il est similairement utile comme source sonore complexe pour la synthèse soustractive. Si *lh* et *r* sont utilisés, il agit comme *gbuzz* ; si les deux sont absents ou égaux à 1, il se réduit au générateur plus simple *buzz* (c'est-à-dire *nh* harmoniques d'amplitude égale commençant avec le fondamental).
- Lire la forme d'onde stockée avec un oscillateur est plus efficace que d'utiliser les unités dynamiques buzz. Cependant, le contenu spectral est invariant et il faut faire attention à ce que les harmoniques les plus hauts ne dépassent pas la fréquence de Nyquist pour éviter les repliements.

Exemples

Voici un exemple simple de la routine GEN11. Il utilise le fichier *gen11.csd* [examples/gen11.csd]. Il génèrera une onde cosinus simple. Voici son graphe :

Graphe de la forme d'onde générée par GEN11.

Exemple 619. Un exemple simple de la routine GEN11.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen11.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the cosine wave stored in Table #1.
  a1 oscil kamp, kcps, ifn
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a simple cosine wave (using GEN11).
f 1 0 16384 11 1 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN10

Crédits

Exemple écrit par Kevin Conder

GEN12

GEN12 — Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.

Description

Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0, adaptée pour la MF modulée en amplitude.

Syntaxe

```
f # date taille 12 intx
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

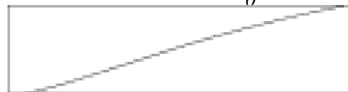
intx -- spécifie l'intervalle des *x* [0 à +*intx*] sur lequel la fonction est définie.

Note

- Ce sous-programme calcule le logarithme naturel d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0 (habituellement écrite comme I_0), sur l'intervalle des x demandé. Cet appel devrait désactiver la normalisation.
- Cette fonction est utile comme facteur d'échelle d'amplitude dans la MF à période synchrone modulée en amplitude. (Voir Palamin & Palamin, *J. Audio Eng. Soc., 36/9*, Sept. 1988, pp.671-684.) L'algorithme est intéressant car il permet de rendre le spectre de MF, habituellement symétrique, assymétrique autour d'une fréquence autre que la porteuse, et il est ainsi utile pour placer des formants. En utilisant un index de lecture dans la table de $I(r - 1/r)$, où I est l'index de modulation et r est un paramètre exponentiel affectant l'importance des partiels, l'algorithme Palamin se montre relativement efficace, ne demandant que des oscil, des lecture de table, et un appel d'*exp*.

Exemples

Voici un exemple simple de la routine GEN12. Il utilise le fichier *gen12.csd* [examples/gen12.csd]. Il génère la fonction $ln(I_0(x))$ de 0 à 20. Voici son graphe :



Graphe de la forme d'onde générée par GEN12.

Exemple 620. Un exemple simple de la routine GEN12.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen12.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kamp tablei kndx, ifn, ixmode

  ; Create a sine wave, use the Table #1 values to control
  ; the amplitude. This creates a sound with a long attack.
  a1 oscil kamp*30000, 440, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a modified Bessel function (using GEN12).
f 1 0 2049 12 20
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Exemple écrit par Kevin Conder

GEN13

GEN13 — Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.

Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

Syntaxe

```
f # date taille 13 xint xamp h0 h1 h2 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

xint -- fournit les valeurs gauches et droites [*-xint, +xint*] de l'intervalle des x sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en p5 est ainsi étendue en une paire négative-positive p5, p6 avant l'appel de *GEN03*. La valeur normale est 1.

xamp -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

h0, h1, h2, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

xamp * int(taille/2)/xint

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.

Note

GEN13 est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0, h1, h2,* ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

Exemples

Voici un exemple simple de la routine GEN13. Il utilise le fichier *gen13.csd* [examples/gen13.csd]. Il crée une fonction qui, lors du waveshaping, séparera une sinus en 3 harmoniques impairs d'importance relative 5:3:1. Voici son graphe :

Graphe de la forme d'onde générée par GEN13.

Exemple 621. Un exemple simple de la routine GEN13.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen13.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kval table kndx, ifn, ixmode

  ; Generate a sine waveform, use our Table #1 value to
  ; vary its frequency by 100 Hz from its base frequency.
  ibasefreq = 440
  kfreq = kval * 100
  a1 oscil 20000, ibasefreq + kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN13).
f 1 0 1025 13 1 1 0 5 0 3 0 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN03, *GEN14* et *GEN15*.

GEN14

GEN14 — Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.

Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

Syntaxe

```
f # date taille 14 xint xamp h0 h1 h2 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

xint -- fournit les valeurs gauches et droites [*-xint, +xint*] de l'intervalle des x sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en p5 est ainsi étendue en une paire négative-positive p5, p6 avant l'appel de *GEN03*. La valeur normale est 1.

xamp -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

h0, h1, h2, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

xamp * int(taille/2)/xint

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.

Note

- *GEN13* est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0, h1, h2,* ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

- *GEN14* stocke un polynôme dont les coefficients dérivent de polynômes de Tchebychev de seconde espèce.

Exemples

Voici un exemple simple de la routine GEN14. Il utilise le fichier *gen14.csd* [examples/gen14.csd]. Il crée une fonction qui, lors du waveshaping, séparera une sinus en 3 harmoniques impairs d'importance relative 5:3:1. Voici son graphe :

Graphe de la forme d'onde générée par GEN14.

Exemple 622. Un exemple simple de la routine GEN14.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen14.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kval table kndx, ifn, ixmode

  ; Generate a sine waveform, use our Table #1 value to
  ; vary its frequency by 100 Hz from its base frequency.
  ibasefreq = 440
  kfreq = kval * 100
  a1 oscil 20000, ibasefreq + kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN14).
f 1 0 1025 14 1 1 0 5 0 3 0 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
```

```
e

</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN03, *GEN13* et *GEN15*.

Crédits

Exemple écrit par Kevin Conder

GEN15

GEN15 — Crée deux tables de fonctions polynomiales mémorisées.

Description

Ce sous-programme crée deux tables de fonctions polynomiales mémorisées, appropriées pour une utilisation en quadrature de phase.

Syntaxe

```
f # date taille 15 xint xamp h0 phs0 h1 phs1 h2 phs2 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

xint -- fournit les valeurs gauches et droites [*-xint, +xint*] de l'intervalle des x sur lequel le polynôme doit être évalué. Ce sous-porgramme appellera éventuellement *GEN03* pour évaluer les deux fonctions ; la valeur en p5 est alors étendue en une paire négative-positive p5, p6 avant l'appel de *GEN03*. La valeur normale est 1.

xamp -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

h0, h1, h2, ..., hn -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

xamp * int(taille/2)/xint

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.

phs0, phs1, ... -- phase en degrés des harmoniques désirés *h0, h1, ...* lorsque les deux fonctions de *GEN15* sont utilisées en quadrature de phase.

Note

GEN15 crée deux tables de même taille, étiquetées *f* # et *f* # + 1. La table # contiendra une fonction de Tchebychev de première espèce, évaluée par *GEN03* avec des harmoniques d'amplitude *h0cos(phs0), h1cos(phs1), ...*. Table # + 1 contiendra une fonction de Tchebychev de deuxième espèce, évaluée par *GEN14* avec les harmoniques *h1sin(phs1), h2sin(phs2),...* (noter le déplacement harmonique). Les deux tables peuvent être utilisées en conjonction dans un réseau de waveshaping qui exploite la quadrature de phase.

Voir Aussi

GEN03, *GEN13* et *GEN14*.

GEN16

GEN16 — Crée une table depuis une valeur initiale jusqu'à une valeur terminale.

Description

Crée une table depuis la valeur *deb* jusqu'à la valeur *fin* en *dur* pas.

Syntaxe

```
f # date taille 16 deb dur type fin
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

deb -- valeur de départ

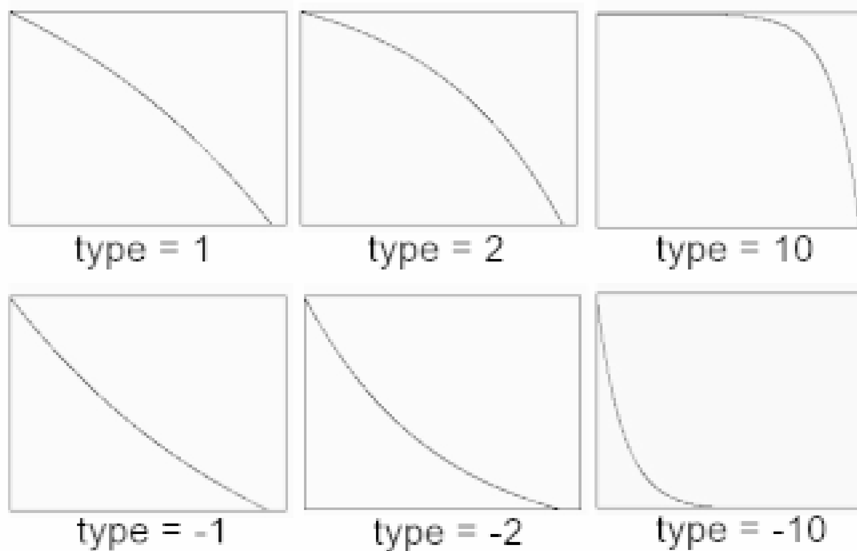
dur -- nombre de segments

type -- si 0, une ligne droite est produite. Si différent de zéro, alors *GEN16* crée la courbe suivante sur *dur* pas :

deb + (fin - deb) * (1 - exp(i*type/(dur-1))) / (1 - exp(type))

fin -- valeur après *dur* segments

Voici quelques exemples de courbes générées pour différentes valeurs de *type* :



Tables générées par GEN16 pour différentes valeurs de *type*.

Note

Si *type* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *type* < 0, la courbe monte rapidement (convexe) ou décroit rapidement (concave). Voir aussi *transeg*.

Exemple 623. Un exemple simple de la routine GEN16.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen16.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

instr 1
  kcps init 1/p3
  kndx phasor kcps

  ifn = p4
  ixmode = 1
  kval table kndx, ifn, ixmode

  ibasefreq = 440
  kfreq = kval * ibasefreq
  a1 oscil 20000, ibasefreq + kfreq, 1
  out a1
endin


</CsInstruments>
<CsScore>


f 1 0 16384 10 1

f 2 0 1024 16 1 1024 1 0
f 3 0 1024 16 1 1024 2 0
f 4 0 1024 16 1 1024 10 0
f 5 0 1024 16 1 1024 -1 0
f 6 0 1024 16 1 1024 -2 0
f 7 0 1024 16 1 1024 -10 0

i 1 0 2 2
i 1 + . 3
i 1 + . 4
i 1 + . 5
i 1 + . 6
i 1 + . 7

e


</CsScore>
</CsoundSynthesizer>
```

Crédits

Auteur : John ffitch

University of Bath, Codemist. Ltd.
Bath, UK
Octobre 2000

Nouveau dans la version 4.09 de Csound

GEN17

GEN17 — Crée une fonction en escalier à partir des paires x-y données.

Description

Ce sous-programme crée une fonction en escalier à partir des paires x-y données.

Syntaxe

```
f # date taille 17 x1 a x2 b x3 c  ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

x1, x2, x3, etc. -- valeurs d'abscisse x, en ordre ascendant, commençant par 0.

a, b, c, etc. -- valeurs y à ces valeurs d'abscisse x, maintenues jusqu'à la valeur d'abscisse x suivante.

Note

Ce sous-programme crée une fonction en escalier de paires x-y dont les valeurs y sont maintenues vers la droite. La valeur de y la plus à droite est ensuite maintenue jusqu'à la fin de la table. Cette fonction est utile pour mettre en correspondance un ensemble de données avec un autre, tel que des numéros de notes MIDI avec des numéros de ftables de sons échantillonés. (voir *loscil*).

Exemples

```
f  1  0  128  -17   0  1   12  2   24  3   36  4   48  5  60  6   72  7   84  8
```

Ceci décrit une fonction en escalier avec huit niveaux croissants successifs, chacun occupant 12 positions sauf pour le dernier qui étend sa valeur jusqu'à la fin de la table. La normalisation est empêchée. En indexant cette table avec un numéro de note MIDI, on retrouvera une valeur différente pour chaque octave jusqu'à la huitième, au-delà de laquelle la valeur retournée restera la même.

Voir Aussi

GEN02

GEN18

GEN18 — Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.

Description

Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes. Chaque forme d'onde utilisée nécessite 4 p-champs et peut se chevaucher avec les autres formes d'onde.

Syntaxe

```
f # date taille 18 fna ampa debuta fina fnb ampb debutb finb ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance-de-2 plus 1 (voir l'instruction f).

fna, fnb, etc. -- numéros des tables pré-existantes à écrire dans la table.

ampa, ampb, etc. -- amplitude des formes d'onde. Ces amplitudes sont relatives, car la forme d'onde composée pourra être post-normalisée. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

debuta, debutb, etc. -- où commencer à écrire fn dans la table.

fina, finb, etc. -- où terminer l'écriture de fn dans la table.

Exemples

```
f 1   0   4096   10   1
f 2   0   1025   18   1   1   0   512   1   1   513   1025
```

f2 consiste en deux copies de f1 écrites dans les positions 0-512 et 513-1025.

Noms anciennement utilisés

GEN18 était appelé *GEN22* dans la version 4.18. Le nom fut changé à cause d'un conflit avec DirectCsound.

Crédits

Auteur : William « Pete » Moss
University of Texas at Austin
Austin, Texas USA
Janvier 2002

Nouveau dans la version 4.18, changé dans la version 4.19

GEN19

GEN19 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 4 p-champs dans *GEN19*.

Syntaxe

```
f # date taille  19  pna  ampa  phsa  dcoa  pnb  ampb  phsb  dcob  ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction f).

pna, pnb, etc. -- numéro de partiel (relativement à un fondamental qui occuperait *taille* positions par période) de sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

ampa, ampb, etc. -- amplitude des partiels *pna, pnb*, etc. Ces amplitudes sont relatives, car la forme d'onde composée peut être normalisée plus tard. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

phsa, phsb, etc. -- phase initiale des partiels *pna, pnb,* etc., exprimée en degrés.

dcoa, dcob, etc. -- Décalage CC (Composante Continue) des partiels *pna, pnb*, etc. Il est appliqué *après* l'amplitude, c'est-à-dire qu'une valeur de 2 montera une sinus d'amplitude 2 de l'intervalle [-2,2] à l'intervalle [0,4] (avant la normalisation finale).

Note

- Ces sous-programmes génèrent des fonctions stockées comme sommes de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas l'onde composée, une fois évaluée, est ensuite normalisée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

Exemples

Voici un exemple simple de la routine GEN19. Il utilise le fichier *gen19.csd* [examples/gen19.csd]. Il génèrera une jolie courbe en cloche, voici son graphe :

Graphe de la forme d'onde générée par GEN19.

Exemple 624. Un exemple simple de la routine GEN19.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac           -iadc    ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen19.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Create an index over the length of our entire note.
  kcps init 1/p3
  kndx phasor kcps

  ; Read Table #1 with our index.
  ifn = 1
  ixmode = 1
  kval table kndx, ifn, ixmode

  ; Generate a sine waveform, use our Table #1 value to
  ; vary its frequency by 100 Hz from its base frequency.
  ibasefreq = 440
  kfreq = kval * 100
  a1 oscil 20000, ibasefreq + kfreq, 2
  out a1
endin


</CsInstruments>
<CsScore>

; Table #1: a bell curve (using GEN19).
f 1 0 16384 -19 1 1 260 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
e


</CsScore>
</CsoundSynthesizer>
```

Voir Aussi

GEN09 et *GEN10*

Crédits

Exemple écrit par Kevin Conder

GEN20

GEN20 — Génère les fonctions de différentes fenêtres.

Description

Ce sous-programme génère les fonctions de différentes fenêtres. Ces fenêtres sont utilisées habituellement pour l'analyse spectrale ou pour des enveloppes de grain.

Syntaxe

```
f # date taille 20 fenêtre max [opt]
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 (+ 1).

fenêtre -- Type de la fenêtre à générer :

- 1 = Hamming
- 2 = Hanning
- 3 = Bartlett (triangle)
- 4 = Blackman (3-termes)
- 5 = Blackman - Harris (4-termes)
- 6 = Gaussienne
- 7 = Kaiser
- 8 = Rectangle
- 9 = Sync

max -- Pour p4 négatif ce sera la valeur absolue au pic de la fenêtre. Si p4 est positif ou si p4 est négatif et p6 est absent la table sera post-normalisée à une valeur maximale de 1.

opt -- Argument facultatif nécessaire pour la fenêtre gaussienne et pour la fenêtre de Kaiser.

Exemples

```
f       1       0       1024    20      5
```

Crée une fonction qui contient une fenêtre de Blackman - Harris à 4 termes avec une valeur maximale de 1.

```
f       1       0       1024    -20     2       456
```

Crée une fonction qui contient une fenêtre de Hanning avec une valeur maximale de 456.

```
f       1       0       1024    -20     1
```

Crée une fonction qui contient une fenêtre de Hamming avec une valeur maximale de 1.

```
f       1       0       1024    20      7       1       2
```

Crée une fonction qui contient une fenêtre de Kaiser avec une valeur maximale de 1. L'argument supplémentaire spécifie comment la fenêtre est "ouverte", par exemple une valeur de 0 donne une fenêtre rectangulaire et une valeur de 10 donne une fenêtre semblable à une fenêtre de Hamming.

```
f       1       0       1024    20      6       1       2
```

Crée une fonction qui contient une fenêtre gaussienne avec une valeur maximale de 1. L'argument supplémentaire spécifie la largeur de la fenêtre, comme l'écart type de la courbe ; dans cette exemple l'écart type vaut 2. La valeur par défaut est 1.

Pour les graphes, voir les *Fonctions Fenêtre*

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK

Nouveau dans la version 3.2 de Csound

L'argument facultatif de la gaussienne a été ajouté dans la version 5.10

GEN21

GEN21 — Génère les tables de différentes distributions aléatoires.

Description

Génère les tables de différentes distributions aléatoires. (Voir aussi *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand* et *weibull*)

Syntaxe

```
f # date taille 21 type niveau [arg1  [arg2]]
```

Initialisation

date et *taille* sont les arguments habituels des fonctions GEN. *niveau* définit l'amplitude. Noter que GEN21 n'effectue pas d'auto-normalisation comme le font la plupart des autres fonctions GEN. *type* définit la distribution à utiliser :

- 1 = Uniforme (seulement des nombres positifs)
- 2 = Linéaire (seulement des nombres positifs)
- 3 = Triangulaire (nombres positifs et négatifs)
- 4 = Exponentielle (seulement des nombres positifs)
- 5 = Biexponentielle (nombres positifs et négatifs)
- 6 = Gaussienne (nombres positifs et négatifs)
- 7 = Cauchy (nombres positifs et négatifs)
- 8 = Cauchy Positive (seulement des nombres positifs)
- 9 = Beta (seulement des nombres positifs)
- 10 = Weibull (seulement des nombres positifs)
- 11 = Poisson (seulement des nombres positifs)

De tous ces cas seulement le 9 (Beta) et le 10 (Weibull) ont besoin d'arguments supplémentaires. Beta nécessite deux arguments et Weibull un.

Exemples

```
f1 0 1024 21 1        ; Uniforme (bruit blanc)
f1 0 1024 21 6        ; Gaussienne
f1 0 1024 21 9 1 1 2 ; Beta (noter que le niveau précède les arguments)
f1 0 1024 21 10 1 2  ; Weibull
```

Toutes les additions ci-dessus furent conçus par l'auteur entre mai et décembre 1994, sous la supervision du Dr Richard Boulanger.

Crédits

Auteur : Paris Smaragdis
MIT, Cambridge
1995

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK

Nouveau dans la version 3.2 de Csound

GEN22

GEN22 — Obsolète.

Description

Obsolète depuis la version 4.19. Utiliser plutôt la routine *GEN18*.

GEN23

GEN23 — Lit des valeurs numériques à partir d'un fichier texte.

Description

Ce sous-programme lit des valeurs numériques à partir d'un fichier ASCII.

Syntaxe

```
f # date taille -23 "nomfichier.txt"
```

Initialisation

"nomfichier.txt" -- les valeurs numériques contenues dans "nomfichier.txt" (qui indique le nom de chemin complet du fichier de caractères à lire) peuvent être séparées par des espaces, des tabulations, des caractères de passage à la ligne ou des virgules. De plus, on peut utiliser comme commentaires des mots qui contiennent des caractères non numériques car ils sont ignorés.

taille -- nombre de points dans la table. Doit être une puissance de 2, une puissance de 2 + 1, ou zéro. Si *taille* = 0, la taille de la table est déterminée par le nombre de valeurs numériques dans *nomfichier.txt*. (Nouveau dans la version 3.57 de Csound)

Note

Tous les caractères suivant un ';' (commentaire) sont ignorés jusqu'à la ligne suivante (les nombres aussi).

Crédits

Auteur : Gabriel Maldonado
Italie
Février 1998

Nouveau dans la version 3.47 de Csound

GEN24

GEN24 — Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

Description

Ce sous-programme lit les valeurs numériques d'une table de fonction déjà allouée et les reproportionne selon les valeurs *min* et *max* données par l'utilisateur.

Syntaxe

```
f # date taille -24 ftable min max
```

Initialisation

#, date, taille -- les paramètres GEN habituels. Voir l'instruction f.

ftable -- ftable doit être une table déjà allouée avec la même taille que cette fonction.

min, max -- l'intervalle de recadrage.

Note

Ce GEN est utile, par exemple, pour éliminer le décalage du début dans les morceaux d'exponentielle permettant d'avoir une vrai origine à zéro.

Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16 de Csound

GEN25

GEN25 — Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

Description

Ces sous-programmes sont utilisés pour construire des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

Syntaxe

```
f # date taille 25 x1 y1 x2 y2 x3  ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

x1, x2, x3, etc. -- positions dans la table auxquelles la valeur y suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

y1, y2, y3, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur x précédente. Elles doivent être non nulles et toutes du même signe.

Note

Si p4 est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un p4 négatif empêchera cette opération.

Voir Aussi

Instruction f, *GEN27*

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK

Nouveau dans la version 3.49 de Csound

GEN27

GEN27 — Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

Description

Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

Syntaxe

```
f # date taille 27 x1 y1 x2 y2 x3 ...
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1. (voir l'*instruction f*).

x1, x2, x3, etc. -- positions dans la table auxquelles la valeur y suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

y1, y2, y3, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur x précédente.

Note

Si p4 est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un p4 négatif empêchera cette opération.

Exemples

```
f 1 0 257 27 0 0 100 1 200 -1 256 0
```

Décrit une fonction qui commence à 0, monte jusqu'à 1 à la 100ème position de la table, descend à -1, à la 200ème position, et revient à 0 à la fin de la table. L'interpolation est linéaire.

Voir Aussi

Instruction f, *GEN25*

Crédits

Auteur : John ffitch
University of Bath/Codemist Ltd.
Bath, UK

Nouveau dans la version 3.49 de Csound

GEN28

GEN28 — Lit un fichier texte qui contient une trajectoire paramétrée par le temps.

Description

Ce générateur de fonction lit un fichier texte qui contient des ensembles de trois valeurs représentant des coordonnées xy et un paramètre temporel indiquant quand placer le signal à cette position, permettant à l'utilisateur de définir une trajectoire paramétrée par le temps. Le format du fichier est de la forme :

```
temps1   X1   Y1
temps2   X2   Y2
temps3   X3   Y3
```

La configuration des coordonnées xy dans l'espace place le signal de la manière suivante :

- a1 est -1, 1
- a2 est 1, 1
- a3 est -1, -1
- a4 est 1, -1

Cela suppose des haut-parleurs disposés avec a1 en avant gauche, a2 en avant droite, a3 en arrière gauche, a4 en arrière droite. Les valeurs supérieures à 1 provoqueront une atténuation des sons comme s'ils étaient distants. *GEN28* crée les valeurs avec une résolution de 10 millisecondes.

Syntaxe

```
f # date taille 28 codfic
```

Initialisation

taille -- nombre de points dans la table. Doit être 0. *GEN28* prend une taille de 0 et alloue la mémoire automatiquement.

codfic -- chaîne de caractères dénotant le nom du fichier source. Une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier nommé est cherché dans le répertoire courant.

Exemples

```
f1 0 0 28 "move"
```

Le fichier "move" ressemblera à ceci :

```
0     -1     1
1      1     1
2      4     4
2.1   -4    -4
3     10   -10
5    -40     0
```

Puisque *GEN28* crée les valeurs avec une résolution de 10 millisecondes, il y aura 500 valeurs créées en interpolant entre X1 et X2, X2 et X3, etc., et entre Y1 et Y2, Y2 et Y3, etc., sur le nombre approprié de valeurs qui sont stockées dans la table de fonction. Le son démarrera à l'avant gauche, il bougera pendant 1 seconde vers l'avant droite, durant la seconde suivante il s'éloignera mais toujours à l'avant droite, ensuite il bougera vers l'arrière gauche en seulement 1/10 de seconde, un peu éloigné. Enfin, pendant les 0,9 secondes restantes le son bougera vers l'arrière droite, modérément éloigné, et il viendra s'arrêter entre les deux canaux gauche (plein ouest !), assez éloigné.

Crédits

Auteur : Richard Karpen
Seattle, Wash
1998

Nouveau dans la version 3.48 de Csound

GEN30

GEN30 — Génère des partiels harmoniques en analysant une table existante.

Description

Extrait un sous-ensemble de la série harmonique d'une forme d'onde existante.

Syntaxe

```
f # date taille 30 src minh maxh [ref_sr] [interp]
```

Exécution

src -- ftable source

minh -- numéro de l'harmonique le plus bas

maxh -- numéro de l'harmonique le plus haut

ref_sr (facultatif) -- maxh est pondéré par (*sr* / *ref_sr*). La valeur par défaut de *ref_sr* est *sr*. Si *ref_sr* est nul ou négatif, il est ignoré.

interp (facultatif) -- si différent de zéro, permet de changer l'amplitude des harmoniques le plus bas et le plus haut en fonction de la partie fractionnaire de *minh* et *maxh*. Par exemple, si *maxh* vaut 11.3 alors le 12ème harmonique est ajouté avec une amplitude de 0.3. Ce paramètre vaut zéro par défaut.

GEN30 ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que *GEN30* utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. *GEN32* permet l'utilisation de l'interpolation linéaire pour le rééchantillonage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

GEN31

GEN31 — Mélange n'importe quelle forme d'onde définie dans une table existante.

Description

Cette routine est semblabe à GEN09, mais permet le mélange de n'importe quelle forme d'onde définie dans une table existante.

Syntaxe

```
f # date taille  31  src  pna ampa phsa  pnb ampb phsb  ...
```

Exécution

src -- numéro de la table source

pna, pnb, ... -- numéro de partiel, doit être un entier positif

ampa, ampb, ... -- échelle d'amplitude

phsa, phsb, ... -- phase initiale (0 à 1)

GEN31 ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que *GEN31* utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. *GEN32* permet l'utilisation de l'interpolation linéaire pour le rééchantillonage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

GEN32

GEN32 — Mélange n'importe quelle forme d'onde, rééchantillonnée soit par TFR soit par interpolation linéaire.

Description

Cette routine est semblable à *GEN31*, mais elle permet la spécification d'une table source pour chaque partiel. Les tables peuvent être rééchantilonnées soit par TFR soit par interpolation linéaire.

Syntaxe

```
f # date taille  32  srca pna ampa phsa  srcb pnb ampb phsb  ...
```

Exécution

srca, srcb -- numéro de table source. Une valeur négative peut être utilisée pour lire une table avec interpolation linéaire (par défaut, la forme d'onde source est transposée et déphasée par TFR) ; c'est moins précis, mais plus rapide, et cela permet des numéros de partiels non entiers et négatifs.

pna, pnb, ... -- numéro de partiel, doit être un entier positif si le numéro de la table source est positif (c'est-à-dire rééchantillonage par TFR).

ampa, ampb, ... -- échelle d'amplitude

phsa, phsb, ... -- phase initiale (0 à 1)

Exemples

```
itmp    ftgen 1, 0, 16384, 7, 1, 16384, -1       ; dent de scie
itmp    ftgen 2, 0, 8192, 10, 1                  ; sinus
; mélange les tables
itmp    ftgen 5, 0, 4096, -32, -2, 1.5, 1.0, 0.25, 1, 2, 0.5, 0,        \
                                1, 3, -0.25, 0.5
; fenêtre
itmp    ftgen 6, 0, 16384, 20, 3, 1
; génère des formes d'onde à bande limitée
inote   =  0
loop0:
icps    =  440 * exp(log(2) * (inote - 69) / 12)        ; une table pour
inumh   =  sr / (2 * icps)                              ; chaque numéro de note MIDI
ift     =  int(inote + 256.5)
itmp    ftgen ift, 0, 4096, -30, 5, 1, inumh
inote   =  inote + 1
        if (inote < 127.5) igoto loop0

        instr 1

kcps    expon 20, p3, 16000
kft     =  int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft     =  (kft > 383 ? 383 : kft)

a1      phasor kcps
a1      tableikt a1, kft, 1, 0, 1

        out a1 * 10000

        endin
        instr 2
```

```
kcps    expon 20, p3, 16000
kft     =  int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft     =  (kft > 383 ? 383 : kft)

kgdur   limit 10 / kcps, 0.1, 1
a1      grain2 kcps, 0.02, kgdur, 30, kft, 6, -0.5

        out a1 * 2000

        endin

--------------
partition :
--------------

t 0 60
i 1 0 10
i 2 12 10
e
```

Crédits

Auteur : Rasmus Ekman

Programmeur : Istvan Varga

Nouveau dans la version 4.17

GEN33

GEN33 — Génère des formes d'onde complexes en mélangeant des sinus.

Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

Syntaxe

```
f # date taille 33 src nh ech [fmode]
```

Initialisation

taille -- nombre de points dans la table. Doit être une puissance de deux et au moins 4.

src -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

ampa, pna, phsa, ampb, pnb, phsb, ...

les paramètres sont :

- *ampa, ampb,* etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna, pnb,* etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu. Avec *GEN33*, le numéro de partiel est arrondi à l'entier le plus proche.
- *phsa, phsb,* etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins 3 * *nh*. Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

nh -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

ech -- échelle d'amplitude.

fmode (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonage est supposé être *fmode* si celui-ci est

positif, ou -(*sr* * *fmode*) si une valeur négative est spécifiée.

Exemples

```
; partiels 1, 4, 7, 10, 13, 16, etc. avec une fréquence de base de 400 Hz

ibsfrq  =  400
; nombre de partiels estimé
inumh   =  int(1.5 + sr * 0.5 / (3 * ibsfrq))
; longueur de la table source
isrcln  =  int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2))))
; crée une table source vide
itmp    ftgen 1, 0, isrcln, -2, 0
ifpos   =  0
ifrq    =  ibsfrq
inumh   =  0
l1:
        tableiw ibsfrq / ifrq, ifpos, 1         ; amplitude
        tableiw ifrq, ifpos + 1, 1              ; fréquence
        tableiw 0, ifpos + 2, 1                 ; phase
ifpos   =  ifpos + 3
ifrq    =  ifrq + ibsfrq * 3
inumh   =  inumh + 1
        if (ifrq < (sr * 0.5)) igoto l1

; stocke la sortie dans la ftable 2 (taille = 262144)

itmp    ftgen 2, 0, 262144, -33, 1, inumh, 1, -1
```

Voir Aussi

GEN09, *GEN34*

Crédits

Programmeur : Istvan Varga
Mars 2002

Nouveau dans la version 4.19

GEN34

GEN34 — Génère des formes d'onde complexes en mélangeant des sinus.

Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

Syntaxe

```
f # date taille 34 src nh ech [fmode]
```

Initialisation

size -- nombre de points dans la table. Doit être une puissance de deux ou une puissance-de-deux plus 1.

src -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

ampa, pna, phsa, ampb, pnb, phsb, ...

les paramètres sont :

- *ampa, ampb,* etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna, pnb,* etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu.
- *phsa, phsb,* etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins 3 * *nh*. Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

nh -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

ech -- échelle d'amplitude.

fmode (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonage est supposé être *fmode* si celui-ci est

positif, ou -(*sr* * *fmode*) si une valeur négative est spécifiée.

Exemples

```
; partiels 1, 4, 7, 10, 13, 16, etc. avec une fréquence de base de 400 Hz

ibsfrq  =  400
; nombre de partiels estimé
inumh   =  int(1.5 + sr * 0.5 / (3 * ibsfrq))
; longueur de la table source
isrcln  =  int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2))))
; crée une table source vide
itmp    ftgen 1, 0, isrcln, -2, 0
ifpos   =  0
ifrq    =  ibsfrq
inumh   =  0
l1:
        tableiw ibsfrq / ifrq, ifpos, 1         ; amplitude
        tableiw ifrq, ifpos + 1, 1              ; fréquence
        tableiw 0, ifpos + 2, 1                 ; phase
ifpos   =  ifpos + 3
ifrq    =  ifrq + ibsfrq * 3
inumh   =  inumh + 1
        if (ifrq < (sr * 0.5)) igoto l1

; stocke la sortie dans la ftable 2 (taille = 262144)

itmp    ftgen 2, 0, 262144, -34, 1, inumh, 1, -1
```

Voir Aussi

GEN09, *GEN33*

Crédits

Programmeur : Istvan Varga
Mars 2002

Nouveau dans la version 4.19

GEN40

GEN40 — Génère une distribution aléatoire à partir d'un histogramme.

Description

Génère une distribution aléatoire continue en partant de la forme d'un histogramme défini par l'utilisateur.

Syntaxe

```
f # date taille 40 tblforme
```

Exécution

La forme de l'histogramme doit être stockée dans une table préalablement définie, en fait, *tblforme* doit contenir le numéro de cette table.

La forme de l'histogramme peut être générée avec n'importe quelle GEN routine. Comme il n'y a pas d'interpolation lorque GEN40 opère la traduction, il est suggéré de donner à la table contenant la forme de l'histogramme une taille raisonnablement grande, afin d'obtenir une meilleure précision (cependant, cette dernière table peut être détruite après le traitement pour récupérer de la mémoire).

Ce sous-programme est prévu pour être utilisé avec l'opcode *cuserrnd* (voir *cuserrnd* pour plus d'information).

Crédits

Auteur : Gabriel Maldonado

GEN41

GEN41 — Génère une liste aléatoire de paires numériques.

Description

Génère une fonction de distribution aléatoire discrète en donnant une liste de paires numériques.

Syntaxe

```
f # date taille -41 valeur1 prob1 valeur2 prob2 valeur3 prob3 ... valeurN probN
```

Exécution

Le premier nombre de chaque paire est une valeur, et le second est la probabilité que cette valeur soit choisie par un algorithme aléatoire. Même si n'importe quel nombre peut être assigné à l'élément probabilité de chaque paire, il vaut mieux lui donner une valeur en pourcentage, afin de rendre les choses plus claires pour l'utilisateur.

Ce sous-programme est prévu pour être utilisé avec les opcodes *duserrnd* et *urd* (voir *duserrnd* pour plus d'information).

Crédits

Auteur : Gabriel Maldonado

GEN42

GEN42 — Génère une distribution aléatoire d'intervalles discrets de valeurs.

Description

Génère une fonction de distribution aléatoire d'intervalles discrets de valeurs en donnant une liste de groupes de trois nombres.

Syntaxe

```
f # date taille -42  min1 max1 prob1 min2 max2 prob2 min3 max3 prob3 ...  minN maxN probN
```

Exécution

Le premier nombre de chaque groupe est la valeur minimum de l'intervalle, le second est la valeur maximum et le troisième est la probabilité qu'un élément appartenant à cet intervalle de valeurs soit choisi par un algorithme aléatoire. Même si n'importe quel nombre peut être assigné à l'élément probabilité de chaque groupe, il vaut mieux lui donner une valeur en pourcentage, afin de rendre les choses plus claires pour l'utilisateur.

Ce sous-programme est prévu pour être utilisé avec les opcodes *duserrnd* et *urd* (voir *duserrnd* pour plus d'information). Comme ni *duserrnd* ni *urd* n'utilisent l'interpolation, il est suggéré de donner une taille raisonnablement grande.

Crédits

Auteur : Gabriel Maldonado

GEN43

GEN43 — Charge un fichier PVOCEX contenant une analyse VP.

Description

Ce sous-programme charge un fichier PVOCEX contenant l'analyse VP (amp-fréq) d'un fichier son et calcule les magnitudes moyennes de toutes les trames d'analyse d'un ou de tous les canaux audio. Il crée ensuite une table avec ces magnitudes pour chaque bin VP.

Syntaxe

```
f # date taille 43 codfic canal
```

Initialisation

taille -- nombre de points dans la table, puissance de deux ou puissance-de-deux plus 1. *GEN43* ne fait aucune distinction entre ces deux tailles, mais la table doit avoir pour taille au moins la moitié de celle de la tfr. Les bins VP couvrent le spectre positif de 0 Hz (index 0 de la table) à la fréquence de Nyquist (index *tailletfr*/2+1 de la table) par incréments réguliers (de taille *sr*/*tailletfr*).

codfic -- un fichier pvocex (qui peut être généré par pvanal).

canal -- numéro du canal audio duquel les magnitudes seront extraites ; un 0 donnera la moyenne des magnitudes de tous les canaux.

La lecture s'arrête à la fin du fichier.

Note

Si p4 est positif, la table sera post-normalisée. Un p4 négatif empêchera la post-normalisation.

Exemples

```
f1 0   512  43 "viola.pvx" 1
f1 0 -1024 -43 "noiseprint.pvx" 0
```

On peut utiliser cette table comme table de masquage pour *pvstencil* et *pvsmaska*. Le premier exemple utilise un fichier d'analyse de vocodeur de phase par TFR à 1024 points duquel on utilise le premier canal. Le second utilise tous les canaux d'un fichier de 2048 points, sans post-normalisation. Pour les applications à la réduction de bruit avec *pvstencil*, il est mieux de ne pas normaliser la table (code GEN négatif).

Crédits

Auteur : Victor Lazzarini

GEN51

GEN51 — Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

Description

Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

Syntaxe

```
f # date taille -51 nbrdegres intervalle freqbase touchebase rapport1 rapport2  .... rapportN
```

Exécution

Les quatre premiers paramètres (c'est-à-dire p5, p6, p7 et p8) définissent les directives de génération suivantes :

p5 (nbrdegres) -- le nombre de degrés de l'échelle microtonale

p6 (intervalle) -- l'intervalle de fréquences couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.

p7 (freqbase) -- la fréquence de base de l'échelle en cps

p8 (touchebase) -- L'indice entier dans la table auquel assigner la fréquence de base inchangée

Les autres paramètres définissent les rapports de l'échelle :

p9 ... pN (rapport1 ... etc.) -- les rapports des degrés de l'échelle

Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 cps assignée à la touche numéro 60, l'instruction f de la partition pour générer la table serait :

```
;            nbrdegrés       fréqbase            rapports  (tempérament égal) .......
;                    intervalle       touchebase
f1 0 64 -51      12        2      261      60        1 1.059463 1.12246 1.18920 ..etc...
```

Après le calcul du gen, la table f1 est remplie avec 64 valeurs de fréquences différentes. Le 60ème élément est rempli avec la valeur de fréquence 261, et tous les autres éléments de la table (précédents et suivants) sont remplis selon les rapports des degrés.

Un autre exemple avec une échelle de 24 degrés, une fréquence de base de 440 cps assignée à la touche numéro 48, et un intervalle de répétition de 1,5 :

```
;            nbrdegrés       fréqbase            rapports  .......
;                    intervalle       touchebase
f1 0 64 -51      24       1.5     440      48        1   1.01  1.02  1.03  ..etc...
```

Crédits

Auteur : Gabriel Maldonado

GEN52

GEN52 — Crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*.

Description

GEN52 crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*. Il peut aussi être utilisé pour extraire un canal d'une table multicanaux et le stocker dans une table mono normale, copier des tables en omettant certains échantillons, ajouter un délai, ou stocker en ordre inverse, etc.

Il faut donner trois paramètres pour chaque canal à traiter. *fsrc* déclare le numéro de la f-table source. Le paramètre *offset* spécifie un décalage pour le fichier source. S'il est différent de 0, le fichier source n'est pas lu depuis le début, un nombre *offset* de valeurs étant ignorées. L'*offset* est utilisé pour déterminer le numéro de canal à lire depuis les f-tables entrelacées, par exemple pour le canal 2, *offset* doit valoir 1. Il peut aussi être utilisé pour fixer un décalage de lecture sur la table source. Ce paramètre donne des valeurs absolues, si bien que si l'on désire un décalage de 20 unités d'échantillonnage pour une f-table à deux canaux, *offset* doit valoir 40. Le paramètre *srcchnls* est utilisé pour fixer le nombre de canaux dans la f-table source. Ce paramètre fixe la taille du pas de progression lors de la lecture de la f-table source.

Quand il y a plus d'un canal (*nchannels* > 1), les f-tables source sont entrelacées dans la table nouvellement créée.

Si la f-table source est finie avant que la f-table destination ne soit remplie, les valeurs restantes sont fixées à 0.

Syntaxe

```
f # date taille 52 ncanaux fsrc1 offset1 srcchnls1 [fsrc2 offset2 srcchnls2 ... fsrcN offsetN srcchnlsN
```

Exemple

```
; tables sources
f 1 0 16384 10 1
f 2 0 16384 10 0 1
; crée une table avec 2 canaux entrelacés
f 3 0 32768 -52 2 1 0 1 2 0 1
; extrait le premier canal de la table 3
f 4 0 16384 -52 1 3 0 2
; extrait le second canal de la table 3
f 5 0 16384 -52 1 3 1 2
```

Crédits

Auteur : Istvan Varga

Les Programmes Utilitaires

Dan Ellis, MIT Media Lab

Les utilitaires de Csound sont des programmes de *prétraitement de fichier son* qui retournent de l'information sur un fichier son ou qui créent une version d'analyse de celui-ci à utiliser par certains générateurs de Csound. Bien que destinés à différents usages, ils ont en commun le mécanisme d'accès au fichier son et sont descriptibles comme un ensemble. Les programmes Utilitaires de Fichiers Son peuvent être appelés de deux manières équivalentes :

`csound` [`-U` *`nomutilitaire`*] [`options`] [*`noms_fichier`*]

`nomutilitaire` [`options`] [*`noms_fichier`*]

Dans le premier cas, l'utilitaire est appelé comme une partie de l'exécutable de Csound, tandis que dans le second il est appelé comme un programme autonome. Le second est plus petit d'environ 200 K, mais les deux formes fonctionnent de manière identique. La première est pratique pour éviter la maintenance et l'utilisation de plusieurs programmes indépendants - un programme fait tout. Quand on utilise cette forme, un *drapeau -U* detecté dans la ligne de commande provoquera l'interprétation des options et des noms suivants comme ceux de l'utilitaire nommé ; cela signifie que le mécanisme de génération de Csound ne sera pas invoqué et que le programme se terminera à la fin du traitement par l'utilitaire.

Répertoires.

Les noms de fichier sont de deux sortes, fichiers son sources et fichiers d'analyse résultants. Chacun a une convention de nommage hiérarchique, influencée par le répertoire depuis lequel l'utilitaire est appelé. Les fichiers son sources avec un nom de chemin complet (commençant par un point (.), une barre oblique (/), ou pour ThinkC incluant un deux-points (:)), ne seront cherchés que dans le répertoire nommé. Les fichiers son sans chemin seront recherchés d'abord dans le répertoire courant, ensuite dans le répertoire nommé par la variable d'environnement SSDIR (si elle est définie), ensuite dans le répertoire nommé par SFDIR. Une recherche infructueuse retournera une erreur "cannot open".

Les fichiers d'analyse résultants sont écrits dans le répertoire courant, ou le répertoire nommé si un chemin est inclus. Pour être ordonné, il est bon de séparer les fichiers d'analyse des fichiers son, habituellement dans un répertoire différent référencé par la variable d'environnement SADIR. Il est commode de lancer l'analyse depuis le répertoire SADIR. Quand un fichier d'analyse est invoqué ultérieurement par un générateur de Csound il est cherché en premier dans le répertoire courant, puis dans le répertoire défini par SADIR.

Formats des Fichiers Son.

Csound peut lire et écrire des fichiers audio dans différents formats. Les formats d'écriture sont décrits par des options de la commande Csound. En lecture, le format est déterminé par l'en-tête du fichier, et les données sont automatiquement converties en virgule flottante pendant le traitement interne. Quand Csound est installé sur un hôte qui a des conventions de fichier son locales (SUN, NeXT, Macintosh) il peut comprendre de manière conditionnelle du code local qui crée des fichiers son non portables vers d'autres hôtes. Cependant, sur tous les hôtes, Csound peut toujours générer et lire des fichiers du type AIFF, qui est ainsi un format portable. Les bibliothèques de sons échantillonnés sont typiquement en AIFF, et la variable d'environnement SSDIR pointe habituellement vers un répertoire contenant de tels sons. S'il est défini, le répertoire SSDIR fait partie des chemins de recherche pour l'accès aux fichiers son. Noter que certains sons échantillonnés AIFF ont un mécanisme de boucle audio pour les notes tenues ; les programmes d'analyse ne parcourent les segments de boucle qu'une fois.

Pour les fichiers son sans en-tête, une valeur *SR* peut être fournie par l'*option -r* (ou sa valeur par dé-

faut). Si l'*en-tête SR* et l'option de ligne de commande sont tous deux présents, la valeur de l'option remplacera l'en-tête.

Quand les programmes d'Analyse accèdent à un son, un seul canal est lu. Pour les fichiers stéréo ou quadro, le canal par défaut est le canal un ; d'autres canaux peuvent être obtenus à la demande.

Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL)

Les utilitaires suivants existent pour l'analyse d'un Fichier Son :

- *ATSA* : Analyse ATS à utiliser avec les opcodes de Csound de *Resynthèse ATS*.
- *CVANAL* : Analyse de Fourier d'une Réponse Impulsionnelle pour l'opérateur *convolve*.
- *HETRO* : Analyse hétérodyne pour le générateur de Csound *adsyn*.
- *LPANAL* : Analyse de codage prédictif linéaire pour les opcodes de Csound de *Resynthèse par Codage Prédictif Linéaire (LPC)*.
- *PVANAL* : Analyse par vocodeur de phase pour le générateur de Csound *pvoc*.

atsa

atsa — Effectue une analyse ATS sur un fichier son.

Description

Analyse ATS à utiliser avec les opcodes de Csound de *Resynthèse ATS*.

Syntaxe

```
csound -U atsa [options] nomfichier_entree nomfichier_sortie
```

Initialisation

Les options suivantes peuvent être positionnées pour atsa. (Les valeurs par défaut sont mises entre parenthèses) :

-b début (0,000000 secondes)
-e durée (0,000000 secondes, signifie jusqu'à la fin)
-l fréquence la plus basse (20,000000 Hz)
-H fréquence la plus haute (20000,000000 Hz)
-d déviation en fréquence (0,100000 de la fréquence d'un partiel)
-c cycles par fenêtre (4 cycles)
-w type de fenêtre (type : 1) (Options : 0=BLACKMAN, 1=BLACKMAN_H, 2=HAMMING, 3=VONHANN)
-h taille de saut (0,250000 de la taille de fenêtre)
-m magnitude la plus faible (-60,000000)
-t longueur de trajectoire (3 trames)
-s longueur minimale de segment (3 trames)
-g longueur minimale des blancs (3 trames)
-T seuil du SMR (30,000000 dB SPL)
-S SMR Minimum de Segment (60,000000 dB SPL)
-P contribution du dernier pic (0,000000 des paramètres du dernier pic)
-M contribution du SMR (0,500000)
-F Type de Fichier (type : 4) (Options : 1=amp. et fréq. seulement, 2=amp., fréq. et phase, 3=amp., fréq. et résiduel, 4=amp., fréq., phase et résiduel)

Paramètres

L'analyse ATS a été conçue par Juan Pampin. Pour une information complète sur ATS visiter : http://www-ccrma.stanford.edu/~juan/ATS.html.

Les paramètres d'analyse doivent être réglés soigneusement pour l'Algorithme d'Analyse (ATSA) afin de capturer la nature du signal à analyser. Comme ils sont nombreux, ATSH offre la possibilité de les Sauvegarder/Charger dans un Fichier Binaire portant l'extension "`.apf`". L'extension n'est pas obligatoire, mais recommandée. Une brève explication de chaque Paramètre d'Analyse suit :

1. Début (secs.): la date de début de l'analyse en secondes.

2. Durée (secs.): la durée de l'analyse en secondes. Un zéro signifie la durée entière du fichier son en entrée.

3. Fréquence la Plus Basse (Hz) : ce paramètre va déterminer partiellement la taille de la Fenêtre d'Analyse à utiliser. Pour calculer la taille de la Fenêtre d'Analyse, la période de la Fréquence la Plus Basse en échantillons (*SR* / *LF*) est multipliée par le nombre de cycles de celle-ci que l'utilisateur veut

caser dans la Fenêtre d'Analyse (voir paramètre 6). Cette valeur est arrondie à la plus proche puissance de deux supérieure pour déterminer la taille de la TFR pour l'analyse. Les échantillons en trop sont remplis par des zéros. Si le signal est un son unique, harmonique, alors la valeur de la Fréquence la Plus Basse sera celle du fondamental ou d'un sous-harmonique de celui-ci. Si le son n'est pas harmonique, alors sa fréquence significative la plus basse pourra être une bonne valeur de départ.

4. Fréquence la Plus Haute (Hz) : fréquence la plus haute à prendre en compte pour la Détection de Pic. Une fois que l'on sait qu'aucune information pertinente ne se trouve au-delà d'une certaine fréquence, l'analyse peut être plus rapide et plus précise en réglant la Fréquence la Plus Haute à cette valeur.

5. Déviation de Fréquence (Rapport) : déviation de fréquence autorisée pour chaque pic dans l'Algorithme de Continuation des Pics, comme fraction de la fréquence concernée. Par exemple, si l'on considère un pic à 440 Hz et une Déviation de 0,1 l'Algorithme de Continuation des Pics n'essaiera de trouver des candidats pour la continuité qu'entre 396 et 484 Hz (10% au-dessus et en-dessous de la fréquence du pic). Une petite valeur produira probablement plus de trajectoires tandis qu'une grande valeur les réduira, mais au prix d'une plus grande difficulté à traiter l'information par la suite.

6. Nombre de Cycles de la Fréquence la Plus Basse à caser dans une Fenêtre d'Analyse : il déterminera aussi partiellement la taille de la Fenêtre de Transformation de Fourier à utiliser. Voir le paramètre 3. Pour des signaux à un seul harmonique, il est supposé être supérieur à 1 (typiquement 4).

7. Taille de Saut (Rapport) : taille de l'intervalle entre une Fenêtre d'Analyse et la suivante exprimée comme une fraction de la Taille de Fenêtre. Par exemple, une Taille de Saut de 0,25 "sautera" de 512 échantillons (les Fenêtres se chevaucheront sur 75% de leur taille). Ce paramètre déterminera aussi la taille des trames d'analyse obtenues. Les signaux qui changent leur spectre très rapidement (comme les sons de la Parole) peuvent nécessiter un taux de trame élevé afin de suivre au mieux leurs changements.

8. Limite d'Amplitude (dB) : la valeur d'amplitude la plus élévée à prendre en compte pour la Détection de Pic.

9. Type de Fenêtre : la forme de la fonction de lissage à utiliser pour l'Analyse de Fourier. Il y a quatre choix possibles pour le moment : Blackman, Blackman-Harris, Von Hann, et Hanning. Des spécifications précises sur celles-ci se trouvent facilement dans la bibliographie sur le traitement numérique du signal.

10. Longueur de Trajectoire (Trames) : L'Algorithme de Continuation des Pics regardera "en arrière" sur un nombre de trames égal à Longueur afin de réaliser sa tâche au mieux, et d'éviter que les trajectoires de fréquence ne s'incurvent trop et perdent leur stabilité. Cependant, une grande valeur pour ce paramètre ralentira l'analyse de manière significative.

11. Longueur Minimale de Segment (Trames) : une fois l'analyse réalisée, les données spectrales peuvent être "nettoyées" durant le post-traitement. Les trajectoires plus petites que cette valeur sont supprimées si leur SMR moyen est inférieur au SMR Minimum de Segment (voir les paramètres 16 et 14). Ceci peut aider à éviter les changements soudains non pertinents tout en gardant un taux de trames élevé, réduisant aussi le nombre de sinusoïdes épisodiques durant la synthèse.

12. Longueur Minimale des Blancs (Trames): comme le paramètre 11, celui-ci est aussi utilisé pour nettoyer les données durant le post-traitement. Dans ce cas, les blancs (valeurs d'amplitude nulle, c'est-à-dire le "silence" théorique) contigus dont le nombre de trames est plus grand que Longueur sont remplis avec des valeurs d'amplitude/fréquence obtenues par interpolation linéaire des trames actives adjacentes. Ce paramètre empêche les interruptions soudaines des trajectoires stables tout en gardant un taux de trames élevé.

13. Seuil du SMR (dB SPL) : également un paramètre de post-traitement, le seuil du SMR est utilisé pour éliminer les partiels avec de faibles moyennes.

14. SMR Minimum de Segment (dB SPL) : ce paramètre est utilisé en combinaison avec le paramètre 11. Les segments courts ayant un SMR moyen inférieur à cette valeur seront supprimés durant le post-traitement.

15. Contribution du Dernier Pic (0 à 1) : comme c'est expliqué dans le Paramètre 10, l'Algorithme de Continuation des Pics regarde "en arrière" sur plusieurs trames afin de réaliser sa tâche au mieux. Ce paramètre aidera à pondérer la contribution du premier des pics précédents sur les autres. Une valeur de zéro signifie que tous les pics précédents (jusqu'à la taille du Paramètre 10) sont pris également en compte.

16. Contribution du SMR (0 à 1) : en plus de la proximité en fréquence des pics, l'Algorithme de Continuation des Pics ATS peut utiliser une information psychoacoustique (le Rapport Signal-Masque, ou SMR) pour améliorer les résultats perceptifs. Ce paramètre indique quelle quantité d'information SMR est utilisée durant la détection. Par exemple, une valeur de 0,5 fait que l'Algorithme de Continuation des Pics utilise 50% d'information SMR et 50% d'information de Proximité en Fréquence pour décider quel est le meilleur candidat pour continuer la trajectoire sinusoïdale.

Exemples

La commande suivante :

```
atsa -b0.1 -e1 -l100 -H10000 -w2 fichieraudio.wav fichieraudio.ats
```

Génère le fichier d'analyse ATS '`fichieraudio.ats`' à partir du fichier original '`fichieraudio.wav`'. L'analyse commence à partir de 0,1 seconde dans le fichier et elle est effectuée sur 1 seconde. La fréquence la plus basse est 100 Hz et la plus haute est 10 kHz. Une fenêtre de Hamming est utilisée pour chaque trame d'analyse.

cvanal

cvanal — Convertit un fichier son en une trame de transformée de Fourier.

Description

Analyse de Fourier d'une Réponse Impulsionnelle pour l'opérateur *convolve*

Syntaxe

```
csound -U cvanal [options] nomfichier_entree nomfichier_sortie
```

```
cvanal [options] nomfichier_entree nomfichier_sortie
```

Initialisation

cvanal -- convertit un fichier son en une trame de transformée de Fourier. Le fichier de sortie peut être utilisé par l'opérateur *convolve* pour réaliser une Convolution Rapide entre un signal d'entrée et la réponse impulsionnelle originale. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

-s srate -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur srate de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

-c canal -- numéro du canal à traiter. S'il est omis, tous les canaux sont traités par défaut. Si une valeur est donnée, seul le canal choisi sera traité.

-b début -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

-d durée -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

Exemples

```
cvanal unson fichiercv
```

analysera le fichier son "unson" pour produire le fichier "fichiercv" à utiliser avec *convolve*.

Pour utiliser des données qui ne sont pas déjà contenues dans un fichier son, un convertisseur de fichier son qui accepte des fichiers texte peut être utilisé pour créer un fichier audio standard, par exemple le format .DAT pour SOX. Ceci est utile pour implémenter des filtres RIF.

Fichiers

Le fichier de sortie a un en-tête spécial *convolve*, contenant les détails du fichier source audio. Les données d'analyse sont stockées comme des nombres « virgule flottante », en forme rectangulaire (réel/imaginaire).

Note

Le fichier d'analyse n'est *pas* indépendant du système ! Assurez-vous que les données originales de la réponse impulsionnelle sont conservées. Si nécessaire, le fichier d'analyse pourra être recréé.

Crédits

Auteur : Greg Sullivan

Basé sur l'algorithme donné dans *Elements Of Computer Music*, par F. Richard Moore.

hetro

hetro — Décompose un fichier son en entrée en composantes sinusoïdales.

Description

Analyse par filtre hétérodyne pour le générateur de Csound *adsyn*.

Syntaxe

```
csound -U hetro [options] nomfichier_entree nomfichier_sortie
```

```
hetro [options] nomfichier_entree nomfichier_sortie
```

Initialisation

hetro prend un fichier son en entrée, le décompose en composantes sinusoïdales, et sort une description de ces composantes sous la forme de pistes de points charnière d'amplitude et de fréquence. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre drapeau et argument.

-s srate -- taux d'échantillonnage du fichier audio en entrée. Il remplacera la valeur srate de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000. Noter que pour la synthèse *adsyn* le taux d'échantillonnage du fichier source et de l'orchestre générateur n'ont pas à être les-mêmes.

-c canal -- numéro du canal à traiter. La valeur par défaut est 1.

-b début -- date de début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

-d durée -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier. La longueur maximale est de 32,766 secondes.

-f freqdeb -- fréquence de départ estimée du fondamental, nécessaire pour initialiser l'analyse par le filtre. La valeur par défaut est 100 (cps).

-h partiels -- nombre d'harmoniques recherchés dans le fichier audio. La valeur par défaut est 10, la valeur maximale dépend de la mémoire disponible.

-M ampmax -- amplitude maximale obtenue par addition sur toutes les pistes simultanées. La valeur par défaut est 32767.

-m ampmin -- seuil d'amplitude en-dessous duquel une paire de pistes amplitude/fréquence sera considérée comme inactive et ne contribuera pas à la somme en sortie. Valeurs typiques : 128 (48 dB en-dessous de l'échelle complète, 64 (54 dB en-dessous), 32 (60 dB en-dessous), 0 (pas de seuillage). Le seuil par défaut est 64 (54 dB en-dessous).

-n brkpts -- nombre initial de points charnière de l'analyse dans chaque piste d'amplitude et de fréquence, avant le seuillage (*-m*) et la consolidation linéaire des points charnière. Les points initiaux sont répartis uniformément sur toute la durée. La valeur par défaut est 256.

-l cutfreq -- substitue un filtre passe-bas de Butterworth du 3ème ordre avec une fréquence de coupure *cutfreq* (en Hz), à la place du filtre par défaut qui est un filtre de moyenne en peigne. La valeur par défaut est 0 (ne pas utiliser).

Exécution

A partir de Csound 4.08, *hetro* peut écrire des fichiers de sortie SDIF si le nom du fichier de sortie se termine par ".sdif" ou ".SDIF". Voir l'utilitaire *sdif2ad* pour plus d'information sur le support de SDIF dans Csound.

Exemples

```
hetro -s44100 -b.5 -d2.5 -h16 -M24000 fichieraudio.test adsynfile7
```

Ceci analyse 2,5 secondes du canal 1 du fichier "fichieraudio.test", enregistré à 44,1 kHz, commençant 0,5 secondes après le début, et place le résultat dans le fichier "adsynfile7". Nous ne voulons que les 16 premiers harmoniques du son, avec 256 points charnière par piste d'amplitude ou de fréquence, et un pic de la somme des amplitudes de 24000. Le fondamental est estimé au commencement à 100 Hz. Le seuil d'amplitude est de 54 dB en-dessous de l'échelle complète.

Le filtre passe-bas de Butterworth n'est pas activé.

Format de Fichier

Le fichier de sortie contient des suites temporelles de valeurs d'amplitude et de fréquence pour chaque harmonique d'une source audio additive complexe. L'information se présente sous la forme de points charnière (date, valeur, date, valeur, ...) en utilisant des entiers sur 16 bit dans l'intervalle 0 - 32767. Le temps est donné en millisecondes, et les fréquences en Hz (cps). Les données des points charnières sont exclusivement non-négatives, et les valeurs -1 et -2 signifient uniquement le début de nouvelles pistes d'amplitude et de fréquence. Une piste se termine par la valeur 32767. Avant d'être écrite en sortie, chaque piste subit une réduction de données par seuillage d'amplitude et consolidation linéaire des points charnière.

Un composant harmonique est défini par deux ensembles de points charnière : un ensemble d'amplitudes, et un ensemble de fréquences. Dans un fichier composé ces ensembles peuvent apparaître dans n'importe quel ordre (amplitude, fréquence, amplitude ....; ou amplitude, amplitude, ..., puis fréquence, fréquence, ...). Durant la resynthèse par *adsyn* les ensembles sont automatiquement appariés (amplitude, fréquence) dans l'ordre dans lequel ils sont trouvés. Il doit y avoir un nombre égal de chaque sorte.

Un fichier de contrôle *adsyn* légal pourrait avoir le format suivant :

```
-1 temps1 valeur1 ... tempsK valeurK 32767 ; points charnière d'amplitude pour le partiel 1
-2 temps1 valeur1 ... tempsL valeurL 32767 ; points charnière de fréquence pour le partiel 1
-1 temps1 valeur1 ... tempsM valeurM 32767 ; points charnière d'amplitude pour le partiel 2
-2 temps1 valeur1 ... tempsN valeurN 32767 ; points charnière de fréquence pour le partiel 2
-2 temps1 valeur1 ..........
-2 temps1 valeur1 ..........               ; pistes appariables pour les partiels 3 et 4
-1 temps1 valeur1 ..........
-1 temps1 valeur1 ..........
```

Crédits

Auteur : Tom Sullivan
1992
Auteur : John ffitch
1994
Auteur : Richard Dobson
2000

Octobre 2002. Merci à Rasmus Ekman, pour l'addition d'une note au sujet du format SDIF.

lpanal

lpanal — Effectue une analyse par prédiction linéaire et par détection de hauteur sur un fichier son.

Description

Analyse par prédiction linéaire pour les opcodes de Csound *Resynthèse par Codage Prédictif Linéaire (LPC)*.

Syntaxe

```
csound -U lpanal [options] nomfichier_entree nomfichier_sortie
```

```
lpanal [options] nomfichier_entree nomfichier_sortie
```

Initialisation

lpanal effectue à la fois une analyse par lpc et par détection de hauteur sur un fichier son pour produire une suite ordonnée de *trames* d'information de contrôle appropriée pour la resynthèse avec Csound. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre le drapeau et sa valeur.

-a -- [stockage alternatif] demande à lpanal d'écrire un fichier avec les valeurs des pôles du filtre plutôt que les fichiers de coefficients de filtre habituels. Quand *lpread* / *lpreson* sont utilisés avec des fichiers de pôles, une stabilisation automatique est effectuée et le filtre ne deviendra pas incontrôlable. (C'est le réglage par défaut dans la GUI Windows) - Changé par Marc Resibois.

-s srate -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur srate de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

-c canal -- numéro du canal à traiter. La valeur par défaut est 1.

-b début -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

-d durée -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

-p npoles -- nombres de pôles pour l'analyse. La valeur par défaut est 34, le maximum 50.

-h taillesaut -- taille du saut (en échantillons) entre les trames d'analyse. Détermine le nombre de trames par seconde (*srate* / *taillesaut*) dans le fichier de contrôle en sortie. La taille des trames d'analyse est de *taillesaut* * 2 échantillons. La valeur par défaut est 200, le maximum 500.

-C chaîne -- texte pour le champ commentaire de l'en-tête du fichier lp. La valeur par défaut est une chaîne nulle.

-P mincps -- fréquence la plus basse (en Hz) pour la détection de hauteur. -P0 signifie pas de détection de hauteur.

-Q maxcps -- fréquence la plus haute (en Hz) pour la détection de hauteur. Plus l'intervalle de hauteurs est étroit, plus l'estimation de hauteur est précise. Les valeurs par défaut sont -P70, -Q200.

-v verbosité -- niveau d'information affiché sur le terminal pendant l'analyse.

- 0 = aucune

- 1 = verbeux
- 2 = débogage

La valeur par défaut est 0.

Exemples

```
lpanal -a -p26 -d2.5 -P100 -Q400 fichieraudio.test lpfil22
```

analysera les premières 2,5 secondes du fichier "fichieraudio.test", produisant *srate* / 200 trames par seconde, chacune contenant les coefficients d'un filtre à 26 pôles et une estimation de hauteur entre 100 et 400 Hz. La sortie stabilisée (*-a*) sera placée dans "lpfil22" dans le répertoire courant.

Format de Fichier

La sortie est un fichier constitué d'un en-tête identifiable plus un ensemble de trames de données d'analyse en virgule flottante. Chaque trame contient quatre valeurs d'information de hauteur et de gain, suivies par *npoles* coefficients de filtre. Le fichier est lisible par l'opcode *lpread* de Csound.

lpanal est une modification importante des programmes d'analyse lpc de Paul Lanksy.

pvanal

pvanal — Convertit un fichier son en une série de trames de transformation de Fourier à court terme.

Description

Analyse de Fourier pour le générateur de Csound *pvoc*

Syntaxe

```
csound -U pvanal [options] nomfic_entree nomfic_sortie
```

```
pvanal [options] nomfic_entree nomfic_sortie
```

Extension de pvanal pour créer un fichier PVOC-EX.

L'utilitaire standard de Csound pvanal a été étendu pour permettre la création d'un fichier au format PVOC-EX, en utilisant l'interface existante. Pour créer un fichier PVOC-EX, le nom de fichier doit avoir comme extension « .pvx », par exemple « test.pvx ». La nécessité pour la taille de TFR d'être une puissance de deux n'est plus obligatoire ici, et n'importe quelle valeur positive est acceptée ; les nombres impairs sont arrondis en interne. Cependant, les tailles en puissance de deux sont toujours préférables pour toutes les applications normales.

Les drapeaux de sélection de canal sont ignorés, et tous les canaux de la source seront analysés et écrits dans le fichier de sortie, jusqu'à la limite, fixée à la compilation, de huit canaux. La taille de la fenêtre d'analyse (itaillefen) est fixée en interne au double de la taille de la TFR.

Initialisation

pvanal convertit un fichier son en une série de trames de transformation de Fourier à court terme (STFT) à espacement temporel régulier (une représentation du domaine fréquentiel). Le fichier de sortie peut être utilisé par *pvoc* pour générer des fragments audio basés sur le son échantillonné original, avec des échelles de temps et des hauteurs arbitraires et modifiées dynamiquement. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

-s srate -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur srate de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

-c canal -- numéro du canal à traiter. La valeur par défaut est 1.

-b début -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

-d durée -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie la fin du fichier.

-n tailletrame -- taille de trame STFT, le nombre d'échantillons dans chaque trame de l'analyse de Fourier. Doit être une puissance de deux dans l'intervalle 16 à 16384. Pour des résultats propres, une trame doit être plus grande que la période de hauteur la plus longue du son échantillonné. Cependant, des trames très longues donnent un "brouillage" temporel ou une réverbération. La largeur de bande de chaque bin de STFT est déterminée par le rapport *srate* / *tailletrame*. La taille de trame par défaut est la plus petite puissance de deux qui correspond à plus de 20 ms de la source (par exemple 256 points avec un échantillonage à 10 kHz, donnant une trame de 25,6 ms).

-w factfen -- facteur de chevauchement de fenêtre. Il contrôle le nombre de trames de transformation de Fourier par seconde. *pvoc* interpolera entre les trames, mais un nombre insuffisant de trames générera des distorsions audibles ; trop de trames donneront un fichier d'analyse gigantesque. 4 est un bon com-

promis pour *factfen*, signifiant que chaque point d'entrée apparaît dans 4 fenêtres de sortie, ou inversement que le décalage entre trames de STFT successives est *tailletrame* / 4. La valeur par défaut est 4. N'utilisez pas cette option en même temps que *-h*.

-h taillesaut -- décalage de trame STFT. Le contraire de l'option précédente, spécifiant l'incrément en échantillons entre les trames d'analyse successives (voir aussi *lpanal*). N'utilisez pas cette option en même temps que *-w*.

-H -- utilise une fenêtre de Hamming à la place de la fenêtre de von Hann employée par défaut.

-K -- utilise une fenêtre de Kaiser à la place de la fenêtre de von Hann employée par défaut. Le paramètre de la fenêtre de Kaiser vaut 6,8 par défaut, mais il peut être fixé avec l'option *-B*.

-B beta -- fixe le paramètre beta d'une fenêtre de Kaiser utilisée, à la valeur en virgule flottante *beta*.

Exemples

```
pvanal unson fichierpv
```

analysera le fichier son "unson" en utilisant les valeurs par défaut de *tailletrame* et de *factfen* pour produire le fichier "pvfile" approprié pour une utilisation avec *pvoc*.

Fichiers

Le fichier de sortie a un en-tête spécial *pvoc* contenant les détails du fichier source audio, le taux des trames d'analyse et le facteur de chevauchement. Les trames de données de l'analyse sont stockées en virgule flottante, avec la magnitude et la « fréquence » (en Hz) des N/2 + 1 premiers bins de Fourier de chaque trame successive. La « fréquence » encode l'incrément de phase de façon à donner une bonne indication de la fréquence réelle pour les harmoniques à fort niveau. Pour les faibles amplitudes ou les harmoniques évoluant rapidement c'est moins significatif.

Diagnostiques

Imprime le nombre total de trames, et le nombre de trames complétées toutes les 20 trames.

Crédits

Auteur : Dan Ellis

MIT Media Lab

Cambridge, Massachussetts

1990

Requêtes sur un Fichier (SNDINFO)

L'utilitaire suivant existe pour les requêtes sur un fichier son :

- *SNDINFO*: Affiche de l'information sur un fichier son.

sndinfo

sndinfo — Affiche de l'information sur un fichier son.

Description

Fournit l'information de base sur un ou plusieurs fichiers son.

Syntaxe

```
csound -U sndinfo [options] fichierson ...

sndinfo [options] fichierson ...
```

Initialisation

sndinfo tentera de trouver chaque fichier nommé, de l'ouvrir en lecture, de lire l'en-tête du fichier son, pour ensuite imprimer un rapport sur l'information de base trouvée. L'ordre de recherche dans les répertoires de fichiers son est celle qui a été décrite précédemment. Si le fichier est de type AIFF, quelques détails plus avancés sont listés en premier.

Il y a deux types d'options :

1. *-i* ou *-i1* imprimera l'information d'instrument, qui comprend les boucles. L'option continue jusqu'à une option *-i0*.

2. L'autre option est *-b* qui imprime l'information de diffusion pour les fichier WAV. Elle peut être arrêtée de façon similaire avec *-b0*.

Exemples

```
csound -U sndinfo test Bosendorfer/"BOSEN mf A0 st" foo foo2
```

où l'on a les variables d'environnement SFDIR = /u/bv/sound, et SSDIR = /so/bv/Samples, pourra produire ceci :

```
util  SNDINFO:
     /u/bv/sound/test:
           srate 22050, monaural, 16 bit shorts, 1.10 seconds
           headersiz 1024, datasiz 48500  (24250 sample frames)

    /so/bv/Samples/Bosendorfer/BOSEN mf A0 st:  AIFF, 197586 stereo samples, base Frq 261.6 (MIDI 60),
     AIFF soundfile, looping with modes 1, 0
     srate 44100, stereo, 16 bit shorts, 4.48 seconds

     headersiz  402, datasiz 790344  (197586 sample frames)

     /u/bv/sound/foo:
           no recognizable soundfile header

     /u/bv/sound/foo2:
            couldn't find
```

Conversion de Fichier (, HET_EXPORT, HET_IMPORT,

PVLOOK, PV_EXPORT, PV_IMPORT, SDIF2AD, SR-CONV)

Les utilitaires suivants existent pour la conversion de fichier :

- *HET_EXPORT* : Exporte un fichier `.het` (produit par *HETRO*) vers un fichier texte à séparateur virgule.
- *HET_IMPORT* : Génère un fichier `.het` (dans le format produit par *HETRO*) à partir d'un fichier texte à séparateur virgule pour l'utiliser avec le générateur *adsyn*.
- *PVLOOK* : affiche une sortie texte formatée de fichiers d'analyse STFT.
- *PV_EXPORT* : Convertit un fichier généré par *PVANAL* en un fichier texte.
- *PV_IMPORT* : Convertit un fichier texte (dans le format généré par *PV_EXPORT*) en un fichier de format *PVANAL* à utiliser par l'opcode *pvoc*.
- *SDIF2AD* : Convertit des fichiers SDIF en fichiers utilisables par adsynt.
- *SRCONV*: Convertit le taux d'échantillonnage d'un fichier audio.

dnoise

dnoise — Réduit le bruit dans un fichier.

Description

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel.

Syntaxe

```
dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree
```

Initialisation

Options spécifiques à dnoise :

- *(pas d'option)* fichier son en entrée à débruiter
- *-i nomfic* fichier de référence du bruit en entrée
- *-o nomfic* fichier son de sortie
- *-N fnum* nombre de filtres passe-bande (par défaut : 1024)
- *-w fovlp* facteur de chevauchement des filtres : {0,1,(2),3} NE PAS UTILISER *-w* ET *-M*
- *-M longfa* longueur de la fenêtre d'analyse (par défaut : N-1 à moins que *-w* ne soit spécifié)
- *-L longfs* longueur de la fenêtre de synthèse (par défaut : M)
- *-D factd* facteur de décimation (par défaut : M/8)
- *-b datedeb* date de début dans le fichier de référence du bruit (par défaut : 0)
- *-B smpdeb* échantillon de départ dans le fichier de référence du bruit (par défaut : 0)
- *-e datefin* date de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-E smpfin* échantillon de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-t seuil* seuil au-dessus du bruit de référence en dB (par défaut : 30)
- *-S gfact* raideur de la coupure au seuil de bruit, intervalle : 1 à 5 (par défaut : 1)
- *-n nbrtrm* nombre de trames de TFR sur lesquelles calculer la moyenne (par défaut : 5)
- *-m gainmin* gain minimum du seuillage de bruit lorsqu'il est fermé (par défaut : -40)

Options de format du fichier son :

- *-A* format de sortie AIFF
- *-W* format de sortie WAV

- *-J* format de sortie IRCAM
- *-h* pas d'en-tête de fichier (non valide pour une sortie AIFF/WAV)
- *-8* échantillons en caractères non signés sur 8 bit
- *-c* échantillons en caractères signés sur 8 bit
- *-a* échantillons en alaw
- *-u* échantillons en ulaw
- *-s* échantillons en entiers courts
- *-l* échantillons en entiers longs
- *-f* échantillons en virgule flottante. Les nombres en virgule flottante sont aussi supportés par les fichiers WAV. (Nouveau dans Csound 3.47.)

Options supplémentaires :

- *-R* verbose - impression d'une information d'état
- *-H [N]* imprime un caractère de type pulsation à chaque écriture dans le fichier son.
- *-- nomfic* sortie de journal dans le fichier nomfic
- *-V* verbose - impression d'une information d'état

Note

DNOISE consulte aussi la variable d'environnement SFOUTYP pour déterminer le format du fichier de sortie.

L'option -i est utilisée pour un fichier de référence du bruit (créé normalement à partir d'un court extrait du fichier à débruiter, dans lequel seul le bruit est audible). Le fichier son d'entrée à débruiter peut être donné n'importe où dans la ligne de commande, sans drapeau.

Exécution

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel. Il fonctionnera mieux dans le cas d'un rapport signal/bruit élevé avec un bruit de type souffle.

L'algorithme est celui suggéré par Moorer & Berger dans « Linear-Phase Bandsplitting: Theory and Applications » presenté à la 76ème Convention, 8-11 Octobre 1984 à New York, de l'Audio Engineering Society (préimpression #2132) sauf qu'il utilise la formulation par Chevauchement-Addition Pondéré pour l'analyse et la synthèse de Fourier à court terme au lieu de la formulation récursive proposée par Moorer & Berger. Le gain pour chaque bin de fréquence est calculé indépendamment selon la formule

gain = g0 + (1-g0) * [moy / (moy + th*th*nref)] ^ sh

où *moy* et *nref* sont la moyenne quadratique du signal et du bruit respectivement pour le bin en question. (Ceci diffère légèrement de la version dans Moorer & Berger.)

Les paramètres critiques *th* et *g0* sont spécifiés en dB et convertis en interne en valeurs décimales. Les

valeurs *nref* sont calculées au début du programme sur la base d'un fichier de bruit (spécifié dans la ligne de commande) qui contient du bruit sans signal.

Les valeurs *moy* sont calculée sur une fenêtre rectangulaire de m trames de TFR centrée sur la date courante. Cela correspond à une extension temporelle de m*D/R (qui vaut typiquement (m*N/8)/R). Le réglage par défaut de N, m, et D devrait convenir pour la plupart des utilisations. Un taux d'échantillonnage supérieur à 16 kHz pourrait signifier un N plus grand.

Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitch

30 décembre 2000

Mis à jour par Rasmus Ekman le 11 mars 2002.

het_export

het_export — Convertit un fichier .het en fichier texte à séparateur virgule.

Syntaxe

```
het_export fichier_het fichier_textecsv
```

```
csound -U het_export fichier_het fichier_textecsv
```

Initialisation

fichier_het - Nom du fichier d'entrée .het.

fichier_textecsv - Nom du fichier texte à séparateur virgule.

L'utilitaire *het_export* génère un fichier texte à séparateur virgule pour pouvoir éditer manuellement un fichier .het produit par l'utilitaire *HETRO*. On peut l'utiliser en combinaison avec *het_import* pour produire des données pour le générateur *adsyn*.

Crédits

Auteur : John ffitch

1995

het_import

het_import — Convertit un fichier texte à séparateur virgule en un fichier .het

Syntaxe

```
het_import fichier_textecsv fichier_het
```

```
csound -U het_import fichier_textecsv fichier_het
```

Initialisation

fichier_textecsv - Nom du fichier texte à séparateur virgule.

fichier_het - Nom du fichier .het de sortie.

L'utilitaire *het_import* génère un fichier *.het* utilisable avec le générateur *adsyn*. Il peut être utilisé en combinaison avec *het_export* pour modifier l'analyse du son faite par l'utilitaire *HETRO*.

Crédits

Auteur : John ffitch

1995

pvlook

pvlook — Affiche une sortie texte formatée de fichiers d'analyse STFT.

Description

Affiche une sortie texte formatée de fichiers d'analyse STFT créés avec *pvanal*.

Syntaxe

```
csound -U pvlook [options] fichier_entree
```

```
pvlook [options] fichier_entree
```

Initialisation

pvlook lit un fichier, et les trajectoires de fréquence et d'amplitude pour chacun des bins de l'analyse, dans un format texte lisible. Le fichier est supposé être un fichier d'analyse STFT crée par *pvanal*. Par défaut, le fichier entier est traité.

-bb n -- commence au bin d'analyse numéro *n*, numérotés à partir de 1. La valeur par défaut est 1.

-eb n -- termine au bin d'analyse numéro *n*. Vaut par défaut la valeur la plus haute.

-bf n -- commence à la trame d'analyse numéro *n*, numérotées à partir de 1. La valeur par défaut est 1.

-ef n -- termine à la trame d'analyse numéro *n*. Vaut par défaut la valeur la plus haute.

-i -- imprime les valeurs en entier. Par défaut en virgule flottante.

Exemples

```
$ csound -U pvlook test.pv
Using csound.txt
Csound Version 3.57 (Aug  3 1999)
util PVLOOK:
; Bins in Analysis: 513
; First Bin Shown: 1
; Number of Bins Shown: 513
; Frames in Analysis: 1184
; First Frame Shown: 1
; Number of Data Frames Shown: 1184

Bin 1 Freqs.0.000 87.891 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 -87.891 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 87.891 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000

Bin 1 Amps. 0.180 0.066 0.252 0.248 0.245 0.246 0.246 0.249
0.252 0.251 0.250 0.248 0.244 0.245 0.248 0.250 0.254 0.251
0.248 0.247 0.244 0.246 0.249 0.250 0.253 0.251 0.247 0.246
0.245 0.246 0.250 0.251 0.252 0.250 0.247 0.245 0.246 0.247
0.251 0.252 0.250 0.249 0.246 0.245 0.248 0.249 0.252 0.253
0.249 0.248 0.245 0.245 0.249 0.251 0.252 0.252 0.249 0.246
0.246 0.245 0.249 0.252 0.252 0.251 0.249 0.245 0.246 0.248
0.250 0.253 0.251 0.249 0.247 0.244 0.247 0.249 0.250 0.253
0.251 0.248 0.247 0.245 0.247 0.250 0.252 0.252 0.251 0.247
0.246 0.246 0.247 0.251 0.252 0.251 0.249 0.246 0.245 0.248
0.249 0.252 0.252 0.249 0.248 0.246 0.245 0.249 0.250 0.252
0.252 0.249 0.247 0.246 0.246 0.249 0.252 0.252 0.251 0.248
0.245 0.246 0.247 0.249 0.253 0.251 0.249 0.247 0.245 0.246
0.248 0.250 0.253 0.251 0.248 0.247 0.244 0.246 0.250 0.251
0.252 0.250 0.247 0.246 0.246 0.248 0.251 0.252 0.251 0.250
0.246 0.245 0.247 0.248 0.251 0.252 0.250 0.248 0.246 0.245
0.248 0.249 0.252 0.252 0.248 0.247 0.245 0.245 0.249 0.251
0.251 0.251 0.248 0.246 0.246 0.247 0.250 0.252 0.251 0.250
0.248 0.244 0.246 0.248 0.250 0.253 0.251 0.248 0.247 0.245
0.247 0.249 0.250 0.252 0.250 0.247 0.246 0.245 0.247 0.251
0.252 0.251 0.250 0.246 0.245 0.247 0.248 0.252 0.252 0.249
0.248 0.245 0.245 0.248 0.249 0.251 0.252 0.248 0.247 0.245
0.245 0.249 0.250 0.251 0.251 0.248 0.246 0.245 0.246 0.249
0.252 0.251 0.250 0.247 0.244 0.246 0.247 0.249 0.252 0.251
0.249 0.247 0.244 0.247 0.249 0.250 0.252 0.250 0.247 0.246
0.245 0.247 0.250 0.251 0.251 0.250 0.246 0.245 0.246 0.248
0.251 0.252 0.250 0.249 0.245 0.245 0.247 0.248 0.251 0.252
0.249 0.247 0.245 0.245 0.248 0.250 0.251 0.251 0.247 0.246
0.245 0.245 0.249 0.251 0.251 0.250 0.247 0.245 0.246 0.246
0.249 0.252 0.251 0.249 0.247 0.244 0.247 0.248 0.250 0.252
0.250 0.247 0.246 0.245 0.247 0.250 0.251 0.252 0.249 0.246
0.245 0.245 0.247 0.251 0.251 0.250 0.249 0.246 0.245 0.247
0.248 0.251 0.251 0.249 0.248 0.245 0.245 0.248 0.249 0.251
0.251 0.248 0.246 0.245 0.245 0.249 0.251 0.251 0.251 0.247
0.245 0.245 0.246 0.249 0.251 0.250 0.249 0.247 0.244 0.246
0.248 0.250 0.252 0.250 0.247 0.246 0.245 0.247 0.249 0.250
0.251 0.249 0.246 0.246 0.245 0.247 0.250 0.250 0.250 0.249
0.245 0.245 0.246 0.248 0.251 0.251 0.249 0.248 0.245 0.245
0.247 0.249 0.251 0.251 0.248 0.246 0.245 0.245 0.248 0.250
0.251 0.250 0.247 0.245 0.245 0.246 0.249 0.251 0.250 0.249
0.246 0.244 0.246 0.247 0.250 0.251 0.250 0.248 0.246 0.245
0.247 0.249 0.250 0.251 0.249 0.247 0.246 0.245 0.247 0.250
0.250 0.251 0.248 0.245 0.245 0.246 0.248 0.251 0.251 0.249
0.248 0.245 0.245 0.247 0.249 0.251 0.251 0.248 0.247 0.245
0.245 0.248 0.249 0.250 0.250 0.247 0.246 0.246 0.246 0.249
0.251 0.250 0.250 0.246 0.245 0.246 0.247 0.250 0.251 0.249
0.248 0.246 0.244 0.246 0.248 0.250 0.251 0.249 0.247 0.246
0.245 0.247 0.250 0.250 0.251 0.249 0.245 0.245 0.246 0.248
0.251 0.250 0.250 0.248 0.245 0.245 0.247 0.248 0.251 0.250
0.248 0.247 0.245 0.246 0.248 0.250 0.251 0.250 0.247 0.246
0.245 0.246 0.249 0.251 0.250 0.249 0.246 0.245 0.246 0.247
0.250 0.251 0.250 0.249 0.246 0.244 0.246 0.248 0.250 0.251
0.249 0.247 0.246 0.245 0.247 0.249 0.250 0.251 0.287 0.331
0.178 0.008 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.140 1.265 2.766 3.289 3.296 3.293 3.296 3.296 3.290 3.293
3.292 3.291 3.297 3.295 3.294 3.296 3.291 3.292 3.294 3.291
3.296 3.297 3.292 3.295 3.292 3.290 3.295 3.293 3.294 3.297
3.292 3.293 3.294 3.290 3.295 3.295 3.292 3.296 3.293 3.291
3.294 3.291 3.293 3.297 3.292 3.295 3.294 3.288 3.293 3.293
3.292 3.297 3.294 3.292 3.295 3.290 3.292 3.295 3.292 3.295
3.295 3.290 3.294 3.292 3.292 3.297 3.293 3.293 3.295 3.290
3.292 3.293 3.290 3.296 3.296 3.292 3.295 3.291 3.290 3.294
3.291 3.294 3.296 3.291 3.293 3.293 3.290 3.295 3.294 3.293
3.296 3.291 3.291 3.293 3.290 3.294 3.296 3.292 3.295 3.293
3.288 3.293 3.292 3.292 3.297 3.292 3.293 3.294 3.289 3.292
3.294 3.291 3.296 3.293 3.291 3.294 3.291 3.292 3.296 3.292
3.294 3.295 3.289 3.292 3.292 3.291 3.296 3.294 3.292 3.295
3.290 3.290 3.293 3.291 3.295 3.296 3.291 3.294 3.291 3.289
3.294 3.292 3.293 3.295 3.291 3.292 3.293 3.290 3.294 3.295
3.292 3.294 3.291 3.289 3.293 3.291 3.293 3.296 3.292 3.293
3.293 3.288 3.292 3.293 3.292 3.296 3.293 3.291 3.294 3.289
3.292 3.295 3.291 3.294 3.293 3.289 3.292 3.291 3.290 3.295
3.293 3.292 3.294 3.289 3.291 3.293 3.290 3.295 3.294 3.290
3.293 3.290 3.289 3.294 3.291 3.293 3.295 3.290 3.292 3.292
3.289 3.293 3.293 3.292 3.295 3.291 3.289 3.292 3.290 3.292
3.295 3.291 3.293 3.292 3.288 3.292 3.291 3.291 3.295 3.291
3.291 3.292 3.289 3.291 3.294 3.291 3.294 3.292 3.289 3.292
3.290 3.290 3.295 3.292 3.293 3.294 3.289 3.291 3.292 3.290
3.294 3.293 3.291 3.293 3.289 3.290 3.293 3.291 3.294 3.295
3.290 3.292 3.291 3.289 3.294 3.293 3.292 3.294 3.290 3.290
3.292 3.289 3.293 3.294 3.291 3.293 3.291 3.289 3.292 3.291
3.291 3.295 3.291 3.291 3.292 3.288 3.292 3.293 3.291 3.295
3.292 3.290 3.292 3.289 3.291 3.294 3.291 3.293 3.292 3.288
3.291 3.291 3.290 3.295 3.292 3.291 3.293 3.289 3.290 3.292
3.290 3.294 3.293 3.290 3.292 3.290 3.289 3.293 3.291 3.292
3.294 3.290 3.290 3.291 3.289 3.293 3.293 3.291 3.293 3.290
3.288 3.291 3.290 3.292 3.294 3.290 3.292 3.291 3.288 3.291
3.291 3.291 3.294 3.291 3.290 3.291 3.288 3.291 3.293 3.291
3.293 3.292 3.288 3.291 3.290 3.290 3.294 3.291 3.291 3.292
3.288 3.290 3.291 3.290 3.294 3.293 3.290 3.292 3.289 3.289
3.293 3.290 3.292 3.293 3.289 3.291 3.290 3.289 3.293 3.292
3.291 3.293 3.289 3.289 3.291 3.289 3.292 3.293 3.290 3.292
3.290 3.288 3.292 3.291 3.291 3.294 3.290 3.290 3.291 3.288
3.291 3.292 3.291 3.293 3.291 3.288 3.291 3.289 3.290 3.293
3.290 3.292 3.292 3.288 3.291 3.291 3.290 3.293 3.291 3.290
3.292 3.288 3.289 3.292 3.290 3.292 3.293 3.289 3.291 3.289
3.288 3.293 3.291 3.291 3.292 3.288 3.289 3.290 3.288 3.292
3.293 3.290 3.292 3.289 3.288 3.291 3.290 3.291 3.293 3.289
3.290 3.290 3.287 3.291 3.291 3.290 3.293 3.290 3.288 3.290
3.288 3.290 3.293 3.291 3.292 3.291 3.288 3.290 3.289 3.289
3.293 3.290 3.290 3.291 3.287 3.289 3.291 3.289 3.292 3.291
3.288 3.290 3.288 3.288 3.292 3.290 3.291 3.292 3.288 3.289
3.290 3.288 3.292 3.292 3.290 3.292 3.289 3.288 3.291 3.289
3.291 3.293 3.289 3.291 3.290 3.287 3.291 3.290 3.290 3.293
3.289 3.289 3.290 3.287 3.290 3.292 3.290 3.292 3.290 3.287
3.290 3.289 3.289 3.292 3.290 3.290 3.291 3.287 3.289 3.290
3.289 3.292 3.291 3.289 3.291 3.288
```

etc...

Crédits

Auteur : Richard Karpen

Seattle, Wash

1993 (Nouveau dans la version 3.57 de Csound)

pv_export

pv_export — Convertit un fichier .pvx en fichier texte à séparateur virgule.

Syntaxe

```
pv_export fichier_pv fichier_texte_csv
```

```
csound -U pv_export fichier_pv fichier_texte_csv
```

Initialisation

fichier_pv - Nom du fichier d'entrée .pvx.

fichier_texte_csv - Nom du fichier texte à séparateur virgule de sortie.

L'utilitaire *pv_export* génère un fichier texte à séparateur virgule pour une édition manuelle d'un fichier .pvx produit par l'utilitaire *PVANAL*. Il peut être utilisé en combinaison avec *pv_import* pour produire des données pour le générateur *pvoc*.

Crédits

Auteur : John ffitch

1995

pv_import

pv_import — Convertit un fichier texte à séparateur virgule en un fichier .pvx.

Syntaxe

```
pv_import fichier_texte_csv fichier_pv
```

```
csound -U pv_import fichier_texte_csv fichier_pv
```

Initialisation

fichier_texte_csv - Nom du fichier texte à séparateur virgule en entrée.

fichier_pv - Nom du fichier .pvx de sortie.

L'utilitaire *pv_import* génère un fichier .pvx utilisable avec le générateur *pvoc*. Il peut être utilisé en combinaison avec *pv_export* pour modifier une analyse de son faite par l'utilitaire *PVANAL*.

Crédits

Auteur : John ffitch

1995

sdif2ad

sdif2ad — Convertit des fichiers SDIF en fichiers utilisables par adsynt.

Description

Convertit des fichiers Sound Description Interchange Format (SDIF) dans le format utilisable par l'opcode de Csound *adsyn*. A partir de la version 4.10 de Csound, *sdif2ad* n'est plus disponible que comme un programme autonome pour console Windows et pour DOS.

Syntaxe

```
csound -U sdif2ad [options] fichier_entree fichier_sortie
```

Initialisation

Options :

- *-s*N -- applique le facteur d'échelle d'amplitude N
- *-p*N -- ne garde que les N premiers partiels. Limité à 1024 partiels. Les indices de piste de partiels de la source sont utilisés directement pour sélectionner le stockage interne. Comme ils peuvent avoir des valeurs arbitraires, le maximum de 1024 partiels peut ne pas être réalisé dans tous les cas.
- *-r* -- fichier de données de sortie en octets inversés. L'option octets inversés est là pour faciliter le transfert entre plates-formes, car le format de fichier *adsyn* de Csound n'est pas portable.

Si le nom de fichier passé à *hetro* a l'extension « .sdif », les données seront écrites en format SDIF comme des trames 1TRC de données de synthèse additive. Le programme utilitaire *sdif2ad* peut être utilisé pour convertir tout fichier SDIF contenant un flot de données 1TRC dans le format *adsyn* de Csound. *sdif2ad* permet à l'utilisateur de limiter le nombre de partiels retenus, et d'appliquer un facteur d'échelle d'amplitude. Ceci est souvent nécessaire, car la spécification SDIF, depuis la réalisation de *sdif2ad*, ne nécessite pas que les amplitudes soient dans un intervalle particulier. *sdif2ad* rapporte sur la console l'information sur le fichier, y compris l'intervalle de fréquence.

Les principaux avantage de SDIF sur le format *adsyn*, pour les utilisateurs de Csound, sont que les fichiers SDIF sont totalement portables d'une plate-forme à l'autre (les données sont en « big-endian »), et qu'ils n'ont pas la limite de durée de 32,76 secondes imposée par le format *adsyn* sur 16 bit. Cette limite est nécessairement imposée par *sdif2ad*. Dans le futur, la lecture du format SDIF pourra être incorporée diretement dans *adsyn*, permettant ainsi l'analyse et le traitement de fichiers de n'importe quelle longueur (seulement limitée par la capacité mémoire du système).

Les utilisateurs doivent se souvenir que les formats SDIF sont toujours en développement. Bien que le format 1TRC soit maintenant bien établi, il peut encore changer.

Pour des informations détaillées sur le Sound Description Interchange Format, se référer au site web du CNMAT : *http://cnmat.CNMAT.Berkeley.EDU/SDIF*

D'autres ressources SDIF (y compris un visionneur) sont disponibles via le site web de NC_DREAM : *http://www.bath.ac.uk/~masjpf/NCD/dreamhome.html*

Crédits

Auteur : Richard Dobson

Somerset, England

Août 2000

Nouveau dans la version 4.08 de Csound

srconv

srconv — Convertit le taux d'échantillonnage d'un fichier audio.

Description

Convertit le taux d'échantillonnage d'un fichier audio de *Rin* à *Rout*. Optionnellement le rapport (*Rin* / *Rout* peut varier linéairement dans le temps selon un ensemble de paires (temps, rapport) dans un fichier auxiliaire.

Syntaxe

```
srconv [options] fichier_entree
```

Initialisation

Options :

- *-P num* = rapport de transposition en hauteur (*srate* / *r*) [ne pas spécifier à la fois *P* et *r*]
- *-Q num* = facteur de qualité (1, 2, 3 ou 4 : par défaut = 2)
- *-i nomfic* = fichier auxiliaire de points charnière (pas de point charnière par défaut, c'est-à-dire pas de changement de rapport)
- *-r num* = taux d'échantillonnage en sortie (doit être spécifié)
- *-o nomfic* = nom du fichier son de sortie
- *-A* = crée un fichier son de sortie au format AIFF
- *-J* = crée un fichier son de sortie au format IRCAM
- *-W* = crée un fichier son de sortie au format WAV
- *-h* = pas d'en-tête dans le fichier son de sortie
- *-c* = échantillons en caractères signés sur 8 bit
- *-a* = échantillons alaw
- *-8* = échantillons en caractères non-signés sur 8 bit
- *-u* = échantillons ulaw
- *-s* = échantillons en entiers courts
- *-l* = échantillons en entiers longs
- *-f* = échantillons en virgule flottante
- *-r N* = remplace le *srate* de l'orchestre
- *-K* = ne génère pas de bloc de pics d'amplitude
- *-R* = réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF)

- *-H#* = imprime une pulsation dans le style 1, 2 ou 3 à chaque écriture dans le fichier son
- *-N* = notification (cloche système) quand le traitement est fini
- -- *nomfic* = compte-rendu dans un fichier

Ce programme effectue une conversion arbitraire du taux d'échantillonnage en haute fidélité. La méthode consiste à parcourir le fichier d'entrée avec un pas d'incrémentation conforme au taux d'échantillonnage désiré, et de calculer les points de sortie comme moyennes convenablement pondérées des points voisins. Il y a deux cas à considérer :

1. les taux d'échantillonnage sont dans un petit rapport entier - les poids sont obtenus de la table
2. les taux d'échantillonnage sont dans un grand rapport entier - les poids sont linéairement interpolés de la table.

Calcul de l'incrément : pour une décimation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage en sortie ; pour une interpolation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage de l'entrée.

Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitch

30 décembre 2000

Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MAKECSD, MIXER, SCALE)

Les divers utilitaires suivants sont disponibles :

- *CS* : Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.
- *CSB64ENC* : Convertit un fichier binaire en un fichier texte encodé en Base64.
- *ENVEXT* : Extrait l'enveloppe d'un fichier vers une liste textuelle.
- *EXTRACTOR* : Extrait une section audio d'un fichier audio.
- *MAKECSD* : Crée un fichier CSD à partir des fichiers d'entrée spécifiés.
- *MIXER* : Mélange ensemble plusieurs fichiers son.
- *SCALE* : Calibre l'amplitude d'un fichier son.

cs

cs — Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

Description

Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

Syntaxe

```
cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]
```

Initialisation

Drapeaux :

- - *OPTIONS* = OPTIONS est une séquence de caractères alphabétiques qui peut être utilisée pour sélectionner l'exécutable Csound à lancer, aussi bien que les options de ligne de commande (voir ci-dessous). L'option 'r' est une valeur par défaut (sélection de la sortie en temps-réel), mais on peut la remplacer.
- *<nom>* = c'est la racine de nom de fichier pour sélectionner les fichiers arguments ; elle peut contenir un chemin. Les fichiers qui ont pour extension `.csd`, `.orc`, ou `.sco` sont recherchés, et soit un CSD soit une paire orc/sco qui correspond à <nom>, le meilleur des deux, est sélectionné. Des fichiers MIDI avec une extension `.mid` sont aussi recherchés, et si l'un deux correspond à <nom> au moins autant que le CSD ou la paire orc/sco, il est utilisé avec l'option -F.

NOTE

Le fichier MIDI n'est pas utilisé si une option -M ou -F est spécifiée par l'utilisateur (nouveau dans la version 4.24.0). A moins qu'il y ait une option (-n ou -o) relative à la sortie audio, un nom de fichier de sortie avec l'extension appropriée est généré automatiquement (basé sur le nom des fichiers d'entrée sélectionnés et sur les options de format). Le fichier de sortie est toujours écrit dans le répertoire courant.

NOTE

les extensions de nom de fichier ne sont pas sensibles à la casse.

- *[OPTIONS DE CSOUND ...]* = n'importe quel nombre d'options supplémentaires pour Csound qui sont simplement copiées dans la ligne de commande finale qui sera exécutée.

La ligne de commande qui est exécutée est générée à partir de quatre origines :

1. L'exécutable de Csound (éventuellement avec options). Une seule possibilité est choisie parmi les trois qui suivent (la dernière à la plus haute priorité) :

   - une valeur par défaut

- la valeur d'une variable d'environnement de CSOUND
- des variables d'environnement avec un nom de la forme CSOUND_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcba est utilisée, et si les variables d'environnement CSOUND_B et CSOUND_C sont définies, la valeur de CSOUND_B sera effective.

2. N'importe quel nombre de listes d'option, ajoutées dans l'ordre suivant :

   - soit quelques valeurs par défaut, soit la valeur de la variable d'environnement CSFLAGS si elle est définie.
   - des variables d'environnement avec un nom de la forme CSFLAGS_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcba est utilisée, et si les variables d'environnement CSFLAGS_A et CSFLAGS_C sont définies par '-M 1 -o dac' et '-m231 -H0', respectivement, la chaîne '-m231 -H0 -M 1 -o dac' sera ajoutée.

3. Les options explicites de [OPTIONS DE CSOUND ...].

4. Toutes les options et les noms de fichiers générés à partir de <nom>.

NOTE

Les options entre apostrophes qui contiennent des espaces sont autorisées.

Exemples

Avec les variables d'environnement suivantes :

```
CSOUND    = csoundfltk.exe -W
CSOUND_D  = csound64.exe -J
CSOUND_R  = csoundfltk.exe -h

CSFLAGS   = -d -m135 -H1 -s
CSFLAGS_D = -f
CSFLAGS_R = -m0 -H0 -o dac1 -M "MIDI Yoke NT:  1" -b 200 -B 6000
```

Et un répertoire qui contient :

```
foo.orc                 piano.csd
foo.sco                 piano.mid
im.csd                  piano2.mid
ImproSculpt2_share.csd  foobar.csd
```

Les commandes suivantes s'exécuteront comme il est montré :

```
cs foo           => csoundfltk.exe -W -d -m135 -H1 -s -o foo.wav  \
foo.orc foo.sco

cs foob          => csoundfltk.exe -W -d -m135 -H1 -s             \
-o foobar.wav foobar.csd

cs -r imp -i adc => csoundfltk.exe -h -d -m135 -H1 -s -m0 -H0     \
-o dac1 -M "MIDI Yoke NT:  1"  \
-b 200 -B 6000 -i adc          \
ImproSculpt2_share.csd

cs -d im         => csound64.exe -J -d -m135 -H1 -s -f -o im.sf   \
im.csd

cs piano         => csoundfltk.exe -W -d -m135 -H1 -s             \
-F piano.mid -o piano.wav      \
```

```
piano.csd

cs piano2        => csoundfltk.exe -W -d -m135 -H1 -s              \
-F piano2.mid -o piano2.wav    \
piano.csd
```

Crédits

Auteur : Istvan Varga

Janvier 2003

csb64enc

csb64enc — Convertit un fichier binaire en un fichier texte encodé en Base64.

Description

L'utilitaire *csb64enc* génère un fichier texte encodé en Base64 à partir d'un fichier binaire, tel qu'un fichier MIDI standard (.mid) ou n'importe quel type de fichier audio. Il est utile pour convertir un fichier dans le format accepté par la section *<CsFileB>* d'un fichier csd, pour y inclure le fichier converti.

Syntaxe

```
csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

Initialisation

Options :

- - *w n* = fixe la largeur de ligne du fichier de sortie à n (par défaut : 72)
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)

Exemples

```
csb64enc -w 78 -o fichier.txt fichier.mid
```

La commande produit un fichier texte encodé en Base64 à partir d'un fichier MIDI standard, *fichier.mid*. Ce fichier peut maintenant être collé dans la section *<CsFileB>* d'un fichier csd.

Voir Aussi

makecsd

Crédits

Auteur : Istvan Varga

Janvier 2003

envext

envext — Extrait l'enveloppe d'un fichier son vers un fichier texte.

Syntaxe

```
envext [-options] fichierson

csound -U envext [-options] fichierson
```

Initialisation

fichierson - Nom du fichier son en entrée.

Les options suivantes sont disponibles pour *envext*. (Les valeurs par défaut sont mises entre parenthèses) :

-o nomfic Nom du fichier de sortie (newenv)
-w taille (en secondes) de la fenêtre d'analyse (0.25)

L'utilitaire *envext* génère un fichier texte contenant des paires de temps et d'amplitude en trouvant les pics absolus dans chaque fenêtre.

Exemple

En tapant la commande (depuis le répertoire manual-fr) :

```
csound -U envext examples/mary.wav
```

on obtiendra un fichier texte contenant :

```
0.000   0.000
0.000   0.000
0.250   0.000
0.500   0.000
0.750   0.000
1.249   0.170
1.499   0.269
1.530   0.307
1.872   0.263
2.056   0.304
2.294   0.241
2.570   0.216
2.761   0.178
3.077   0.011
3.251   0.001
3.500   0.000
```

qui montre le temps pour le pic d'amplitude dans chaque fenêtre mesurée.

Crédits

Auteur : John ffitch

1995

extractor

extractor — Extrait une section audio d'un fichier audio.

Description

Extrait une section audio, par temps ou échantillon, d'un fichier son existant.

Syntaxe

```
extractor [OPTIONS ... ] fichierentree
```

Initialisation

Options :

- -*S entier* = Démarre l'extraction à l'échantillon dont le numéro est donné.
- -*Z entier* = Termine l'extraction à l'échantillon dont le numéro est donné.
- - *Q entier* = Extrait le nombre donné d'échantillons.
- -*T fpnum* = Démarre l'extraction au temps donné en secondes.
- -*E fpnum* = Termine l'extraction au temps donné en secondes.
- -*D fpnum* = Extrait la durée donnée en secondes.
- -*R* = Réécrit continuellement l'en-tête lors de l'écriture du fichier son (WAV/AIFF).
- -*H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- -*N* = Signal d'alerte (habituellement la cloche système) à la fin.
- -*v* = Mode verbeux.
- -*o nomfic* = Nom du fichier de sortie (par défaut : test.wav)

Exemples

Les valeurs par défaut sont :

```
extractor -S 0 -Z fin-du-fichier -o test
```

Par exemple

```
extractor -S 10234 -D 2.13 in.aiff -o out.wav
```

Cela crée un nouveau fichier son extrait à partie de l'échantillon 10234 et durant 2,13 secondes.

Crédits

Auteur : John ffitch

1994

makecsd

makecsd — Crée un fichier CSD à partir des fichiers spécifiés en entrée.

Description

Crée un fichier CSD à partir des fichiers spécifiés en entrée. Le premier fichier d'entrée qui a une extension .orc (la casse n'est pas significative) est mis dans la section <CsInstruments>, et le premier fichier d'entrée qui a une extension .sco devient <CsScore>. Tous les fichiers restants sont encodés en Base64 et ajoutés dans des balises <CsFileB>. Une section <CsOptions> vide est toujours ajoutée.

Un filtrage du texte est effectué sur les fichiers d'orchestre et de partition :

- les caractères de nouvelle ligne sont convertis dans le format natif du système sur lequel *makecsd* est exécuté.
- les lignes vides sont enlevées du début et de la fin des fichiers.
- tous les espaces restant en fin de ligne sont supprimés.
- facultativement, les tabulations peuvent être développées en espaces avec une taille de tabulation spécifiée par l'utilisateur.

Syntaxe

```
makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

Initialisation

Options :

- - *t n* = développe les tabulations en espaces en utilisant une taille de tabulation égale à n (désactivé par défaut). Ceci s'applique seulement à l'orchestre et à la partition.
- - *w n* = fixe la largeur de ligne Base64 à n (par défaut : 72). Note : l'orchestre et la partition ne sont pas concernés.
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)

Exemples

```
makecsd -t 6 -w 78 -o fichier.csd fichier.mid fichier.orc fichier.sco sample.aif
```

Crée un fichier CSD à partir de fichier.orc et de fichier.sco (les tabulations sont développées en espaces sachant qu'une tabulation vaut 6 caractères), et fichier.mid et sample.aif sont ajoutés dans des balises <CsFileB> contenant les données encodées en Base64 avec une largeur de ligne de 78 caractères. Le fichier de sortie est fichier.csd.

Crédits

Auteur : Istvan Varga

Janvier 2003

mixer

mixer — Mélange ensemble plusieurs fichiers son.

Description

Mélange ensemble plusieurs fichiers son, démarrant à des temps différents et avec une sélection individuelle des canaux dans les fichiers d'entrée.

Syntaxe

```
mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...
```

Initialisation

Options :

- *-A* = Génère un fichier de sortie en AIFF.
- *-W* = Génère un fichier de sortie en WAV.
- *-h* = Génère un fichier de sortie sans en-tête.
- *-c* = Génère des échantillons en caractères signés sur 8 bit.
- *-a* = Génère des échantillons alaw.
- *-u* = Génère des échantillons ulaw.
- *-s* = Génère des échantillons en entiers courts.
- *-l* = Génère des échantillons en entiers longs (32 bit).
- *-f* = Génère des échantillons en virgule flottante.
- *-F arg* = Spécifie le gain à appliquer au fichier d'entrée qui suit. Si arg est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- *-S entier* = Indique à partir de quel échantillon commencer le mixage dans le fichier d'entrée suivant.
- *-T fpnum* = Indique à quel date (en secondes) commencer le mixage dans le fichier d'entrée suivant.
- *-1* = Mixer le canal 1 du fichier son suivant.
- *-2* = Mixer le canal 2 du fichier son suivant.
- *-3* = Mixer le canal 3 du fichier son suivant.
- *-4* = Mixer le canal 4 du fichier son suivant.
- -^ *entx enty* = Mixer le canal x du fichier son suivant vers le canal y dans le fichier de sortie.
- *-v* = Mode verbeux.
- *-R* = Réécrit continuellement l'en-tête lors des opérations d'écriture du fichier son (WAV/AIFF)

- -*H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- -*N* = Alerte (habituellement la cloche système) lorsque le mixage est fini.
- -*o nomfic* = nom du fichier de sortie (par défaut : test.wav)

Exemples

Les valeurs par défaut sont :

```
mixer -s -otest -F 1.0 -S 0
```

Par exemple

```
mixer -F 0.96 in1.wav -S 300 -2 in2.aiff -S 300 -^4 1 in3.wav -o out.wav
```

Cela crée un nouveau fichier son avec un gain constant de 0,96 pour in1.wav, le second canal de in2.aiff mixé après 300 échantillons et le canal 4 de in3.wav sorti comme le canal 1 après 300 échantillons.

Crédits

Auteur : John ffitch

1994

scale

scale — Calibre l'amplitude d'un fichier son.

Description

Prend un fichier son et le calibre en appliquant un gain, constant ou variable. L'échelle peut être comme un multiplicateur, un maximum ou un pourcentage de 0dB.

Syntaxe

```
scale [OPTIONS ... ] fichier
```

Initialisation

Options :

- *-A* = Génère un fichier de sortie AIFF.
- *-W* = Génère un fichier de sortie WAV.
- *-h* = Génère un fichier de sortie sans en-tête.
- *-c* = Génère des échantillons en caractères signés sur 8 bit.
- *-a* = Génère des échantillons alaw.
- *-u* = Génère des échantillons ulaw.
- *-s* = Génère des échantillons en entiers courts.
- *-l* = Génère des échantillons en entiers longs (32 bit)
- *-f* = Génère des échantillons en virgule flottante.
- *-F arg* = Spécifie le gain à appliquer. Si *arg* est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- *-M fpnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit la valeur donnée.
- *-P fpnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit le pourcentage donné de 0dB.
- *-R* = Réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Alerte (habituellement la cloche système) lorsque c'est fini.
- *-o nomfic* = nom du fichier de sortie (par défaut : test.wav)

Exemples

```
scale -s -W -F 0.96 -o out.wav sound.wav
```

Ceci crée un nouveau fichier son avec un gain constant de 0,96. C'est particulièrement utile si le fichier d'entrée est en format virgule flottante.

Crédits

Auteur : John ffitch

1994

Crédits

Dan Ellis

MIT Media Lab

Cambridge, Massachussetts

Cscore

Cscore est une API (interface de programmation d'application) pour générer et manipuler des fichiers de partition numérique. Elle fait partie de l'API plus grande de Csound et elle comprend un certain nombre de fonctions appelables depuis un programme écrit par l'utilisateur en langage C. *Cscore* peut étre invoquée comme un préprocesseur de partition autonome ou comme élément d'une exécution de Csound en incluant l'option -C dans ses arguments :

`cscore` [*fichier_partition_entree*] [> *fichier_partition_sortie*]

(où *cscore* est le nom du programme que vous avez écrit), ou

`csound` [-C] [`autresoptions`] [*nomorch*] [*nompartition*]

Les fonctions de l'API disponibles augmentent la bibliothèque de fonctions du langage C ; elles peuvent lire des fichiers de partition numérique standard ou pré-triée, modifier et étendre les données de différentes manières, et ensuite les rendre disponibles pour une exécution par un orchestre de Csound.

Le programme écrit par l'utilisateur dans le langage C est compilé et lié à la bibliothèque de Csound (ou au programme de ligne de commande csound) par l'utilisateur. Il n'est pas indispensable de bien connaître le langage C pour écrire ce programme, car les appels de fonction ont une syntaxe simple, et sont suffisamment puissants pour faire la plus grande partie du travail compliqué. C pourra apporter plus de puissance par la suite selon les besoins.

Les sections suivantes expliquent toutes les étapes de l'utilisation de *Cscore* :

- *Evènements, Listes et Opérations* - Explique la syntaxe des fonctions de *Cscore* et les structures de données.
- *Ecrire un Programme de Contrôle Cscore* - Montre par l'exemple comment écrire votre propre programme de contrôle.
- *Compiler un Programme Cscore* - Décrit les étapes de la compilation et de l'édition des liens avec la bibliothèque de Csound.
- *Exemples Plus Avancés* - Traite de questions avancées comme plusieurs partitions en entrée et les détails de l'exécution de *Cscore* au sein d'une exécution de Csound.

Evénements, Listes et Opérations

Un évènement dans *Cscore* est équivalent à une instruction d'une *partition numérique standard* ou d'une partition résolue en temps (le format dans lequel Csound écrit une partition triée -- consultez n'importe quel fichier *score.srt*), et il est stocké en interne en format de temps résolu. Il est important de noter que lorsque *Cscore* est utilisé en mode autonome, il est incapable de comprendre les « commodités » non numériques que Csound permet dans le format de partition en entrée. C'est pourquoi, les partitions utilisant des fonctionnalités telles que le report (carry), les rampes, les expressions et autres devront être triées au préalable avec l'utilitaire *scsort* ou bien utilisées avec un exécutable *Csound* modifié contenant le programme *Cscore* de l'utilisateur. Les opcodes de partition avec des argument macros (r, m, n, and {}) ne sont pas interprétés.

Les évènements de partition sont lus à partir d'un fichier de partition existant et stockés chacun dans une structure C. Les principaux composants de ces structures sont un opcode et un tableau de valeurs de p-

champs. *Cscore* gère la lecture des évènements et leur mise en mémoire pour vous. Le format de la structure commence comme suit :

```
typedef struct {
     CSHDR h;        /* en-tête pour la gestion de l'espace */
     char  *strarg;  /* adresse d'un argument chaîne faculatif */
     char  op;       /* opcode-t, w, f, i, a, s ou e */
     short pcnt;
     MYFLT p2orig;   /* p2, p3 non résolus */
     MYFLT p3orig;
     MYFLT p[1];     /* tableau des p-champs p0, p1, p2 ... */
} EVENT;
```

MYFLT est l'un des types C *float* ou *double* selon la manière dont votre copie de la bibliothèque de Csound a été compilée. Vous avez juste à déclarer les variables en virgule flottante de votre programme avec le type MYFLT pour être compatible.

Toute fonction de *Cscore* qui crée, lit ou copie un évènement retournera un pointeur sur la structure dans laquelle les données de l'évènement sont stockées. Ce pointeur d'évènement peut être utilisé pour accéder aux composants de la structure, de la forme *e->op* ou *e->p[n]*. Chaque évènement nouvellement stocké provoquera la création d'un nouveau pointeur, et une séquence de nouveaux évènements générera une séquence de pointeurs distincts qu'il faudra stocker. Les groupes de pointeurs d'évènement sont stockés dans une liste d'évènements qui a sa propre structure :

```
typedef struct {
     CSHDR  h;
     int nslots;    /* nombre maximal d'évènements dans cette liste */
     int nevents;   /* nombre d'évènements présents */
     EVENT *e[1];   /* tableau de pointeurs d'évènement e0, e1, e2.. */
} EVLIST;
```

Toute fonction qui crée ou modifie une liste retournera un pointeur sur la nouvelle liste. Ce pointeur de liste peut être utilisé pour accéder à ses composants pointeurs d'évènement, de la forme *a->e[n]*. Les pointeurs d'évènement et les pointeurs de liste sont ainsi les outils de base pour manipuler les données d'un fichier de partition. Les pointeurs et les listes de pointeurs peuvent être copiés et réordonnés sans modifier les valeurs des données auxquelles ils font référence. Cela signifie que l'on peut copier et manipuler les notes et les phrases depuis un niveau de contrôle élevé. Alternativement, les données d'un évènement ou d'un groupe d'évènements peuvent être modifiées sans changer les pointeurs d'évènement ou de liste. Les fonctions de l'API *Cscore* permettent de créer et de manipuler des partitions de cette manière.

Avec Csound 5, les noms de toutes les fonctions de l'API *Cscore* ont été changés pour être plus explicites. De plus, chaque fonction nécessite maintenant un pointeur sur un objet CSOUND en premier argument. La structure de l'objet CSOUND n'a pas d'importance (en fait il ne peut pas être modifié dans un programme utilisateur). Le moyen d'obtenir ce pointeur sur un objet CSOUND sera montré dans la section suivante. Les fonctions de *Cscore* et ses structures de données sont déclarées dans le fichier d'en-tête `cscore.h` que vous devez inclure dans le code de votre programme avant leur utilisation.

Les noms des fonctions de *Cscore* spécifient si elles opèrent sur des évènements ou sur des listes d'évènements. Dans le sommaire suivant des appels de fonction disponibles, on utilise quelques conventions de nommage :

```
Le symbole cs est un pointeur vers un objet CSOUND (CSOUND *);
Les symboles e, f sont des pointeurs sur des évènements (notes);
Les symboles a, b sont des pointeurs sur des listes (arrays) de tels évènements;
Le symbole n est un paramètre entier de type int;
"..." indique un paramètre chaîne (soit une constante soit une variable de type char *);
Le symbole fp est un pointeur sur un fichier (FILE *) en flot d'entrée de partition;

syntaxe d'appel                                 description
---------------                                 -----------
/* Fonctions pour travailler avec des évènements */
e = cscoreCreateEvent(cs, n);                   crée un évènement vide avec n pchamps
e = cscoreDefineEvent(cs, "...");               définit un évènement par la chaîne de caractères ...
e = cscoreCopyEvent(cs, f);                     fait une nouvelle copie de l'évènement f
e = cscoreGetEvent(cs);                         lit l'évènement suivant dans le fichier de partition en
    cscorePutEvent(cs, e);                      écrit l'évènement e dans le fichier de partition en so
    cscorePutString(cs, "...");                 écrit l'évènement défini par la chaîne dans la partitio
                                                  en sortie

/* Fonctions pour travailler avec des listes d'évènements */
a = cscoreListCreate(cs, n);                    crée une liste d'évènements vide avec n emplacements
a = cscoreListAppendEvent(cs, a, e);            ajoute l'évènement e à la fin de la liste a
a = cscoreListAppendStringEvent(cs, a, "...");  ajoute l'évènement défini par la chaîne à la liste a
a = cscoreListCopy(cs, b);                      copie la liste b (mais pas les évènements)
a = cscoreListCopyEvents(cs, b);                copie les évènements de b, en créant une nouvelle liste
a = cscoreListGetSection(cs);                   lit tous les évènements de la partition en entrée, jusq
                                                  prochain s ou e
a = cscoreListGetNext(cs, nbeats);              lit les prochaines nbeats pulsations de la partition en
                                                  (nbeats est un MYFLT)
a = cscoreListGetUntil(cs, beatno);             lit tous les évènements de la partition en entrée jusqu
                                                  pulsation beatno (MYFLT)
a = cscoreListSeparateF(cs, b);                 sépare les instructions f de la liste b vers la liste a
a = cscoreListSeparateTWF(cs, b);               sépare les instructions t,w & f de la liste b vers la l
a = cscoreListAppendList(cs, a, b);             ajoute la liste b à la liste a
a = cscoreListConcatenate(cs, a, b);            concaténation des listes a et b (identique au précédent
    cscoreListSort(cs, a);                      trie la liste a en ordre chronologique selon p[2]
n = cscoreListCount(cs, a);                     retourne le nombre d'évènements dans la liste a
a = cscoreListExtractInstruments(cs, b, "..."); extrait les notes des instruments ... (pas de nouveaux
                                                  évènements)
a = cscoreListExtractTime(cs, b, from, to);     extrait les notes d'une période de temps, en créant de
                                                  nouveaux évènements (from et to sont des MYFLT)
    cscoreListPut(cs, a);                       écrit les évènements de la liste a dans le fichier de p
                                                  sortie
    cscoreListPlay(cs, a);                      envoie les évènements de la liste a vers l'orchestre de
                                                une exécution immédiate (ou les imprime s'il n'y a pas

/* Fonctions pour réclamer de la mémoire */
    cscoreFreeEvent(cs, e);                     libère l'espace de l'évènement e
    cscoreListFree(cs, a);                      libère l'espace de la liste a (mais pas les évènements)
    cscoreListFreeEvents(cs, a);                libère les évènements de la liste a, et l'espace de la

/* Fonctions pour travailler avec plusieurs fichiers de partition en entrée */
fp = cscoreFileGetCurrent(cs);                  récupère le pointeur du fichier de partition en entrée
                                                  actif (au départ trouve le pointeur du fichier de par
                                                  entrée de la ligne de commande)
fp = cscoreFileOpen(cs, "filename");            ouvre un autre fichier de partition en entrée (5 au max
     cscoreFileSetCurrent(cs, fp);              fait de fp le pointeur sur le fichier de partition
                                                  actuellement actif
     cscoreFileClose(cs, fp);                   ferme le fichier de partition en relation avec FILE *fp
```

Sous Csound 4, les noms des fonctions et leurs paramètres étaient les suivants :

```
  syntaxe d'appel                          description
  ---------------                          -----------
  e = createv(n);         crée un évènement vide avec n pchamps
  e = defev("...");       définit un évènement par la chaîne de caractères ...
  e = copyev(f);          fait une nouvelle copie de l'évènement f
  e = getev();            lit l'évènement suivant dans le fichier de partition en entrée
  putev(e);               écrit l'évènement e dans le fichier de partition en sortie
  putstr("...");          écrit l'évènement défini par la chaîne dans la partition en sortie
  a = lcreat(n);          crée une liste d'évènements vide avec n emplacements
       int  n;
  a = lappev(a,e);        ajoute l'évènement e à la fin de la liste a
  a = lappstrev(a,"..."); ajoute l'évènement défini par la chaîne à la liste a
```

```
  a = lcopy(b);            copie la liste b (mais pas les évènements)
  a = lcopyev(b);          copie les évènements de b, en créant une nouvelle liste
  a = lget();              lit tous les évènements de la partition en entrée, jusqu'au
                             prochain s ou e
  a = lgetnext(nbeats);    lit les prochaines nbeats pulsations de la partition en entrée
       float  nbeats;
  a = lgetuntil(beatno);   lit tous les évènements de la partition en entrée jusqu'à la
                             pulsation beatno
       float  beatno;
  a = lsepf(b);            sépare les instructions f de la liste b vers la liste a
  a = lseptwf(b);          sépare les instructions t,w & f de la liste b vers la liste a
  a = lcat(a,b);           concaténation (ajout) de la liste b à la liste a
  lsort(a);                trie la liste a en ordre chronologique selon p[2]
  a = lxins(b,"...");      extrait les notes des instruments ... (pas de nouveaux évènements)
  a = lxtimev(b,from,to);  extrait les notes d'une période de temps, en créant de nouveaux
       float  from, to;      évènements
  lput(a);                 écrit les évènements de la liste a dans le fichier de partition en sortie
  lplay(a);                envoie les évènements de la liste a vers l'orchestre de Csound pour
                           une exécution immédiate (ou les imprime s'il n'y a pas d'orchestre)
  relev(e);                libère l'espace de l'évènement e
  lrel(a);                 libère l'espace de la liste a (mais pas les évènements)
  lrelev(a);               libère les évènements de la liste a, et l'espace de la liste
  fp = getcurfp();         récupère le pointeur du fichier de partition en entrée actuellement
                           actif (au départ trouve le pointeur du fichier de partition en entrée
                           de la ligne de commande)
  fp = filopen("filename"); ouvre un autre fichier de partition en entrée (5 au maximum)
  setcurfp(fp);            fait de fp le pointeur sur le fichier de partition actuellement actif
  filclose(fp);            ferme le fichier de partition en relation avec FILE *fp
```

Ecrire un Programme de Contrôle Cscore

Le format général d'un programme de contrôle *Cscore* est :

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
 /*  DECLARATIONS DES VARIABLES  */
 /*  CORPS DU PROGRAMME  */
}
```

L'instruction *include* définira les structures d'évènement et de liste et toutes les fonctions de l'API *Cscore* pour le programme. Il faut que le nom de la fonction de l'utilisateur soit *cscore* si elle doit être liée avec le programme *main* standard dans `cscormai.c` ou liée comme routine *Cscore* interne pour un exécutable de Csound personnalisé. Cette fonction *cscore()* reçoit un argument de `cscormai.c` ou de Csound -- *CSOUND *cs* -- qui est un pointeur sur un objet Csound. Le pointeur *cs* doit être passé en premier paramètre à toutes les fonctions de l'API *Cscore* que le programme appelle.

Le programme C suivant lira depuis une *partition numérique standard*, jusqu'à (mais sans l'inclure) la première *instruction s* ou *e*, puis il écrira ces données (inchangées) en sortie.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
  EVLIST *a;                    /* a est autorisé à pointer sur une liste d'évènements */
  a = cscoreListGetSection(cs); /* lit les évènements, retourne le pointeur de liste */
  cscoreListPut(cs, a);         /* écrit ces évènements en sortie (inchangés) */
  cscorePutString(cs, "e");     /* écrit la chaîne e sur la sortie */
}
```

Aprés l'exécution de *cscoreListGetSection()*, la variable *a* pointe sur une liste d'adresses d'évènements, qui pointent chacune sur un évènement stocké. Nous avons utilisé ce même pointeur pour permettre à une autre fonction de liste -- *cscoreListPut()* -- d'accéder à tous les évènements qui ont été lus et de les écrire en sortie. Si nous définissons maintenant un autre symbole *e* comme pointeur d'évènement, alors l'instruction

```
e = a->e[4];
```

lui affectera le contenu du 4ème emplacement de la structure *EVLIST*, *a*. Ce contenu est un pointeur sur un évènement, qui comprend lui-même un tableau de valeurs de champs de paramètre. Ainsi le terme *e->p[5]* signifiera la valeur du champ de paramètre 5 du 4ème évènement dans la *EVLIST* dénotée par *a*. Le programme ci-dessous multipliera la valeur de ce *p-champ* par 2 avant de l'écrire en sortie.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
  EVENT  *e;                    /* un pointeur sur un évènement */
  EVLIST *a;
  a = cscoreListGetSection(cs); /* lit une partition comme une liste d'évènements */
  e = a->e[4];                  /* pointe sur l'évènement 4 dans la liste a */
  e->p[5] *= 2;                 /* trouve le p-champ 5, multiplie sa valeur par 2 */
  cscoreListPut(cs, a);         /* écrit en sortie la liste d'évènements */
  cscorePutString(cs, "e");     /* ajoute une instruction de "fin de partition" */
}
```

Considérez maintenant la partition suivante, dans laquelle *p[5]* contient la fréquence en Hz.

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
e
```

Si cette partition est donnée au programme principal précédent, la sortie résultante ressemblera à ceci :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 512 10000        ; p[5] est devenu 512 au lieu de 256.
i 1 7 3 0 880 10000
e
```

Notez que le 4ème évènement est en fait la seconde note de la partition. Jusqu'ici nous n'avons pas fait de distinction entre les notes et les tables de fonction mises en place dans une partition numérique. Les deux peuvent être classées comme évènement. Notez aussi que notre 4ème évènement a été stocké dans le champ *e[4]* de la structure. Pour être compatible avec la notation des *p-champs* de Csound, nous ignorerons *p[0]* et *e[0]* dans les structures d'évènement et de liste, en stockant *p1* dans *p[1]*, l'évènement 1 dans *e[1]*, etc. Les fonctions de *Cscore* adoptent toutes cette convention.

Pour étendre l'exemple ci-dessus, nous pourrions décider d'utiliser les mêmes pointeurs *a* et *e* pour examiner chacun des évènements dans la liste. Noter que *e* n'a pas été fixé au nombre 4, mais au contenu du 4ème emplacement de la liste. Pour inspecter le *p5* de l'évènement précédent dans la liste, nous n'avons

qu'à redéfinir *e* avec l'affectation

```
e = a->e[3];
```

et référencer le 5ème emplacement du tableau de *p-champs* avec l'expression

```
e->p[5]
```

Plus généralement, nous pouvons utiliser une variable entière comme indice du tableau *e[]*, et accéder séquentiellement à chaque évènement en utilisant une boucle et en incrémentant l'indice. Le nombre d'évènements stockés dans une *EVLIST* est contenu dans le membre *nevents* de la structure.

```
int index;    /* démarre avec e[1] car e[0] n'est pas utilisé */
for (index = 1; index <= a->nevents; index++)
{
  e = a->e[index];
  /* faire quelque chose avec e */
}
```

L'exemple ci-dessus démarre avec *e[1]* et augmente l'indice à chaque passage dans la boucle (*index++*) jusqu'à ce qu'il soit plus grand que *a->nevents*, l'indice du dernier évènement dans la liste. Les instructions à l'intérieur de la boucle *for* sont exécutées une dernière fois quand *index* égale *a->nevents*.

Dans le programme suivant nous utiliserons la même partition en entrée. Cette fois nous séparerons les instructions de *ftable* des instructions de *note*. Nous écrirons ensuite en sortie les trois évènements de note stockés dans la liste *a*, puis nous créerons une seconde section de partition constituée de l'ensemble de hauteurs original et d'une version transposée de celui-ci. Cela apportera un doublement à l'octave.

Ici, notre indice dans le tableau est *n* et il est incrémenté dans un bloc *for* qui boucle *nevents* fois, ce qui permet d'appliquer une instruction au même *p-champ* des évènements successifs.

```
#include  "cscore.h"
void cscore(CSOUND *cs)
{
  EVENT  *e, *f;
  EVLIST *a, *b;
  int n;

  a = cscoreListGetSection(cs);            /* lit la partition dans la liste d'évènements "a" */
  b = cscoreListSeparateF(cs, a);          /* sépare les instructions f */
  cscoreListPut(cs, b);                    /* écrit les instructions f dans la partition en sortie */
  e = cscoreDefineEvent(cs, "t 0 120");    /* définit un évènement pour l'instruction de tempo */
  cscorePutEvent(cs, e);                   /* écrit l'instruction de tempo dans la partition */
  cscoreListPut(cs, a);                    /* écrit les notes */
  cscorePutString(cs, "s");                /* fin de section */
  cscorePutEvent(cs, e);                   /* écrit l'instruction de tempo encore une fois */
  b = cscoreListCopyEvents(cs, a);         /* fait une copie des notes dans "a" */
  for (n = 1; n <= b->nevents; n++)        /* répète les lignes suivantes nevents fois : */
  {
    f = b->e[n];
    f->p[5] *= 0.5;                        /* transpose la hauteur d'une octave vers le bas */
  }
  a = cscoreListAppendList(cs, a, b);      /* ajoute ces notes aux hauteurs originales */
  cscoreListPut(cs, a);
  cscorePutString(cs, "e");
}
```

La sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000
i 1 4 3 0 128 10000
i 1 7 3 0 440 10000
e
```

Si la sortie est écrite dans un fichier, le fait que les évènements ne soient pas ordonnés n'est pas un problème. La sortie est écrite dans un fichier (ou sur la sortie standard) chaque fois que la fonction *cscoreListPut()* est utilisée. Cependant, si ce programme était appelé durant une exécution de Csound et que la fonction *cscoreListPlay()* était remplacée par *cscoreListPut()*, alors les évènements seraient envoyés à l'orchestre au lieu du fichier et il faudrait qu'ils soient préalablement triés en appelant la fonction *cscoreListSort()*. Les détails de la sortie de la partition et de son exécution quand on utilise *Cscore* depuis Csound sont décrits dans la section suivante.

Ensuite nous étendons le programme ci-dessus en utilisant la boucle *for* pour lire *p[5]* et *p[6]*. Dans la partition originale *p[6]* dénote l'amplitude. Pour créer un diminuendo sur l'octave inférieure ajoutée, qui soit indépendant de l'ensemble de notes original, une variable appelée *dim* sera utilisée.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
  EVENT  *e, *f;
  EVLIST *a, *b;
  int n, dim;                          /* déclare deux variables entières */

  a = cscoreListGetSection(cs);
  b = cscoreListSeparateF(cs, a);
  cscoreListPut(cs, b);
  cscoreListFreeEvents(cs, b);
  e = cscoreDefineEvent(cs, "t 0 120");
  cscorePutEvent(cs, e);
  cscoreListPut(cs, a);
  cscorePutString(cs, "s");
  cscorePutEvent(cs, e);               /* écrit une autre instruction de tempo */
  b = cscoreListCopyEvents(cs, a);
  dim = 0;                             /* initialise dim à 0 */
  for (n = 1; n <= b->nevents; n++)
  {
    f = b->e[n];
    f->p[6] -= dim;                    /* soustrait la valeur courante de dim */
    f->p[5] *= 0.5;                    /* transpose la hauteur une octave plus bas */
    dim += 2000;                       /* augmente dim pour chaque note */
  }
  a = cscoreListAppendList(cs, a, b);  /* ajoute ces notes aux hauteurs originales */
  cscoreListPut(cs, a);
  cscorePutString(cs, "e");
}
```

En utilisant à nouveau la même partition en entrée, la sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000     ; Trois notes originales aux pulsations
i 1 4 3 0 256 10000     ; 1, 4 et 7 sans diminuendo.
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000     ; Trois notes transposées une octave plus bas
i 1 4 3 0 128 8000      ; également aux pulsations 1, 4 et 7
i 1 7 3 0 440 6000      ; avec diminuendo.
e
```

Dans le programme suivant la même séquence de trois notes sera répétée à divers intervalles de temps. La date de début de chaque groupe est déterminée par les valeurs du tableau *cue*. Cette fois le *dim* se produira sur chaque groupe de notes plutôt que sur chaque note. Remarquez la position de l'instruction qui incrémente la variable *dim* en dehors de la boucle *for* intérieure.

```
#include "cscore.h"
int cue[3] = {0,10,17};                     /* déclare un tableau de 3 entiers */
void cscore(CSOUND *cs)
{
  EVENT  *e, *f;
  EVLIST *a, *b;
  int n, dim, cuecount;                     /* déclare la nouvelle variable cuecount */

  a = cscoreListGetSection(cs);
  b = cscoreListSeparateF(cs, a);
  cscoreListPut(cs, b);
  cscoreListFreeEvents(cs, b);
  e = cscoreDefineEvent(cs, "t 0 120");
  cscorePutEvent(cs, e);
  dim = 0;
  for (cuecount = 0; cuecount <= 2; cuecount++) /* les éléments de cue sont numérotés 0, 1, 2 */
  {
    for (n = 1; n <= a->nevents; n++)
    {
      f = a->e[n];
      f->p[6] -= dim;
      f->p[2] += cue[cuecount];              /* ajoute les valeurs de cue */
    }
    printf("; diagnostic:  cue = %d\n", cue[cuecount]);
    dim += 2000;
    cscoreListPut(cs, a);
  }
  cscorePutString(cs, "e");
}
```

Ici la boucle *for* intérieure lit les évènements de la liste *a* (les notes) et la boucle *for* extérieure lit chaque *répétition* des évènements de la liste *a* (les "répliques" du groupe de hauteurs). Ce programme démontre aussi un moyen utile de résolution de problème au moyen de la fonction *printf*. Le *point-virgule* commence la chaîne de caractères pour produire un commentaire dans le fichier de partition résultant. Dans ce cas, la valeur de *cue* est imprimée en sortie pour s'assurer que le programme prend le bon membre du *tableau* au bon moment. Losrque les données de sortie sont fausses ou que des messages d'erreur sont rencontrés, la fonction *printf* peut aider à identifier le problème.

A partir du même fichier d'entrée, le programme C ci-dessus générera la partition suivante. Pouvez-vous expliquer pourquoi le dernier ensemble de notes ne démarre pas au bon moment et comment corriger le problème ?

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
; diagnostic:  cue = 0
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
; diagnostic:  cue = 10
i 1 11 3 0 440 8000
i 1 14 3 0 256 8000
i 1 17 3 0 880 8000
; diagnostic:  cue = 17
i 1 28 3 0 440 4000
i 1 31 3 0 256 4000
i 1 34 3 0 880 4000
e
```

Compiler un Programme Cscore

Un programme *Cscore* peut être invoqué comme un *programme autonome* ou comme une partie de Csound placée entre le tri de la partition et son exécution par l'orchestre :

`cscore` [*fichier_partition_entrée*] [> *fichier_partition_sortie*]

ou

`csound` [-C] [`autresoptions`] [*nomorch*] [*nompartition*]

Avant d'essayer de compiler votre propre programme *Cscore*, vous voudrez sans doute obtenir une copie du code source de Csound. Téléchargez la distribution des sources la plus récente pour votre plate-forme ou bien récupérez (check out) une copie du module csound5 depuis le CVS de Sourceforge. Il y a plusieurs fichiers dans les sources qui vous aideront. Il y a dans le répertoire `examples/cscore/` plusieurs exemples de programmes de contrôle *Cscore*, y compris tous les exemples contenus dans ce manuel. Et il y a dans le répertoire `frontends/cscore/` les deux fichiers *cscoremain.c* et *cscore.c*. *cscoremain.c* contient une simple fonction *main* qui réalise toute l'initialisation qu'un programme *Cscore* autonome doit faire avant d'appeler votre fonction de contrôle. Cette « souche » main initialise Csound, lit les arguments de la ligne de commande, ouvre les fichiers de partition en entrée et en sortie, et appelle ensuite une fonction *cscore()*. Comme il est décrit ci-dessus, vous êtes chargé d'écrire la fonction *cscore()* et de la fournir dans un autre fichier. Le fichier *frontends/cscore/cscore.c* montre l'exemple le plus simple d'une fonction *cscore()* qui lit une partition de n'importe quelle longueur et l'écrit inchangée sur la sortie.

Ainsi, pour créer un programme autonome, écrivez un programme de contrôle en suivant les indications de la section précédente. Supposons que vous ayez sauvegardé ce programme dans un fichier nommé "*mycscore.c*". Vous devez ensuite compiler ce programme et le lier avec la bibliothèque de Csound et *cscoremain.c* pour créer un exécutable, en suivant l'ensemble de directives ci-dessous qui s'applique à votre système d'exploitation. Il sera utile d'avoir une certaine familiarité avec le compilateur C de votre ordinateur car l'information ci-dessous ne peut pas être exhaustive pour tous les systèmes existants.

Linux et Unix

Les commandes suivantes supposent que vous ayez copié votre fichier *mycscore.c* dans le même répertoire que *cscoremain.c*, que vous ayez ouvert un terminal sur ce même répertoire et que vous ayez installé au préalable une distribution binaire de Csound qui aura placé une bibliothèque *libcsound.a* ou *libcsound.so* dans */usr/local/lib* et les fichiers d'en-tête pour l'API de Csound dans */usr/local/include/csound*.

Pour la compilation et l'édition de liens, tapez :

```
gcc mycscore.c cscoremain.c -o cscore -lcsound -L/usr/local/lib -I/usr/local/include/csound
```

Pour l'exécution (avec envoi des résultats sur la sortie standard), tapez :

```
./cscore test.sco
```

Il est possible que sur certains systèmes Unix le compilateur C soit nommé *cc* ou quelque chose d'autre que *gcc*.

Windows

Csound est ordinairement compilé sur Windows au moyen de l'environnement MinGW qui fournit GCC -- le même compilateur utilisé sur Linux -- au travers d'un shell de commande (MSYS) à la Unix. Comme les bibliothèques pré-compilées pour Csound sur Windows sont construites de cette maniére, vous utiliserez probablement MinGW pour la liaison avec celles-ci. Si vous avez construit Csound en utilisant un autre compilateur, vous serez sans doute capable de construire également *Cscore* avec ce compilateur.

La compilation de programmes *Cscore* autonomes en utilisant MinGW devrait être similaire à la procédure ci-dessus pour Linux avec les chemins de la bibliothèque et des en-têtes changés pour pointer là où Csound est installé sur le système Windows. *(Les contributions plus détaillées sur ces instructions seront les bienvenues car le rédacteur de cet article n'a pas pu tester Cscore sur une machine Windows).*

OS X

Les commandes suivantes supposent que vous ayez copié votre fichier *mycscore.c* dans le même répertoire que *cscoremain.c* et que vous ayez ouvert un terminal dans ce répertoire. De plus, les outils de développement fournis par Apple (incluant le compilateur GCC) doivent être installés sur votre système et vous devez avoir installé une distribution binaire de Csound qui aura placé le framework Csoundlib dans */Library/Frameworks*.

Utilisez cette commande pour la compilation et l'édition de liens. (Il peut y avoir un avertissement sur de "multiples définitions du symbole _cscore").

```
gcc cscore.c cscoremain.c -o cscore -framework CsoundLib -I/Library/Frameworks/CsoundLib.framework/Heac
```

Pour l'exécution (avec envoi des résultats sur la sortie standard) :

```
./cscore test.sco
```

MacOS 9

Vous devrez avoir installé CodeWarrior ou un autre environnement de développement sur votre ordinateur (MPW peut fonctionner). Téléchargez la distribution des sources pour OS 9 (elle aura un nom comme *Csound5.05_OS9_src.smi.bin*).

Si vous utiliser CodeWarrior, trouvez et ouvrez le fichier de projet "Cscore5.cw8.mcp" dans le répertoire "Csound5.04-OS9-source:macintosh:Csound5Library:". Ce fichier de projet est configuré pour utiliser les fichiers source *cscore.c* et *cscoremain_MacOS9.c* situés dans l'arborescence des sources csound5 et la librairie partagée Csound5Lib produite lors de la compilation de Csound avec le fichier de projet "Csound5.cw8.mcp". Il vous faut substituer votre propre fichier du programme *Cscore* à la place de *cscore.c* et soit avoir compilé Csound5Lib avant, soit substituer une copie de la bibliothèque dans le projet à partir de la distribution binaire de Csound pour OS 9. Le fichier *cscoremain_MacOS9.c* contient du code spécialisé pour la configuration de la bibliothèque de console SIOUX de CodeWarrior et permet l'entrée d'arguments de ligne de commande avant le lancement du programme.

Une fois que les fichiers nécessaires sont inclus dans la fénêtre du projet, cliquez sur le bouton "Make" et CodeWarrior produira une application nommée « *Cscore* ». Quand vous lancez cette application, elle affiche d'abord une fenêtre vous permettant de saisir les arguments pour la fonction principale. Vous n'avez qu'à taper le nom de fichier ou le nom de chemin complet de la partition en entrée -- ne tapez pas "cscore". Le fichier d'entrée doit se trouver dans le même répertoire que l'application sinon vous devrez taper un chemin complet ou relatif pour le fichier. La sortie sera affichée dans la fenêtre de console. Vous pouvez utiliser la commande *Save* du menu *File* avant de quitter la console. Alternativement, dans la fenêtre de dialogue de la ligne de commande, vous pouvez choisir de rediriger la sortie dans un fichier en cliquant sur le bouton *File* sur le côté droit de la fenêtre de dialogue. (Notez que la fenêtre de console ne peut afficher qu'environ 32000 caractères, ce qui rend l'écriture dans un fichier nécessaire pour les grandes partitions).

Rendre Cscore utilisable depuis Csound

Pour opérer depuis Csound, suivez d'abord les instructions pour compiler Csound (voir *Construire Csound*) qui concernent le système d'exploitation que vous utilisez. Une fois que vous avez réussi à construire un système Csound non modifié, substituez alors votre propre fonction *cscore()* à celle qui se trouve dans le fichier *Top/cscore_internal.c*, et reconstruisez Csound.

L'exécutable résultant est votre Csound spécial, utilisable comme ci-dessus. L'*option -C* invoquera votre programme *Cscore* après le tri de la partition d'entrée dans « *score.srt* ». Les détails de ce qui se passe lorsque vous lancez Csound avec l'option -C flag sont donnés dans la section suivante.

Csound 5 fournit aussi un moyen supplémentaire d'exécuter votre propre programme *Cscore* depuis Csound. En utilisant l'API, une application hôte peut mettre en place une *fonction d'appel en retour (callback)* de *Cscore*, qui est une fonction que Csound appellera à la place de sa fonction interne *cscore()*. L'avantage de cette approche est qu'il n'est pas nécessaire de recompiler la totalité de Csound. Un autre bénéfice est que l'application hôte peut choisir pendant l'exécution la fonction de callback parmi plusieurs fonctions *Cscore*. L'inconvénient est que vous devez écrire une application hôte.

Une approche simple pour utiliser un callback *Cscore* via l'API serait de modifier le programme main standard de Csound -- qui est un hôte simple de Csound -- contenu dans le fichier *frontends/csound/csound_main.c*. L'ajout d'un appel à *csoundSetCscoreCallback()* après l'appel à *csoundCreate()* mais avant l'appel à *csoundCompile()* devrait faire l'affaire. En recompilant ce fichier et en le liant à une bibliothèque de Csound existante, on obtiendra une version de Csound en ligne de commande qui fonctionne comme celle qui est décrite ci-dessus. N'oubliez pas de taper l'option -C.

Notes au sujet des formats de partition et du comportement de l'exécutable

Comme indiqué précedemment, les fichiers d'entrée de *Cscore* peuvent se trouver dans leur forme originale ou résolue en temps et pré-triée ; cette modalité sera préservée (section par section) lors de la lecture, du traitement et de l'écriture des partitions. Le traitement autonome utilisera le plus souvent des sources non résolues en temps et créera de nouveau fichiers de même forme. Lors du traitement depuis Csound, la partition en entrée arrivera déjà résolue en temps et triée, et pourra ainsi être envoyée directement (normalement section par section) à l'orchestre. Un des avantages de cette façon d'utiliser *Cscore*

est que toutes les commodités de syntaxe du langage de partition complet de Csound peuvent être utilisées -- macros, expressions arithmétiques, carry, rampes, etc. -- car la partition passera par les phases "Carry, Tempo, Tri" du traitement avant d'être transmise au programme *Cscore* fourni par l'utilisateur.

Lors du traitement dans Csound, une liste d'évènements peut être transmise à un orchestre de Csound en utilisant *cscoreListPlay()*. Il peut y avoir n'importe quel nombre d'appels de *cscoreListPlay()* dans un programme *Cscore*. Chaque liste ainsi transmise peut-être résolue ou non en temps, mais chaque liste doit être en ordre chronologique strict par rapport à *p2* (soit grâce au pré-traitement de tri soit en utilisant *cscoreListSort()*). S'il n'y a pas de *cscoreListPlay()* dans un module *Cscore* exécuté depuis Csound, tous les évènements écrits en sortie (via *cscorePutEvent()*, *cscorePutString()*, ou *cscoreListPut()*) sont envoyés dans une nouvelle partition dans le répertoire courant nommée « *cscore.out* ». Csound invoque alors à nouveau le tri de partition avant d'envoyer cette nouvelle partition à l'orchestre pour son exécution. La partition de sortie triée finale est écrite dans un fichier nommé « *cscore.srt* ».

Un programme *Cscore* autonome utilisera normalement la commande « put » pour écrire dans son fichier de sortie. Si un programme *Cscore* autonome appelle *cscoreListPlay()*, les évènements ainsi destinés à l'exécution seront envoyés sur la sortie comme s'ils provenaient de *cscoreListPut()*.

Une liste de notes envoyée par *cscoreListPlay()* pour exécution doit être distincte dans le temps des listes de notes suivantes. Aucune fin de note ne doit dépasser la date de début de la liste suivante, car *cscoreListPlay()* complètera chaque liste avant d'attaquer la suivante (comme un marqueur de Section qui ne réinitialise pas le temps local à zéro). C'est important lorsque l'on utilise *cscoreListGetNext()* ou *cscoreListGetUntil()* pour charger et traiter des segments de partition avant exécution, car ces fonctions pourraient ne lire qu'un partie d'une section non triée.

Exemples Plus Avancés

Le programme suivant démontre la lecture à partir de deux fichiers d'entrée différents. L'idée est d'alterner entre deux partitions de 2 sections, et d'écrire les sections entrelacées dans un seul fichier de sortie.

```
#include "cscore.h"              /*   CSCORE_SWITCH.C  */
cscore(CSOUND* cs)               /* appelable depuis Csound ou comme cscore autonome */
{
  EVLIST *a, *b;
  FILE   *fp1, *fp2;                      /* deux pointeurs sur des flots de fichier de partition */
  fp1 = cscoreFileGetCurrent(cs);         /* la partition de la ligne de commande */
  fp2 = cscoreFileOpen(cs, "score2.srt"); /* une partition supplémentaire */
  a = cscoreListGetSection(cs);           /* lit une section de la partition 1 */
  cscoreListPut(cs, a);                   /* l'écrit en sortie telle quelle */
  cscorePutString(cs, "s");
  cscoreFileSetCurrent(cs, fp2);
  b = cscoreListGetSection(cs);           /* lit une section de la partition 2 */
  cscoreListPut(cs, b);                   /* l'écrit en sortie telle quelle */
  cscorePutString(cs, "s");
  cscoreListFreeEvents(cs, a);            /* facultatif, pour libérer de l'espace */
  cscoreListFreeEvents(cs, b);
  cscoreFileSetCurrent(cs, fp1);
  a = cscoreListGetSection(cs);           /* lit la section suivante de la partition 1 */
  cscoreListPut(cs, a);                   /* l'écrit en sortie */
  cscorePutString(cs, "s");
  cscoreFileSetCurrent(cs, fp2);
  b = cscoreListGetSection(cs);           /* lit la section suivante de la partition 2 */
  cscoreListPut(cs, b);                   /* l'écrit en sortie */
  cscorePutString(cs, "e");
}
```

Finalement, nous montrons comment prendre un fichier de partition littérale, non interprétée et lui insuffler un peu d'expressivité rythmique. La théorie des pulsations métriques liées au compositeur a été étudiée en profondeur par Manfred Clynes, et la suite est dans l'esprit de ce travail. Ici, la stratégie consiste à créer d'abord un *tableau* de nouvelles dates de *début* pour chaque début possible de double croche,

puis par indexation dans ce tableau, d'ajuster le début et la durée de chaque note de la partition d'entrée aux dates interprétées. On montre aussi comment un orchestre de Csound peut être invoqué de façon répétitive depuis un générateur de partition pendant l'exécution.

```
#include "cscore.h"              /*   CSCORE_PULSE.C  */

/* programme pour appliquer une pulsation aux durées interprétées   */
/* à une partition existante en 3/4, premiers temps sur 0, 3, 6 ... */


static float four[4] = { 1.05, 0.97, 1.03, 0.95 };   /* largeur de pulsation des 4 */
static float three[3] = { 1.03, 1.05, .92 };         /* largeur de pulsation des 3 */


cscore(CSOUND* cs)            /* Cet exemple doit être appelé depuis Csound  */
{
  EVLIST  *a, *b;
  EVENT  *e, **ep;
  float pulse16[4*4*4*4*3*4];    /* tableau de doubles croches, 3/4, 256 mesures */
  float acc16, acc1,inc1, acc3,inc3, acc12,inc12, acc48,inc48, acc192,inc192;
  float *p = pulse16;
  int  n16, n1, n3, n12, n48, n192;

  /* remplit le tableau avec les dates de début de l'interprétation  */
  for (acc192=0.,n192=0; n192<4; acc192+=192.*inc192,n192++)
    for (acc48=acc192,inc192=four[n192],n48=0; n48<4; acc48+=48.*inc48,n48++)
      for (acc12=acc48,inc48=inc192*four[n48],n12=0;n12<4; acc12+=12.*inc12,n12++)
        for (acc3=acc12,inc12=inc48*four[n12],n3=0; n3<4; acc3+=3.*inc3,n3++)
          for (acc1=acc3,inc3=inc12*four[n3],n1=0; n1<3; acc1+=inc1,n1++)
            for (acc16=acc1,inc1=inc3*three[n1],n16=0; n16<4; acc16+=.25*inc1*four[n16],n16++)
              *p++ = acc16;


  /* for (p = pulse16, n1 = 48; n1--; p += 4)  /* montre les valeurs & les différences */
  /*   printf("%g %g %g %g %g %g %g %g\n", *p, *(p+1), *(p+2), *(p+3),
  /*  *(p+1)-*p, *(p+2)-*(p+1), *(p+3)-*(p+2), *(p+4)-*(p+3)); */

  a = cscoreListGetSection(cs);               /* lit une section de la partition résolue en temps */
  b = cscoreListSeparateTWF(cs, a);           /* sépare les instructions de jeu et de fonction */
  cscoreListPlay(cs, b);                      /* et les envoie à l'exécution */
  a = cscoreListAppendStringEvent(cs, a, "s"); /* ajoute une instruction de section à la liste de notes
  cscoreListPlay(cs, a);                      /* joue la liste de notes sans interprétation */
  for (ep = &a->e[1], n1 = a->nevents; n1--; ) { /* maintenant modifie les pulsations */
    e = *ep++;
    if (e->op == 'i') {
      e->p[2] = pulse16[(int)(4. * e->p2orig)];
      e->p[3] = pulse16[(int)(4. * (e->p2orig + e->p3orig))] - e->p[2];
    }
  }

  cscoreListPlay(cs, a);                      /* maintenant joue la liste modifiée */
}
```

Etendre Csound

Ajouter des Générateurs Unitaires

Si les générateurs unitaires existants de Csound ne répondent pas à vos besoins, il est relativement aisé d'étendre Csound en écrivant de nouveaux générateurs unitaires en C ou en C++. Le traducteur, le chargeur et le moniteur d'exécution traiteront votre module comme n'importe quel autre module, pourvu que vous suiviez certaines conventions.

Historiquement, on réalisait ceci avec des générateurs unitaires intégrés, c'est-à-dire dont le code est lié statiquement avec le reste de l'exécutable de Csound.

Aujourd'hui, on préfère créer des générateurs unitaires sous forme de plugin. Ce sont des bibliothèques à liaison dynamique (DLL) sous Windows, et des modules chargeables (bibliothèques partagées chargées par `dlopen`) sur Linux. Csound recherche et charge ces plugins au moment de l'exécution. L'avantage de cette méthode, naturellement, est que les plugins créés par n'importe quel développeur, n'importe quand, peuvent être utilisés avec des versions de Csound déjà existantes.

Créer un Générateur Unitaire Intégré

Vous avez besoin d'une structure définissant les entrées, les sorties et l'espace de travail, plus du code d'initialisation et du code d'exécution. Mettons un exemple de tout cela dans deux nouveaux fichiers, `newgen.h` et `newgen.c`. Les exemples donnés sont pour Csound 5. Pour les versions antérieures, il faut omettre le premier paramètre (`CSOUND *csound`) dans toutes les fonctions d'opcode.

```
/* newgen.h  -  définit une structure */

/* Déclare les structures et les fonctions de Csound. */
#include "csoundCore.h"

typedef struct
{
  OPDS h;                                         /* en-tête requis */
  MYFLT *result, *istrt, *incr, *itime, *icontin; /* adr des arg de sortie et d'entrée */
  MYFLT curval, vincr;                            /* espace de données privé */
  long countdown;                                 /* ditto */
} RMP;


/* newgen.c -  code d'initialisation et d'exécution */
/* Déclare les structures et les fonctions de Csound. */
#include "csoundCore.h"
/* Déclare la structure RMP. */
#include "newgen.h"

int rampset (CSOUND *csound, RMP * p)     /* à l'initialisation de la note : */
{
  if (*p->icontin == FL(0.0))
    p->curval = *p->istrt;                /* reçoit si besoin la nouvelle valeur de début */
  p->vincr = *p->incr / csound->esr;      /* fixe l'increment au taux-s par sec. */
  p->countdown = *p->itime * csound->esr; /* compteur pour iduree en secondes */
  return OK;
}

int ramp (CSOUND *csound, RMP * p)        /* pendant l'exécution de la note : */
{
  MYFLT *rsltp = p->result;               /* initialise un pointeur sur le tableau de sortie */
  int nn = csound->ksmps;                 /* taille du tableau donnée par l'orchestre */
  do
    {
      *rsltp++ = p->curval;               /* copie la valeur courante vers la sortie */
      if (--p->countdown > 0)             /* pour les premières iduree secondes, */
        p->curval += p->vincr;            /* incrémenter la valeur */
    }
```

```
  while (--nn);
  return OK;
}
```

Maintenant nous ajoutons ce module à la table du traducteur dans `entry1.c`, sous le nom d'opcode `rampt` :

```
#include "newgen.h"

int rampset(CSOUND *, RMP *), ramp(CSOUND *, RMP *);

/*  opname  dsblksiz  thread    outypes   intypes      iopadr        kopadr      aopadr  */
{  "rampt",  S(RMP),    5,        "a",     "iiio",   (SUBR)rampset, (SUBR)NULL, (SUBR)ramp  },
```

Finalement, il faut relier Csound avec le nouveau module. Ajoutez le nom du fichier C à la liste `libCsoundSources` dans le fichier `SConstruct` :

```
libCsoundSources = Split('''
Engine/auxfd.c
...
OOps/newgen.c
...
Top/utility.c
''')
```

Lancez scons comme vous le feriez pour toute autre construction de Csound, et le nouveau module sera intégré dans votre Csound.

Les actions ci-dessus ont ajouté un nouveau générateur au langage Csound. C'est une fonction de rampe linéaire au taux audio qui modifie une valeur d'entrée selon une pente définie par l'utilisateur pour une durée donnée. Cette rampe peut éventuellement continuer depuis la dernière valeur de la note précédente. L'entrée correspondante du manuel de Csound ressemblerait à ceci :

```
ar rampt idebut, ipente, iduree [, icontin]
```

idebut -- valeur du début d'une rampe linéaire au taux audio. Eventuellement ignorée s'il y a un drapeau de continuité.

ipente -- pente de la rampe, exprimée comme le taux de changement des y par seconde.

iduree -- durée de la rampe en secondes, après laquelle la valeur est tenue jusqu'à la fin de la note.

icontin (facultatif) -- drapeau de continuité. S'il est à zéro, la rampe démarrera depuis l'entrée *idebut*. Sinon, la rampe démarrera depuis la dernière valeur de la note précédente. La valeur par défaut est zéro.

Le fichier `newgen.h` comprend une liste de paramètres de sortie et d'entrée définie sur une ligne. Ce sont les ports par lesquels le nouveau générateur communiquera avec les autres générateurs dans un instrument. La communication se fait par *adresse*, pas par *valeur*, et c'est une liste de pointeurs sur des valeurs de type MYFLT (*double* si la macro USE_DOUBLE est définie, et *float* autrement). Il n'y a aucune restriction sur les noms, mais les types d'argument d'entrée-sortie sont définis plus loin par des chaînes de caractères dans `entry1.c` (intypes, outypes). Les types intypes sont habituellement *x*, *a*, *k*, et *i*, suivant les conventions normales du manuel de Csound ; on trouve aussi *o* (facultatif, par défaut 0), *p* (facultatif,

par défaut 1). Les types outypes comprennent *a*, *k*, *i* et *s* (asig ou ksig). Il est important que tous les noms d'argument de la liste se voient attribuer un type d'argument correspondant dans `entry1.c`. De plus, les arguments de type-i ne sont valides qu'à l'initialisation, et les arguments des autres types ne sont valables que pendant l'exécution. Les lignes suivantes de la structure RMP déclarent l'espace de travail nécessaire pour que le code soit réentrant. Ceci permet d'utiliser le module plusieurs fois dans plusieurs copies d'instrument tout en préservant toutes les données.

Le fichier `newgen.c` contient deux sous-programmes, appelés chacun avec un pointeur sur l'instance de Csound et un pointeur sur la structure RMP allouée de façon unique et ses données. Les sous-programmes peuvent être de trois sortes : initialisation de note, génération de signal au taux-`k`, génération de signal au taux-`a`. Normalement, un module requiert deux de ces sous-programmes : initialisation, et un sous-programme soit de taux-`k`, soit de taux-`a` qui sera inséré dans divers listes chaînées de tâches exécutables quand un instrument est activé. Les type de chaînage apparaissent dans `entry1.c` sous deux formes : noms *isub*, *ksub* et *asub* ; et un index de chaînage qui est la somme de isub=1, ksub=2, asub=4. Le code lui-même peut référencer (mais ça ne devrait être qu'en lecture) les membres publiques de la structure CSOUND définie dans `csoundCore.h`, dont les plus utiles sont :

```
OPARMS  *oparms
MYFLT   esr                 taux d'échantillonage défini par l'utilisateur
MYFLT   ekr                 taux de contrôle défini par l'utilisateur
int     ksmps               ksmps défini par l'utilisateur
int     nchnls              nchnls défini par l'utilisateur
int     oparms->odebug      option -v de la ligne de commande
int     oparms->msglevel    option -m de la ligne de commande
MYFLT   tpidsr              2 * PI / esr
```

Tables de Fonction

pour accéder aux tables de fonction en mémoire, une aide spéciale est disponible. La nouvelle structure définie doit comprendre un pointeur

```
FUNC        *ftp;
```

initialisé par l'instruction

```
ftp = csound->FTFind(csound, p->ifuncno);
```

où MYFLT *ifuncno est un argument d'entrée de type-i contenant le numéro de la ftable. La table stockée est alors en ftp->ftable, et d'autres données comme sa longueur, les masques de phase, les convertisseurs cps-incrément, sont aussi accessibles depuis ce pointeur. Voir la structure FUNC dans `csoundCore.h`, le code de csoundFTFind() dans `fgens.c`, et le code de oscset() et de koscil() dans `OOps/ugens2.c`.

Espace Supplémentaire

Parfois les besoins en espace d'un module sont trop grands pour faire partie d'une structure (limite supérieure de 65279 octets, due au paramètre en entier court non-signé *dsblksiz* et aux codes réservés >= 0xFF00), ou ils dépendent d'une valeur d'argument-i qui n'est pas connue avant l'initialisation. De l'espace supplémentaire peut être alloué dynamiquement et géré proprement en incluant la ligne

```
AUXCH       auxch;
```

dans la structure définié (*p), puis en utilisant ce type de code dans le module d'initialisation :

```
csound->AuxAlloc(csound, npoints * sizeof(MYFLT), &p->auxch);
```

L'adresse de l'espace auxiliaire est gardée dans une chaîne d'espaces similaires appartenant à cet intrument, et elle est gérée automatiquement lorsque l'instrument est dupliqué ou passé au ramasse-miettes durant l'exécution. L'assignation

```
void *auxp = p->auxch.auxp;
```

trouvera les espaces alloués pour une utilisation pendant l'initialisation et pendant l'exécution. Voir la structure LINSEG dans `ugens1.h` et le code de lsgset() and klnseg() dans `OOps/ugens1.c`.

Partage de Fichier

Lorsque l'on accède souvent à un fichier externe, ou si on le fait depuis plusieurs endroits, il est souvent efficace de lire le fichier entier dans la mémoire. On accomplit ceci en incluant la ligne

```
MEMFIL     *mfp;
```

dans la structure définie (*p), puis en utilisant le style de code suivant dans le module d'initialisation :

```
p->mfp = csound->ldmemfile(csound, nomfic);
```

où char *nomfic est une chaîne contenant le nom du fichier requis. Les données lues se trouveront entre

```
(char *)p->mfp->beginp;  et  (char *)p->mfp->endp;
```

Les fichiers chargés n'appartiennent pas à un instrument particulier, mais sont automatiquement partagés pour des accès multiples. Voir la structure ADSYN dans `ugens3.h` et le code de adset() et de adsyn() dans `OOps/ugens3.c`.

Arguments Chaîne

Pour permettre un argument d'entrée de type chaîne (disons MYFLT *inomfic) dans votre structure définie (*p), assignez-lui le type d'argument *S* dans `entry1.c`, et incluez le code suivant dans le module d'initialisation :

```
strcpy(nomfic, (char*)p->inomfic);
```

Voir le code pour adset() dans `OOps/ugens3.c`, lprdset() dans `OOps/ugens5.c`, et pvset() dans `OOps/ugens8.c`.

Ajouter un Générateur Unitaire comme Plugin

La procédure pour créer un généteur unitaire comme plugin ressemble beaucoup à celle qui est utilisée pour créer un générateur intégré. Le code du générateur unitaire sera le même à part les différences suivantes.

En supposant à nouveau que votre générateur s'appelle `newgen`, effectuez les étapes suivantes :

1. Ecrivez vos fichiers `newgen.c` et `newgen.h` comme vous le feriez pour un générateur unitaire intégré. Mettez ces fichiers dans le répertoire `csound5/Opcodes`.
2. Mettez `#include "csdl.h"` dans les sources de votre générateur unitaire, au lieu de `#include "csoundCore.h"`.
3. Ajoutez vos champs `OENTRY` et les fonctions d'enregistrement du générateur unitaire au bas de votre fichier C. Exemple (mais vous pouvez avoir autant de générateurs unitaires que vous le voulez dans un plugin) :

```
#define S sizeof

static OENTRY localops[] = {
{
    { "rampt", S(RMP), 5, "a", "iiio", (SUBR)rampset, (SUBR)NULL, (SUBR)ramp  },
};

/*
 * La macro suivante de csdl.h définit
 * la fonction d'enregistrement d'opcode "csound_opcode_init()"
 * pour la table des opcodes locaux.
 */
LINKAGE
```

4. Ajoutez votre plugin comme nouvelle cible dans la section des opcodes en plugin du fichier de construction `SConstruct` :

```
pluginEnvironment.SharedLibrary('newgen',
    Split('''Opcodes/newgen.c
    Opcodes/un_autre_fichier_utilise_par_newgen.c
    Opcodes/encore_un_autre_fichier_utilise_par_newgen.c'''))
```

5. Lancer la construction de Csound de la manière usuelle.

Référence de `OENTRY`

La structure `OENTRY` (voir `H/csoundCore.h`, `Engine/entry1.c`, et `Engine/rdorch.c`) contient les champs publiques suivants :

```
opname, dsblksiz, thread, outypes, intypes, iopadr, kopadr, aopadr
```

`dsblksiz`
: Il y a deux types d'opcode, polymorphe et non-polymorphe. Pour les opcodes non-polymorphes, le drapeau `dsblksiz` spécifie la taille de la structure de l'opcode en octets, et les arguments sont toujours passés à l'opcode au même taux. Les opcodes polymorphes peuvent accepter des arguments à des taux différents, et la façon dont ces arguments sont réellement distribués aux autres opcodes est déterminée par le drapeau `dsblksiz` et les conventions de nommage suivantes (note : la liste suivante est incomplète, voir `Engine/entry1.c` pour tous les codes spéciaux possibles pour dsblksiz) :

`0xffff` Le type du premier argument en sortie détermine quelle fonction de générateur unitaire est réellement appelée : `xxx` -> `xxx.a`, `xxx.i`, ou `xxx.k`.

`0xfffe` Les types des deux premiers arguments en entrée déterminent quelle fonction de générateur unitaire est réellement appelée : `xxx` -> `xxx.aa`, `xxx.ak`, `xxx.ka`, ou `xxx.kk`, comme dans le générateur unitaire `oscil`.

`0xfffd` Fait référence à un argument en entrée de type a ou k, comme dans le générateur unitaire `peak`.

`thread` Spécifie le(s) taux utilisé(s) pour appeler les fonctions de générateur unitaire, comme suit :

Tableau 18. Taux d'appel des ugens selon le paramètre thread

0	taux-`i` *ou* taux-`k` (sortie B seulement)
1	taux-`i`
2	taux-`k`
3	taux-`i` *et* taux-`k`
4	taux-`a`
5	taux-`i` *et* taux-`a`
7	taux-`i` *et* (taux-`k` *ou* taux-`a`)

`outypes` Liste les valeurs de retour des fonctions de générateur unitaire, s'il y en a. Les types permis sont (note : la liste suivante est incomplète, voir `Engine/entry1.c` pour tous les types possibles en sortie) :

Tableau 19. Liste des types de sortie des ugens

i	scalaire de taux-`i`
k	scalaire de taux-`k`
a	vecteur de taux-`a`
x	scalaire de taux-`k` ou vecteur de taux-`a`
f	type fsig de flux pvoc de taux-`f`
m	arguments multiples en sortie de taux-`a`

`intypes` Liste les arguments, s'il y en a, que prennent les fonctions de générateur unitaire. Les types permis sont (note : la liste suivante est incomplète, voir `Engine/entry1.c` pour tous les types possibles en entrée) :

Tableau 20. Liste des types d'entrée des ugens

i	scalaire de taux-`i`
k	scalaire de taux-`k`
a	vecteur de taux-`a`
x	scalaire de taux-`a` ou vecteur de taux-`a`

`f`	type fsig de flux pvoc de taux-`f`
`S`	Chaîne
`B`	
`l`	
`m`	Commence une liste indéfinie d'arguments de taux-`i` (n'importe quel nombre)
`M`	Commence une liste indéfinie d'arguments (n'importe quel taux, n'importe quel nombre)
`N`	Commence une liste indéfinie d'arguments facultatifs (aux taux-`a`, -`k`, -`i`, ou -`S`) (n'importe quel nombre impair)
`n`	Commence une liste indéfinie d'arguments au taux-`i` (n'importe quel nombre impair)
`O`	facultatif au taux-`k`, 0 par défaut
`o`	facultatif au taux-`i`, 0 par défaut
`p`	facultatif au taux-`i`, 1 par défaut
`q`	facultatif au taux-`i`, 10 par défaut
`V`	facultatif au taux-`k`, 0.5 par défaut
`v`	facultatif au taux-`i`, 0.5 par défaut
`j`	facultatif au taux-`i`,-1 par défaut
`h`	facultatif au taux-`i`, 127 par défaut
`y`	Commence une liste indéfinie d'arguments au taux-`a` (n'importe quel nombre)
`z`	Commence une liste indéfinie d'arguments au taux-`k` (n'importe quel nombre)
`Z`	Commence une liste indéfinie d'argumenents alternant les taux-`k` et -`a` (kaka...) (n'importe quel nombre)

`iopadr` L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée à l'initialisation, ou NULL s'il n'y a pas de fonction.

`kopadr` L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée au taux-`k`, ou NULL s'il n'y a pas de fonction.

`aopadr` L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée au taux-`a`, ou NULL s'il n'y a pas de fonction.

Partie IV. Référence Rapide des Opcodes

Table des matières

Référence Rapide des Opcodes 2433

Référence Rapide des Opcodes

Syntaxe de l'Orchestre : En-tête.

```
0dbfs = iarg
0dbfs
```

```
kr = iarg
```

```
ksmps = iarg
```

```
nchnls = iarg
```

```
sr = iarg
```

Syntaxe de l'Orchestre : Bloc d'Instructions.

```
endin
```

```
endop
```

```
instr i, j, ...
```

```
opcode nom, outtypes, intypes
```

Syntaxe de l'Orchestre : Macros.

```
#define NAME # replacement text #
```

```
#define NAME(a' b' c') # replacement text #
```

```
$NAME
```

```
#ifdef NAME
  ....
#else
  ....
#end
```

```
#ifndef NAME
  ....
#else
  ....
#end
```

```
#include "filename"
```

```
#undef NAME
```

Générateurs de Signal : Synthèse/Resynthèse Additive.

```
ares adsyn kamod, kfmod, ksmod, ifilcod
```

```
ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

```
ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

```
ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \
               [, ioctcnt] [, iphs]
```

Générateurs de Signal : Oscillateurs Elémentaires.

```
kres lfo kamp, kcps [, itype]
ares lfo kamp, kcps [, itype]

ares oscbnk  kcps, kamd, kfmd, kpmd, iovrlap, iseed, kl1minf, kl1maxf, \
      kl2minf, kl2maxf, ilfomode, keqminf, keqmaxf, keqminl, keqmaxl, \
      keqminq, keqmaxq, ieqmode, kfn [, il1fn] [, il2fn] [, ieqffn]   \
      [, ieqlfn] [, ieqqfn] [, itabl] [, ioutfn]

ares oscil xamp, xcps, ifn [, iphs]
kres oscil kamp, kcps, ifn [, iphs]

ares oscil3 xamp, xcps, ifn [, iphs]
kres oscil3 kamp, kcps, ifn [, iphs]

ares oscili xamp, xcps, ifn [, iphs]
kres oscili kamp, kcps, ifn [, iphs]

ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]

ares osciliktp kcps, kfn, kphs [, istor]

ares oscilikts xamp, xcps, kfn, async, kphs [, istor]

ares osciln kamp, ifrq, ifn, itimes

ares oscils iamp, icps, iphs [, iflg]

ares poscil aamp, acps, ifn [, iphs]
ares poscil aamp, kcps, ifn [, iphs]
ares poscil kamp, acps, ifn [, iphs]
ares poscil kamp, kcps, ifn [, iphs]
ires poscil kamp, kcps, ifn [, iphs]
kres poscil kamp, kcps, ifn [, iphs]

ares poscil3 kamp, kcps, ifn [, iphs]
kres poscil3 kamp, kcps, ifn [, iphs]

kout vibr kAverageAmp, kAverageFreq, ifn

kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \
      kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \
      kcpsMaxRate, ifn [, iphs]
```

Générateurs de Signal : Oscillateurs à Spectre Dynamique.

```
ares buzz xamp, xcps, knh, ifn [, iphs]

ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]

ares mpulse kamp, kintvl [, ioffset]

ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \
      [, iphs] [, iskip]

ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]

kfn vco2ft kcps, iwave [, inyx]

ifn vco2ift icps, iwave [, inyx]
```

```
ifn vco2init iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]
```

Générateurs de Signal : Synthèse FM.

```
ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn

ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn

ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn

ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
      ifn3, ifn4, ivfn

ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
      ifn3, ifn4, ivfn

ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, \
      ifn2, ifn3, ifn4, ivibfn

ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
      ifn4, ivfn

ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]

ares foscili xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

Générateurs de Signal : Synthèse Granulaire.

```
asig diskgrain Sfname, kamp, kfreq, kpitch, kgrsize, kprate, \
      ifun, iolaps[, ioffset, imaxgrsize]

ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]

ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kgliss [, iskip]

ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \
      iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]

ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \
      iwfn, imgdur [, igrnd]

ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \
      [, iseed] [, imode]

ares grain3 kcps, kphs, kfmd, kpmd, kgdur, kdens, imaxovr, kfn, iwfn, \
      kfrpow, kprpow [, iseed] [, imode]

ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \
      igskip_os, ilength, kgap, igap_os, kgsize, igsize_os, iatt, idec \
      [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]

a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
              kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
              ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmasks, \
              kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
              ifmamptab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
              ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, \
              kwaveform4, iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
              asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
              [, iopcode_id]

async [,aphase] partikkelsync iopcode_id
```

```
ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsize, \
      irandw, ioverlap, ifn2, itimemode

ar1, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \
      ibeg, iwsize, irandw, ioverlap, ifn2, itimemode

asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \
      ifun2, iolaps

asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \
      klend, ifun1, ifun2, iolaps[,istart, iskip]

ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]
```

Générateurs de Signal : Synthèse Hyper Vectorielle.

```
hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]

hvs2 kx, ky, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]

hvs3 kx, ky, kz, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfig-
Tab]
```

Générateurs de Signal : Générateurs Linéaires et Exponentiels.

```
kout expcurve kindex, ksteepness

ares expon ia, idur, ib
kres expon ia, idur, ib

ares expseg ia, idur1, ib [, idur2] [, ic] [...]
kres expseg ia, idur1, ib [, idur2] [, ic] [...]

ares expsega ia, idur1, ib [, idur2] [, ic] [...]

ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz

kout gainslider kindex

ares jspline xamp, kcpsMin, kcpsMax
kres jspline kamp, kcpsMin, kcpsMax

ares line ia, idur, ib
kres line ia, idur, ib

ares linseg ia, idur1, ib [, idur2] [, ic] [...]
kres linseg ia, idur1, ib [, idur2] [, ic] [...]

ares linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz

kout logcurve kindex, ksteepness

ksig loopseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]

ksig loopsegp  kphase, kvalue0, kdur0, kvalue1 \
      [, kdur1, ... , kdurN-1, kvalueN]

ksig lpshold kfreq, ktrig, ktime0, kvalue0  [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]

ksig lpsholdp  kphase, ktrig, ktime0, kvalue0  [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

```
ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax

kscl scale kinput, kmax, kmin

ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

Générateurs de Signal : Générateurs d'Enveloppe.

```
ares adsr iatt, idec, islev, irel [, idel]
kres adsr iatt, idec, islev, irel [, idel]

ares envlpx xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
kres envlpx kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]

ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]

ares linen xamp, irise, idur, idec
kres linen kamp, irise, idur, idec

ares linenr xamp, irise, idec, iatdec
kres linenr kamp, irise, idec, iatdec

ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]

ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]

ares xadsr iatt, idec, islev, irel [, idel]
kres xadsr iatt, idec, islev, irel [, idel]
```

Générateurs de Signal : Modèles et Emulations.

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]

ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid

ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]

aI3, aV2, aV1 chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step

ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]

ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]

ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn

ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]

ax, ay, az lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip [, iskipinit]

kiter, koutrig mandel  ktrig, kx, ky, kmaxIter

ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [, iminfreq]

ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \
      [, idoubles] [, itriples]

ares moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \
```

```
      [, ifriction] [, iskip]

ares prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]
al,ar prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]

ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]

ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]

ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]

ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]

ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]

ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
      [, ifreq1] [, ifreq2]

ares vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec

ares voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn
```

Générateurs de Signal : Phaseurs.

```
ares phasor xcps [, iphs]
kres phasor kcps [, iphs]

ares phasorbnk xcps, kndx, icnt [, iphs]
kres phasorbnk kcps, kndx, icnt [, iphs]

aphase, asyncout syncphasor xcps, asyncin, [, iphs]
```

Générateurs de Signal : Générateurs de Nombres Aléatoires (de Bruit).

```
ares betarand krange, kalpha, kbeta
ires betarand krange, kalpha, kbeta
kres betarand krange, kalpha, kbeta

ares bexprnd krange
ires bexprnd krange
kres bexprnd krange

ares cauchy kalpha
ires cauchy kalpha
kres cauchy kalpha

aout cuserrnd kmin, kmax, ktableNum
iout cuserrnd imin, imax, itableNum
kout cuserrnd kmin, kmax, ktableNum

aout duserrnd ktableNum
iout duserrnd itableNum
kout duserrnd ktableNum

ares exprand klambda
ires exprand klambda
kres exprand klambda

ares gauss krange
ires gauss krange
kres gauss krange
```

```
kout jitter kamp, kcpsMin, kcpsMax

kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3

ares linrand krange
ires linrand krange
kres linrand krange

ares noise xamp, kbeta

ares pcauchy kalpha
ires pcauchy kalpha
kres pcauchy kalpha

ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]

ares poisson klambda
ires poisson klambda
kres poisson klambda

ares rand xamp [, iseed] [, isel] [, ioffset]
kres rand xamp [, iseed] [, isel] [, ioffset]

ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]

ares randi xamp, xcps [, iseed] [, isize] [, ioffset]
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]

ares random kmin, kmax
ires random imin, imax
kres random kmin, kmax

ares randomh kmin, kmax, acps
kres randomh kmin, kmax, kcps

ares randomi kmin, kmax, acps
kres randomi kmin, kmax, kcps

ax rnd31 kscl, krpow [, iseed]
ix rnd31 iscl, irpow [, iseed]
kx rnd31 kscl, krpow [, iseed]

seed ival

kout trandom ktrig, kmin, kmax

ares trirand krange
ires trirand krange
kres trirand krange

ares unirand krange
ires unirand krange
kres unirand krange

aout = urd(ktableNum)
iout = urd(itableNum)
kout = urd(ktableNum)

ares weibull ksigma, ktau
ires weibull ksigma, ktau
kres weibull ksigma, ktau
```

Générateurs de Signal : Reproduction de Sons Echantillonnés.

```
a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \
      [, istutterspeed] [, istutterchance] [, ienvchoice ]

a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
      inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]
```

```
asig flooper kamp, kpitch, istart, idur, ifad, ifn

asig flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \
      [, istart, imode, ifenv, iskip]

aleft, aright fluidAllOut

fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue

fluidCCk iEngineNumber, iChannelNumber, iControllerNumber, kValue

fluidControl ienginenum, kstatus, kchannel, kdata1, kdata2

ienginenum fluidEngine [iReverbEnabled] [, iChorusEnabled] [,iNumChannels] [, iPolypho-
ny]

isfnum fluidLoad soundfont, ienginenum[, ilistpresets]

fluidNote ienginenum, ichannelnum, imidikey, imidivel

aleft, aright fluidOut ienginenum

fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum

fluidSetInterpMethod ienginenum, ichannelnum, iInterpMethod

ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
      [, imod2] [, ibeg2] [, iend2]

ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
      [, imod2] [, ibeg2] [, iend2]

ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
      ar15, ar16] loscilx xamp, kcps, ifn \
      [, iwsize, ibas, istrt, imod1, ibeg1, iend1]

ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istrt] [, istor]

ares lposcil kamp, kfreqratio, kloop, kend, ifn [, iphs]

ares lposcil3 kamp, kfreqratio, kloop, kend, ifn [, iphs]

ar lposcila aamp, kfreqratio, kloop, kend, ift [,iphs]

ar1, ar2 lposcilsa aamp, kfreqratio, kloop, kend, ift [,iphs]

ar1, ar2 lposcilsa2 aamp, kfreqratio, kloop, kend, ift [,iphs]

sfilist ifilhandle

ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]

ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]

ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]

ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
      [, iflag] [, ioffset]

ir sfload "filename"

ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend,
kcrossfade, ifn \
      [, istart, imode, ifenv, iskip]

sfpassign istartindex, ifilhandle[, imsgs]

ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

```
ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

sfplist ifilhandle

ir sfpreset iprog, ibank, ifilhandle, ipreindex

asig, krec sndloop ain, kpitch, ktrig, idur, ifad

ares waveset ain, krep [, ilen]
```

Générateurs de Signal : Synthèse par Balayage.

```
scanhammer isrc, idst, ipos, imult

ares scans kamp, kfreq, ifn, id [, iorder]

aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel

scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, iright, kpos, kstrngth, ain, idisp, id

kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]

ares xscans kamp, kfreq, ifntraj, id [, iorder]

xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]

xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, iright, kpos, kstrngth, ain, idisp, id
```

Générateurs de Signal : Accès aux Tables.

```
kres oscil1 idel, kamp, idur, ifn

kres oscil1i idel, kamp, idur, ifn

ir tab_i indx, ifn[, ixmode]
kr tab kndx, ifn[, ixmode]
ar tab xndx, ifn[, ixmode]
tabw_i isig, indx, ifn [,ixmode]
tabw ksig, kndx, ifn [,ixmode]
tabw asig, andx, ifn [,ixmode]

ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]

ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]

ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

Générateurs de Signal : Synthèse par Terrain d'Ondes.

```
aout wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \
      itabx, itaby
```

Générateurs de Signal : Modèles Physiques par Guide d'Onde.

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]

ares repluck iplk, kamp, icps, kpick, krefl, axcite

ares streson asig, kfr, ifdbgain

ares wgbow kamp, kfreq, kpres, krat, kvibf, kvamp, ifn [, iminfreq]

ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvel] \
      [, ibowpos] [, ilow]

ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn [, iminfreq]

ares wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq]

ares wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq] [, ijetrf] [, iendrf]

ares wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

ares wgpluck2 iplk, kamp, icps, kpick, krefl
```

E/S de Signal : E/S Fichier.

```
dumpk  ksig, ifilname, iformat, iprd

dumpk2 ksig1, ksig2, ifilname, iformat, iprd

dumpk3 ksig1, ksig2, ksig3, ifilname, iformat, iprd

dumpk4 ksig1, ksig2, ksig3, ksig4, ifilname, iformat, iprd

ficlose ihandle
ficlose Sfilename

fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [,...]

fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]

fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [,...]

ihandle fiopen ifilename, imode

fout ifilename, iformat, aout1 [, aout2, aout3,...,aoutN]

fouti ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]

foutir ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]

foutk ifilename, iformat, kout1 [, kout2, kout3,....,koutN]

fprintks "filename", "string", [, kval1] [, kval2] [...]

fprints "filename", "string" [, ival1] [, ival2] [...]

kres readk ifilname, iformat, iprd

kr1, kr2 readk2 ifilname, iformat, iprd

kr1, kr2, kr3 readk3 ifilname, iformat, iprd

kr1, kr2, kr3, kr4 readk4 ifilname, iformat, iprd
```

E/S de Signal : Entrée de Signal.

```
ar1 [, ar2 [, ar3 [, ... ar24]]] diskin ifilcod, kpitch [, iskiptim] \
      [, iwraparound] [, iformat] [, iskipinit]

a1[, a2[, ... a24]] diskin2 ifilcod, kpitch[, iskiptim \
      [, iwrap[, iformat [, iwsize[, ibufsize[, iskipinit]]]]]]

ar1 in

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
      ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \
      ar27, ar28, ar29, ar30, ar31, ar32 in32

ar1 inch ksig1

ar1, ar2, ar3, ar4, ar5, ar6 inh

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino

ar1, ar2,  ar3, a4 inq

inrg kstart, ain1 [,ain2, ain3, ..., ainN]

ar1, ar2 ins

kvalue invalue "channel name"
Sname invalue "channel name"

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \
      ar13, ar14, ar15, ar16 inx

inz ksig1

ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskptim] [, iformat] \
      [, iskipinit] [, ibufsize]
```

E/S de Signal : Sortie de Signal.

```
mdelay kstatus, kchan, kd1, kd2, kdelay

aout1 [,aout2 ... aoutX] monitor

out asig

out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \
      asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \
      asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \
      asig27, asig28, asig29, asig30, asig31, asig32

outc asig1 [, asig2] [...]

outch ksig1, asig1 [, ksig2] [, asig2] [...]

outh asig1, asig2, asig3, asig4, asig5, asig6

outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8

outq asig1, asig2, asig3, asig4

outq1 asig

outq2 asig

outq3 asig
```

```
outq4 asig

outrg kstart, aout1 [,aout2, aout3, ..., aoutN]

outs asig1, asig2

outs1 asig

outs2 asig

outvalue "channel name", kvalue
outvalue "channel name", "string"

outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \
      asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16

outz ksig1

soundout  asig1, ifilcod [, iformat]

soundouts  asigl, asigr, ifilcod [, iformat]
```

E/S de Signal : Bus Logiciel.

```
kval chani kchan
aval chani kchan

chano kval, kchan
chano aval, kchan

 chn_k Sname, imode[, itype, idflt, imin, imax]
 chn_a Sname, imode
 chn_S Sname, imode

chnclear Sname

gival chnexport Sname, imode[, itype, idflt, imin, imax]
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
gaval chnexport Sname, imode
gSval chnexport Sname, imode

ival chnget Sname
kval chnget Sname
aval chnget Sname
Sval chnget Sname

chnmix aval, Sname

itype, imode, ictltype, idflt, imin, imax chnparams

chnset ival, Sname
chnset kval, Sname
chnset aval, Sname
chnset Sval, Sname

setksmps iksmps

xinarg1 [, xinarg2] ... [xinargN] xin

xout xoutarg1 [, xoutarg2] ... [, xoutargN]
```

E/S de Signal : Impression et Affichage.

```
dispfft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg]

display xsig, iprd [, inprds] [, iwtflg]
```

```
flashtxt  iwhich, String

print iarg [, iarg1] [, iarg2] [...]

printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]

printk itime, kval [, ispace]

printk2 kvar [, inumspaces]

printks "string", itime [, kval1] [, kval2] [...]

prints "string" [, kval1] [, kval2] [...]
```

E/S de Signal : Requêtes sur les Fichiers Sons.

```
ir filelen ifilcod, [iallowraw]

ir filenchnls ifilcod [, iallowraw]

ir filepeak ifilcod [, ichnl]

ir filesr ifilcod [, iallowraw]
```

Modificateurs de Signal : Modificateurs d'Amplitude.

```
ares balance asig, acomp [, ihp] [, iskip]

ares clip asig, imeth, ilimit [, iarg]

ar compress aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook

ares dam asig, kthreshold, icomp1, icomp2, irtime, iftime

ares gain asig, krms [, ihp] [, iskip]
```

Modificateurs de Signal : Convolution et Morphing.

```
ar1 [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]

ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias

ares dconv asig, isize, ifn

a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskipsamples \
      [, iirlen[, iskipinit]]]

ftmorf kftndx, iftfn, iresfn

ar1 [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitionsize, ichannel]
```

Modificateurs de Signal : Retard.

```
ares delay asig, idlt [, iskip]

ares delay1 asig [, iskip]
```

```
kr delayk   ksig, idel[, imode]
kr vdel_k   ksig, kdel, imdel[, imode]

ares delayr idlt [, iskip]

delayw asig

ares deltap kdlt

ares deltap3 xdlt

ares deltapi xdlt

ares deltapn xnumsamps

aout deltapx adel, iwsize

deltapxw ain, adel, iwsize

ares multitap asig [, itime1] [, igain1] [, itime2] [, igain2] [...]

ares vdelay asig, adel, imaxdel [, iskip]

ares vdelay3 asig, adel, imaxdel [, iskip]

aout vdelayx ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]

aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]

aout vdelayxw ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \
      imd, iws [, ist]

aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```

Modificateurs de Signal : Panning et Spatialisation.

```
ao1, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
      [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
      au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
      at, au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
      [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]

ao1, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
      [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
      au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
      at, au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
      [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]

aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \
      kord0, kord1 , kord2
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \
      asig, kalpha, kbeta, kord0, kord1, kord2, kord3

aw, ax, ay, az bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \
      asig, kalpha, kbeta

aleft, aright hrtfer asig, kaz, kelev, HRTFcompact
```

```
aleft, aright hrtfmove asrc, kAz, kElev, ifilel, ifiler [, imode, ifade, isr]

aleft, aright hrtfmove2 asrc, kAz, kElev, ifilel, ifiler [,ioverlap, iradius, isr]

      aleft, aright hrtfstat asrc, iAz, iElev, ifilel, ifiler [,iradius, isr]

a1, a2 locsend
a1, a2,  a3, a4 locsend

a1, a2 locsig asig, kdegree, kdistance, kreverbsend
a1, a2,  a3, a4 locsig asig, kdegree, kdistance, kreverbsend

a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]

a1, a2 pan2 asig, xp [, imode]

a1, a2, a3, a4  space asig, ifn, ktime, kreverbsend, kx, ky

aW, aX, aY, aZ spat3d ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]

aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]

spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]

k1 spdist ifn, ktime, kx, ky

a1, a2, a3, a4 spsend

ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]

ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]

ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]

ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]

ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]

ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
      [, ifld2] [...]

vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]

vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]

vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
      ifld2, [...]
```

Modificateurs de Signal : Réverbération.

```
ares alpass asig, krvt, ilpt [, iskip] [, insmps]

a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]

ares comb asig, krvt, ilpt [, iskip] [, insmps]

aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]

ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \
      [, idel3] [, igain3] [, istor]

ares nreverb asig, ktime, khdif [, iskip] [,inumCombs] [, ifnCombs] \
      [, inumAlpas] [, ifnAlpas]

ares reverb asig, krvt [, iskip]
```

```
ares reverb2 asig, ktime, khdif [, iskip] [,inumCombs] \
      [, ifnCombs] [, inumAlpas] [, ifnAlpas]

aoutL, aoutR reverbsc ainL, ainR, kfblvl, kfco[, israte[, ipitchm[, iskip]]]

ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]

ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

Modificateurs de Signal : Opérateurs du Niveau Echantillon.

```
denorm a1[, a2[, a3[, ... ]]]

ares diff asig [, iskip]
kres diff ksig [, iskip]

kres downsamp asig [, iwlen]

ares fold asig, kincr

ares integ asig [, iskip]
kres integ ksig [, iskip]

ares interp ksig [, iskip] [, imode]

ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]

a(x) (arguments de taux-k seulement)

i(x) (arguments de taux-k seulement)

k(x) (arguments de taux-i seulement)

ares samphold asig, agate [, ival] [, ivstor]
kres samphold ksig, kgate [, ival] [, ivstor]

ares upsamp ksig

kval vaget kndx, avar

vaset kval, kndx, avar
```

Modificateurs de Signal : Limiteurs de Signal.

```
ares limit asig, klow, khigh
ires limit isig, ilow, ihigh
kres limit ksig, klow, khigh

ares mirror asig, klow, khigh
ires mirror isig, ilow, ihigh
kres mirror ksig, klow, khigh

ares wrap asig, klow, khigh
ires wrap isig, ilow, ihigh
kres wrap ksig, klow, khigh
```

Modificateurs de Signal : Effets Spéciaux.

```
ar distort asig, kdist, ifn[, ihp, istor]
```

```
ares distort1 asig, kpregain, kpostgain, kshape1, kshape2[, imode]

ares flanger asig, adel, kfeedback [, imaxd]

ares harmon asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \
      iminfrq, iprd

ares harmon2 asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]
ares harmon3 asig, koct, kfrq1, \
    kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]
ares harmon4 asig, koct, kfrq1, \
    kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]

ares phaser1 asig, kfreq, kord, kfeedback [, iskip]

ares phaser2 asig, kfreq, kq, kord, kmode, ksep, kfeedback
```

Modificateurs de Signal : Filtres Standard.

```
ares atone asig, khp [, iskip]

ares atonex asig, khp [, inumlayer] [, iskip]

ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]

ares biquada asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]

ares butbp asig, kfreq, kband [, iskip]

ares butbr asig, kfreq, kband [, iskip]

ares buthp asig, kfreq [, iskip]

ares butlp asig, kfreq [, iskip]

ares butterbp asig, kfreq, kband [, iskip]

ares butterbr asig, kfreq, kband [, iskip]

ares butterhp asig, kfreq [, iskip]

ares butterlp asig, kfreq [, iskip]

ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]

aout mode ain, kfreq, kQ [, iskip]

ares tone asig, khp [, iskip]

ares tonex  asig, khp [, inumlayer] [, iskip]
```

Modificateurs de Signal : Filtres Standard : Résonants.

```
ares areson asig, kcf, kbw [, iscl] [, iskip]

ares bqrez asig, xfco, xres [, imode] [, iskip]

ares lowpass2 asig, kcf, kq [, iskip]

ares lowres asig, kcutoff, kresonance [, iskip]

ares lowresx asig, kcutoff, kresonance [, inumlayer] [, iskip]

ares lpf18 asig, kfco, kres, kdist
```

```
asig moogladder ain, kcf, kres[, istor]

ares moogvcf asig, xfco, xres [,iscale, iskip]

ares moogvcf2 asig, xfco, xres [,iscale, iskip]

ares reson asig, kcf, kbw [, iscl] [, iskip]

ares resonr asig, kcf, kbw [, iscl] [, iskip]

ares resonx asig, kcf, kbw [, inumlayer] [, iscl] [, iskip]

ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]

ares resonz asig, kcf, kbw [, iscl] [, iskip]

ares rezzy asig, xfco, xres [, imode, iskip]

ahp,alp,abp,abr statevar ain, kcf, kq [, iosamps, istor]

alow, ahigh, aband svfilter  asig, kcf, kq [, iscl]

ares tbvcf asig, xfco, xres, kdist, kasym [, iskip]

ares vlowres asig, kfco, kres, iord, ksep
```

Modificateurs de Signal : Filtres Standard : Contrôle.

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]

kres atonek ksig, khp [, iskip]

kres lineto ksig, ktime

kres port ksig, ihtim [, isig]

kres portk ksig, khtim [, isig]

kres resonk ksig, kcf, kbw [, iscl] [, iskip]

kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]

kres tlineto ksig, ktime, ktrig

kres tonek ksig, khp [, iskip]
```

Modificateurs de Signal : Filtres Spécialisés.

```
ares dcblock ain [, igain]

ares dcblock2 ain [, iorder] [, iskip]

asig eqfil ain, kcf, kbw, kgain[, istor]

ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN

asig fofilter ain, kcf, kris, kdec[, istor]

ar1, ar2 hilbert asig

ares nlfilt ain, ka, kb, kd, kC, kL

ares pareq asig, kc, kv, kq [, imode] [, iskip]
```

```
ar rbjeq asig, kfco, klvl, kQ, kS[, imode]

ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \
      ia1,ia2, ..., iaN
```

Modificateurs de Signal : Guides d'Onde.

```
ares wguide1 asig, xfreq, kcutoff, kfeedback

ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \
      kfeedback1, kfeedback2
```

Modificateurs de Signal : Distorsion Non-Linéaire.

```
aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]

aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]

aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]

aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]

aout powershape ain, kShapeAmount [, ifullscale]
```

Modificateurs de Signal : Comparateurs et Accumulateurs.

```
amax max ain1 [, ain2] [, ain3] [, ain4] [...]
kmax max kin1 [, kin2] [, kin3] [, kin4] [...]

knumkout max_k asig, ktrig, itype

amax maxabs ain1 [, ain2] [, ain3] [, ain4] [...]
kmax maxabs kin1 [, kin2] [, kin3] [, kin4] [...]

maxabsaccum aAccumulator, aInput

maxaccum aAccumulator, aInput

amin min ain1 [, ain2] [, ain3] [, ain4] [...]
kmin min kin1 [, kin2] [, kin3] [, kin4] [...]

amin minabs ain1 [, ain2] [, ain3] [, ain4] [...]
kmin minabs kin1 [, kin2] [, kin3] [, kin4] [...]

minabsaccum aAccumulator, aInput

minaccum aAccumulator, aInput
```

Contrôle d'Instrument : Contrôle d'Horloge.

```
clockoff inum

clockon inum
```

Contrôle d'Instrument : Valeurs Conditionnelles.

```
(a == b ? v1 : v2)

(a >= b ? v1 : v2)

(a >  b ? v1 : v2)

(a <= b ? v1 : v2)

(a <  b ? v1 : v2)

(a != b ? v1 : v2)
```

Contrôle d'Instrument : Contrôle de Durée.

```
ihold

turnoff

turnoff2 kinsno, kmode, krelease


turnon insnum [, itime]
```

Contrôle d'Instrument : Appel d'Instrument.

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]

event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]

mute insnum [, iswitch]
mute "insname" [, iswitch]

remove insnum

schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \
      [, ip4] [, ip5] [...]
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \
      [, ip4] [, ip5] [...]

schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
      [, ip4] [, ip5] [...]

schedule insnum, iwhen, idur [, ip4] [, ip5] [...]
schedule "insname", iwhen, idur [, ip4] [, ip5] [...]

schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]

scoreline Sin, ktring

scoreline_i Sin
```

Contrôle d'Instrument : Contrôle Séquentiel d'un Programme.

```
cggoto condition, label
```

```
cigoto condition, label

ckgoto condition, label

cngoto condition, label

else

elseif xa R xb then

endif

goto label

if ia R ib igoto label
if ka R kb kgoto label
if xa R xb goto label
if xa R xb then

igoto label

kgoto label

loop_ge   indx, idecr, imin, label
loop_ge   kndx, kdecr, kmin, label

loop_gt   indx, idecr, imin, label
loop_gt   kndx, kdecr, kmin, label

loop_le   indx, incr, imax, label
loop_le   kndx, kncr, kmax, label

loop_lt   indx, incr, imax, label
loop_lt   kndx, kncr, kmax, label

tigoto label

timout istrt, idur, label
```

Contrôle d'Instrument : Controle de l'Exécution en Temps Réel.

```
ir active insnum
kres active kinsnum

cpuprc insnum, ipercent

      exitnow

jacktransport icommand [, ilocation]

maxalloc insnum, icount

prealloc insnum, icount
prealloc "insname", icount
```

Contrôle d'Instrument : Initialisation et Réinitialisation.

```
ares = xarg
ires = iarg
kres = karg

ares init iarg
ires init iarg
kres init iarg

insno nstrnum "name"
```

```
p(x)

pset icon1 [, icon2] [...]

reinit label

rigoto label

rireturn

ir tival
```

Contrôle d'Instrument : Détection et Contrôle.

```
kres button knum

ktrig changed kvar1 [, kvar2,..., kvarN]

kres checkbox knum

kres control knum

ares follow asig, idt

ares follow2 asig, katt, krel

Svalue getcfg iopt

ktrig  metro  kfreq [, initphase]

ksig  miditempo

icount pcount

kres peak asig
kres peak ksig

ivalue pindex ipfieldIndex

koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh [, ifrqs] [, iconf] \
      [, istrt] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]

kcps, krms pitchamdf asig, imincps, imaxcps [, icps] [, imedi] \
      [, idowns] [, iexcps] [, irmsmedi]

kcps, kamp ptrack asig, ihopsize[,ipeaks]

 rewindscore

kres rms asig [, ihp] [, iskip]

kres[, kkeydown] sensekey

ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times

setctrl inum, ival, itype

 setscorepos  ipos

splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]

ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \
      istartempo, ifn [, idisprd] [, itweek]

tempo ktempo, istartempo
```

```
kres tempoval

ktrig  timedseq  ktimpnt, ifn, kp1 [,kp2, kp3, ...,kpN]

kout trigger ksig, kthreshold, kmode

trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]

kx, ky xyin iprd, ixmin, ixmax, iymin, iymax [, ixinit] [, iyinit]
```

Contrôle d'Instrument : Piles.

```
xval1, [xval2, ... , xval31] pop
ival1, [ival2, ... , ival31] pop

fsig pop_f

push  xval1, [xval2, ... , xval31]
push  ival1, [ival2, ... , ival31]

push_f  fsig

stack  iStackSize
```

Contrôle d'Instrument : Contrôle de sous-instrument.

```
a1, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]
a1, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]

subinstrinit instrnum [, p4] [, p5] [...]
subinstrinit "insname" [, p4] [, p5] [...]
```

Contrôle d'Instrument : Lecture du Temps.

```
ir date

Sir dates [ itime]

ir readclock inum

ires rtclock
kres rtclock

kres timeinstk
kres timeinsts

kres timeinsts

ires timek
kres timek

ires times
kres times
```

Contrôle des Tables de Fonction.

```
ftfree ifno, iwhen
```

```
gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]

ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...

sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istrt   \
      [, ilpmod[, ilps[, ilpe]]]]]]]]]
```

Contrôle des Tables de Fonction : Requêtes sur une Table.

```
ftchnls(x) (init-rate args only)

ftlen(x) (init-rate args only)

ftlptim(x) (init-rate args only)

ftsr(x) (init-rate args only)

nsamp(x) (init-rate args only)

ires tableng ifn
kres tableng kfn

tb0_init ifn
tb1_init ifn
tb2_init ifn
tb3_init ifn
tb4_init ifn
tb5_init ifn
tb6_init ifn
tb7_init ifn
tb8_init ifn
tb9_init ifn
tb10_init ifn
tb11_init ifn
tb12_init ifn
tb13_init ifn
tb14_init ifn
tb15_init ifn
iout = tb0(iIndex)
kout = tb0(kIndex)
iout = tb1(iIndex)
kout = tb1(kIndex)
iout = tb2(iIndex)
kout = tb2(kIndex)
iout = tb3(iIndex)
kout = tb3(kIndex)
iout = tb4(iIndex)
kout = tb4(kIndex)
iout = tb5(iIndex)
kout = tb5(kIndex)
iout = tb6(iIndex)
kout = tb6(kIndex)
iout = tb7(iIndex)
kout = tb7(kIndex)
iout = tb8(iIndex)
kout = tb8(kIndex)
iout = tb9(iIndex)
kout = tb9(kIndex)
iout = tb10(iIndex)
kout = tb10(kIndex)
iout = tb11(iIndex)
kout = tb11(kIndex)
iout = tb12(iIndex)
kout = tb12(kIndex)
iout = tb13(iIndex)
kout = tb13(kIndex)
iout = tb14(iIndex)
kout = tb14(kIndex)
iout = tb15(iIndex)
kout = tb15(kIndex)
```

Contrôle des Tables de Fonction : Sélection Dynamique.

```
ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tableikt kndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tablekt kndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablexkt xndx, kfn, kwarp, iwsize [, ixmode] [, ixoff] [, iwrap]
```

Contrôle des Tables de Fonction : Opérations de Lecture/Ecriture.

```
ftload "filename", iflag, ifn1 [, ifn2] [...]

ftloadk "filename", ktrig, iflag, ifn1 [, ifn2] [...]

ftsave "filename", iflag, ifn1 [, ifn2] [...]

ftsavek "filename", ktrig, iflag, ifn1 [, ifn2] [...]

tablecopy kdft, ksft

tablegpw kfn

tableicopy idft, isft

tableigpw ifn

tableimix idft, idoff, ilen, is1ft, is1off, is1g, is2ft, is2off, is2g

tableiw isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]

tablemix kdft, kdoff, klen, ks1ft, ks1off, ks1g, ks2ft, ks2off, ks2g

ares tablera kfn, kstart, koff

tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmode]
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmode]

kstart tablewa kfn, asig, koff

tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwgmode]
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwgmode]

kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

aout tabmorpha aindex, aweightpoint, atabnum1, atabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \
      ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

tabplay  ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]

tabrec   ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]
```

FLTK : Conteneurs.

```
FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]
```

FLgroupEnd

FLpack iwidth, iheight, ix, iy, itype, ispace, iborder

FLpackEnd

FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]

FLpanelEnd

FLscroll iwidth, iheight [, ix] [, iy]

FLscrollEnd

FLtabs iwidth, iheight, ix, iy

FLtabsEnd

FLTK : Valuateurs.

```
kout, ihandle FLcount "label", imin, imax, istep1, istep2, itype, \
      iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]
```

```
koutx, kouty, ihandlex, ihandley FLjoy "label", iminx, imaxx, iminy, \
      imaxy, iexpx, iexpy, idispx, idispy, iwidth, iheight, ix, iy
```

```
kout, ihandle FLknob "label", imin, imax, iexp, itype, idisp, iwidth, \
      ix, iy [, icursorsize]
```

```
kout, ihandle FLroller "label", imin, imax, istep, iexp, itype, idisp, \
      iwidth, iheight, ix, iy
```

```
kout, ihandle FLslider "label", imin, imax, iexp, itype, idisp, iwidth, \
      iheight, ix, iy
```

```
kout, ihandle FLtext "label", imin, imax, istep, itype, iwidth, \
      iheight, ix, iy
```

FLTK : Autres.

ihandle **FLbox** "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]

```
kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, \
      iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

```
kout, ihandle FLbutton "label", ion, ioff, itype, iwidth, iheight, ix, \
      iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

ihandle **FLcloseButton** "label", iwidth, iheight, ix, iy

ihandle **FLexecButton** "command", iwidth, iheight, ix, iy

inumsnap **FLgetsnap** index [, igroup]

ihandle **FLhvsBox** inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]

FLhvsBox kx, ky, ihandle

kascii **FLkeyIn** [ifn]

FLloadsnap "filename" [, igroup]

kx, ky, kb1, kb2, kb3 **FLmouse** [, imode]

FLprintk itime, kval, idisp

FLprintk2 kval, idisp

FLrun

FLsavesnap "filename" [, igroup]

inumsnap, inumval **FLsetsnap** index [, ifn, igroup]

FLsetSnapGroup igroup

FLsetVal ktrig, kvalue, ihandle

FLsetVal_i ivalue, ihandle

```
FLslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
      [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

FLslidBnk2 "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart_index]

FLslidBnk2Set ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLslidBnk2Setk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]

ihandle **FLslidBnkGetHandle**

FLslidBnkSet ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLslidBnkSetk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLupdate

ihandle **FLvalue** "label", iwidth, iheight, ix, iy

FLvkeybd "keyboard.map", iwidth, iheight, ix, iy

```
FLvslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
      [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

FLvslidBnk2 "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart_index]

```
koutx, kouty, kinside  FLxyin ioutx_min, ioutx_max, iouty_min, iouty_max, \
      iwindx_min, iwindx_max, iwindy_min, iwindy_max [, iexpx, iexpy, ioutx, iouty]
```

```
vphaseseg kphase, ioutab, ielems, itab1,idist1,itab2 \
      [,idist2,itab3, ... ,idistN-1,itabN]
```

FLTK : Apparence.

FLcolor ired, igreen, iblue [, ired2, igreen2, iblue2]

FLcolor2 ired, igreen, iblue

FLhide ihandle

FLlabel isize, ifont, ialign, ired, igreen, iblue

FLsetAlign ialign, ihandle

FLsetBox itype, ihandle

FLsetColor ired, igreen, iblue, ihandle

FLsetColor2 ired, igreen, iblue, ihandle

FLsetFont ifont, ihandle

FLsetPosition ix, iy, ihandle

FLsetSize iwidth, iheight, ihandle

FLsetText "itext", ihandle

FLsetTextColor ired, iblue, igreen, ihandle

FLsetTextSize isize, ihandle

FLsetTextType itype, ihandle

FLshow ihandle

Opérations Mathématiques : Comparateurs et Accumulateurs.

clear avar1 [, avar2] [, avar3] [...]

vincr accum, aincr

Opérations Mathématiques : Opérations Arithmétiques et Logiques.

a **+** b (no rate restriction)

a **/** b (no rate restriction)

a **%** b (no rate restriction)

a ***** b (no rate restriction)

a **&&** b (ET logique ; pas de taux audio)

a **&** b (ET binaire)

~ a (NON binaire)

a **|** b (bitwise OR)

a **<<** b (bitshift left)

a **>>** b (bitshift left)

a **#** b (NON-EQUIVALENCE binaire)

a **||** b (logical OR; not audio-rate)

a **^** b (b not audio-rate)

a **#** b (no rate restriction)

Opérations Mathématiques : Fonctions Mathématiques.

abs(x) (pas de restriction de taux)

ceil(x) (argument au taux d'initialisation, de contrôle ou audio)

exp(x) (pas de restriction de taux)

floor(x) (argument au taux d'initialisation, de contrôle ou audio)

frac(x) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)

int(x) (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)

log(x) (pas de restriction de taux)

log10(x) (pas de restriction de taux)

logbtwo(x) (argument au taux d'initialisation ou de contrôle seulement)

powoftwo(x) (argument au taux d'initialisation ou de contrôle seulement)

round(x) (des arguments de taux-i, -k ou -a sont permis)

sqrt(x) (pas de restriction de taux)

Opérations Mathématiques : Fonctions Trigonométriques.

cos(x) (pas de restriction de taux)

cosh(x) (pas de restriction de taux)

cosinv(x) (pas de restriction de taux)

sin(x) (pas de restriction de taux)

sinh(x) (pas de restriction de taux)

sininv(x) (pas de restriction de taux)

tan(x) (pas de restriction de taux)

tanh(x) (pas de restriction de taux)

taninv(x) (pas de restriction de taux)

Opérations Mathématiques : Fonctions d'Amplitude.

ampdb(x) (pas de restriction de taux)

ampdbfs(x) (pas de restriction de taux)

db(x)

dbamp(x) (arguments de taux-i ou -k seulement)

dbfsamp(x) (arguments de taux-i ou -k seulement)

Opérations Mathématiques : Fonctions aléatoires.

birnd(x) (taux-i ou -k seulement)

rnd(x) (taux-i ou -k seulement)

Opérations Mathématiques : Opcodes Equivalents à des Fonctions.

```
ares divz xa, xb, ksubst
ires divz ia, ib, isubst
kres divz ka, kb, ksubst
```

```
ares mac asig1, ksig1 [, asig2] [, ksig2] [, asig3] [, ksig3] [...]
```

```
ares maca asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]

aout polynomial ain, k0 [, k1 [, k2 [...]]]

ares pow aarg, kpow [, inorm]
ires pow iarg, ipow [, inorm]
kres pow karg, kpow [, inorm]

ares product asig1, asig2 [, asig3] [...]

ares sum asig1 [, asig2] [, asig3] [...]

ares taninv2 ay, ax
ires taninv2 iy, ix
kres taninv2 ky, kx
```

Conversion des Hauteurs : Fonctions.

```
cent(x)

cpsmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)

cpsoct (oct)  (pas de restriction de taux)

cpspch (pch)  (arguments de taux-i ou -k seulement)

octave(x)

octcps (cps)  (arguments de taux-i ou -k seulement)

octmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)

octpch (pch)  (arguments de taux-i ou -k seulement)

pchmidinn (MidiNoteNumber)  (arguments de taux-i ou -k seulement)

pchoct (oct)  (arguments de taux-i ou -k seulement)

semitone(x)
```

Conversion des Hauteurs : Opcodes de Hauteurs.

```
icps cps2pch ipch, iequal

kcps cpstun ktrig, kindex, kfn

icps cpstuni index, ifn

icps cpsxpch ipch, iequal, irepeat, ibase
```

MIDI en Temps-Réel : Entrée.

```
kaft aftouch [imin] [, imax]

ival chanctrl ichnl, ictlno [, ilow] [, ihigh]
kval chanctrl ichnl, ictlno [, ilow] [, ihigh]

idest ctrl14 ichan, ictlno1, ictlno2, imin, imax [, ifn]
kdest ctrl14 ichan, ictlno1, ictlno2, kmin, kmax [, ifn]

idest ctrl21 ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
kdest ctrl21 ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

```
idest ctrl7 ichan, ictlno, imin, imax [, ifn]
kdest ctrl7 ichan, ictlno, kmin, kmax [, ifn]
adest ctrl7 ichan, ictlno, kmin, kmax [, ifn] [, icutoff]

ctrlinit ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \
      [, ival3] [,...ival32]

initc14 ichan, ictlno1, ictlno2, ivalue

initc21 ichan, ictlno1, ictlno2, ictlno3, ivalue

initc7 ichan, ictlno, ivalue

massign ichnl, insnum[, ireset]
massign ichnl, "insname"[, ireset]

idest midic14 ictlno1, ictlno2, imin, imax [, ifn]
kdest midic14 ictlno1, ictlno2, kmin, kmax [, ifn]

idest midic21 ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
kdest midic21 ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]

idest midic7 ictlno, imin, imax [, ifn]
kdest midic7 ictlno, kmin, kmax [, ifn]

ival midictrl inum [, imin] [, imax]
kval midictrl inum [, imin] [, imax]

ival notnum

ibend pchbend [imin] [, imax]
kbend pchbend [imin] [, imax]

pgmassign ipgm, inst[, ichn]
pgmassign ipgm, "insname"[, ichn]

ires polyaft inote [, ilow] [, ihigh]
kres polyaft inote [, ilow] [, ihigh]

ival veloc [ilow] [, ihigh]
```

MIDI en Temps-Réel : Sortie.

```
nrpn kchan, kparmnum, kparmvalue

outiat ichn, ivalue, imin, imax

outic ichn, inum, ivalue, imin, imax

outic14 ichn, imsb, ilsb, ivalue, imin, imax

outipat ichn, inotenum, ivalue, imin, imax

outipb ichn, ivalue, imin, imax

outipc ichn, iprog, imin, imax

outkat kchn, kvalue, kmin, kmax

outkc kchn, knum, kvalue, kmin, kmax

outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax

outkpat kchn, knotenum, kvalue, kmin, kmax

outkpb kchn, kvalue, kmin, kmax

outkpc kchn, kprog, kmin, kmax
```

MIDI en Temps-Réel : Convertisseurs.

```
iamp ampmidi iscal [, ifn]
```

```
icps cpsmidi
```

```
icps cpsmidib [irange]
kcps cpsmidib [irange]
```

```
icps cpstmid ifn
```

```
ioct octmidi
```

```
ioct octmidib [irange]
koct octmidib [irange]
```

```
ipch pchmidi
```

```
ipch pchmidib [irange]
kpch pchmidib [irange]
```

MIDI en Temps-Réel : E/S Génériques.

```
kstatus, kchan, kdata1, kdata2 midiin
```

```
midiout kstatus, kchan, kdata1, kdata2
```

MIDI en Temps-Réel : Extension d'Evènements.

```
kflag release
```

```
xtratim iextradur
```

MIDI en Temps-Réel : Sortie de Note.

```
midion kchn, knum, kvel
```

```
midion2 kchn, knum, kvel, ktrig
```

```
moscil kchn, knum, kvel, kdur, kpause
```

```
noteoff ichn, inum, ivel
```

```
noteon ichn, inum, ivel
```

```
noteondur ichn, inum, ivel, idur
```

```
noteondur2 ichn, inum, ivel, idur
```

MIDI en Temps-Réel : Interopérabilité MIDI/Partition.

```
midichannelaftertouch xchannelaftertouch [, ilow] [, ihigh]
```

```
ichn midichn

midicontrolchange xcontroller, xcontrollervalue [, ilow] [, ihigh]

mididefault xdefault, xvalue

midinoteoff xkey, xvelocity

midinoteoncps xcps, xvelocity

midinoteonkey xkey, xvelocity

midinoteonoct xoct, xvelocity

midinoteonpch xpch, xvelocity

midipitchbend xpitchbend [, ilow] [, ihigh]

midipolyaftertouch xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]

midiprogramchange xprogram
```

MIDI en Temps-Réel : System Realtime.

```
mclock ifreq

mrtmsg imsgtype
```

MIDI en Temps-Réel : Banques de Réglettes.

```
i1,...,i16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
ifn16
k1,...,k16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
ifn16

i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
ifn32
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
      initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
ifn32

i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum16, imin16, imax16, init16, ifn16
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum16, imin16, imax16, init16, ifn16

k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
      icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16

kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, .... , ictlnum16, imin16, imax16, init16, ifn16

kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16

i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum32, imin32, imax32, init32, ifn32
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum32, imin32, imax32, init32, ifn32

k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
      ..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32

kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \
```

```
      imax1, init1, ifn1, .... , ictlnum32, imin32, imax32, init32, ifn32

kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32

i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum64, imin64, imax64, init64, ifn64
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum64, imin64, imax64, init64, ifn64

k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
      icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64

kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \
      imax1, init1, ifn1, .... , ictlnum64, imin64, imax64, init64, ifn64

kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64

i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum8, imin8, imax8, init8, ifn8
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
      ictlnum8, imin8, imax8, init8, ifn8

k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
      ..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8

kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8

kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
      init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8

k1, k2, ...., k16 sliderKawai imin1, imax1, init1, ifn1, \
      imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16
```

Traitement Spectral : STFT.

```
ktableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

ares pvadd ktimpnt, kfmod, ifilcod, ifn, ibins [, ibinoffset] \
      [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]

pvbufread ktimpnt, ifile

ares pvcross ktimpnt, kfmod, ifile, kampscale1, kampscale2 [, ispecwp]

ares pvinterp ktimpnt, kfmod, ifile, kfreqscale1, kfreqscale2, \
      kampscale1, kampscale2, kfreqinterp, kampinterp

ares pvoc ktimpnt, kfmod, ifilcod [, ispecwp] [, iextractmode] \
      [, ifreqlim] [, igatefn]

kfreq, kamp pvread ktimpnt, ifile, ibin

tableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

tablexseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

ares vpvoc ktimpnt, kfmod, ifile [, ispecwp] [, ifn]
```

Traitement Spectral : LPC.

```
ares lpfreson asig, kfrqratio

lpinterp islot1, islot2, kmix
```

```
krmsr, krmso, kerr, kcps lpread ktimpnt, ifilcod [, inpoles] [, ifrmrate]

ares lpreson asig

lpslot islot
```

Traitement Spectral : Non-Standard.

```
wsig specaddm wsig1, wsig2 [, imul2]

wsig specdiff wsigin

specdisp wsig, iprd [, iwtflg]

wsig specfilt wsigin, ifhtim

wsig spechist wsigin

koct, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \
      irolloff [, iodd] [, iconfs] [, interp] [, ifprd] [, iwtflg]

wsig specscal wsigin, ifscale, ifthresh

ksum specsum wsig [, interp]

wsig spectrum xsig, iprd, iocts, ifrqa [, iq] [, ihann] [, idbout] \
      [, idsprd] [, idsinrs]
```

Traitement Spectral : Streaming.

```
fsig binit fin, isize

ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks

ares pvsadsyn fsrc, inoscs, kfmod [, ibinoffset] [, ibinincr] [, iinit]

fsig pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]

fsig pvsarp fsigin, kbin, kdepth, kgain

fsig pvsbandp fsigin, xlowcut,
      xlowfull, xhighfull, xhighcut[, ktype]

fsig pvsbandr fsigin, xlowcut,
    xlowfull, xhighfull, xhighcut[, ktype]

kamp, kfr pvsbin fsig, kbin

fsig pvsblur fsigin, kblurtime, imaxdel

ihandle, ktime  pvsbuffer fsig, ilen

fsig pvsbufread  ktime, khandle[, ilo, ihi]

fsig pvscale fsigin, kscal[, ikeepform, igain]]

kcent pvscent fsig

fsig pvscross fsrc, fdest, kamp1, kamp2

fsig pvsdemix fleft, fright, kpos, kwidth, ipoints

fsig pvsdiskinSFname,ktscal,kgain[,ioffset, ichan]

pvsdisp fsig[, ibins, iwtflg]
```

fsig **pvsfilter** fsigin, fsigfil, kdepth[, igain]

fsig **pvsfread** ktimpt, ifn [, ichan]

fsig **pvsfreeze** fsigin, kfreeza, kfreezf

pvsftr fsrc, ifna [, ifnf]

kflag **pvsftw** fsrc, ifna [, ifnf]

pvsfwrite fsig, ifile

fsig **pvshift** fsigin, kshift, klowest[, ikeepform, igain]

ffr,fphs **pvsifd** ain, ifftsize, ihopsize, iwintype[,iscal]

fsig **pvsin** kchan[,isize,iolap,iwinsize,iwintype,iformat]

ioverlap, inumbins, iwinsize, iformat **pvsinfo** fsrc

fsig **pvsinit** isize[,iolap,iwinsize,iwintype, iformat]

fsig **pvsmaska** fsrc, ifn, kdepth

fsig **pvsmix** fsigin1, fsigin2

fsig **pvsmooth** fsigin, kacf, kfcf

fsig **pvsmorph** fsig1, fsig2, kampint, kfrqint

fsig **pvsosc** kamp, kfreq, ktype, isize [,ioverlap] [, iwinsize] [, iwintype] [, iformat]

pvsout fsig, kchan

kfr, kamp **pvspitch** fsig, kthresh

fsig **pvstencil** fsigin, kgain, klevel, iftable

fsig **pvsvoc** famp, fexc, kdepth, kgain

ares **pvsynth** fsrc, [iinit]

asig **resyn** fin, kscal, kpitch, kmaxtracks, ifn

asig **sinsyn** fin, kscal, kmaxtracks, ifn

asig **tradsyn** fin, kscal, kpitch, kmaxtracks, ifn

fsig **trcross** fin1, fin2, ksearch,kdepth[,kmode]

fsig **trfilter** fin, kamnt, ifn

fsig, kfr,kamp **trhighest** fin1, kscal

fsig, kfr,kamp **trlowest** fin1, kscal

fsig **trmix** fin1, fin2

fsig **trscale** fin, kpitch[, kgain]

fsig **trshift** fin, kpshift[, kgain]

fsiglow, fsighi **trsplit** fin, ksplit[, kgainlow, kgainhigh]

Traitement Spectral : ATS.

```
ar ATSadd ktimepnt, kfmod, iatsfile, ifn, ipartials[, ipartialoffset, \
        ipartialincr, igatefn]
```

```
ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]

ATSbufread ktimepnt, kfmod, iatsfile, ipartials[, ipartialoffset, \
          ipartialincr]

ar ATScross ktimepnt, kfmod, iatsfile, ifn, kmylev, kbuflev, ipartials \
          [, ipartialoffset, ipartialincr]

idata ATSinfo iatsfile, ilocation

kamp ATSinterpread kfreq

kfrq, kamp ATSpartialtap ipartialnum

kfreq, kamp ATSread ktimepnt, iatsfile, ipartial

kenergy ATSreadnz ktimepnt, iatsfile, iband

ar ATSsinnoi ktimepnt, ksinlev, knzlev, kfmod, iatsfile, ipartials \
          [, ipartialoffset, ipartialincr]
```

Traitement Spectral : Loris.

```
lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv

ar lorisplay ireadidx, kfreqenv, kampenv, kbwenv

lorisread ktimpnt, ifilcod, istoreidx, kfreqenv, kampenv, kbwenv[, ifadetime]
```

Chaînes : Définition.

```
Sdst strget indx

strset iarg, istring
```

Chaînes : Manipulation.

```
puts Sstr, ktrig[, inonl]

Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]

Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]

Sdst strcat Ssrc1, Ssrc2

Sdst strcatk Ssrc1, Ssrc2

ires strcmp S1, S2

kres strcmpk S1, S2

Sdst strcpy Ssrc
Sdst = Ssrc

Sdst strcpyk Ssrc

ipos strindex S1, S2

kpos strindexk S1, S2
```

```
ilen strlen Sstr

klen strlenk Sstr

ipos strrindex S1, S2

kpos strrindexk S1, S2

Sdst strsub Ssrc[, istart[, iend]]

Sdst strsubk Ssrc, kstart, kend
```

Chaînes : Conversion.

```
ichr strchar Sstr[, ipos]

kchr strchark Sstr[, kpos]

Sdst strlower Ssrc

Sdst strlowerk Ssrc

ir strtod Sstr
ir strtod indx

kr strtodk Sstr
kr strtodk kndx

ir strtol Sstr
ir strtol indx

kr strtolk Sstr
kr strtolk kndx

Sdst strupper Ssrc

Sdst strupperk Ssrc
```

Vectoriel : Tableaux.

```
vtaba  andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]

vtabi  indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]

vtabk  kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]

vtable1k  kfn,kout1 [, kout2, kout3, .... , koutN ]

vtablea  andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]

vtablei  indx, ifn, interp, ixmode, iout1 [, iout2, iout3, .... , ioutN ]

vtablek  kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]

vtablewa  andx, kfn, ixmode, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]

vtablewi  indx, ifn, ixmode, inarg1 [, inarg2, inarg3 , .... , inargN ]

vtablewk  kndx, kfn, ixmode, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]

vtabwa  andx, ifn, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]

vtabwi  indx, ifn, inarg1 [, inarg2, inarg3 , .... , inargN ]

vtabwk  kndx, ifn, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
```

Vectoriel : Opérations Scalaires.

vadd ifn, kval, kelements [, kdstoffset] [, kverbose]

vadd_i ifn, ival, ielements [, idstoffset]

vexp ifn, kval, kelements [, kdstoffset] [, kverbose]

vexp_i ifn, ival, ielements[, idstoffset]

vmult ifn, kval, kelements [, kdstoffset] [, kverbose]

vmult_i ifn, ival, ielements [, idstoffset]

vpow ifn, kval, kelements [, kdstoffset] [, kverbose]

vpow_i ifn, ival, ielements [, idstoffset]

Vectoriel : Opérations Vectorielles.

vaddv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vaddv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vcopy ifn, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]

vcopy_i ifn, ifn2, ielements [,idstoffset, isrcoffset]

vdivv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vdivv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vexpv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vexpv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vmap ifn1, ifn2, ielements [,idstoffset, isrcoffset]

vmultv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vmultv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vsubv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vsubv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

Vectoriel : Enveloppes.

vexpseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]

vlinseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]

Vectoriel : Limitation et Enroulement.

```
vlimit  ifn, kmin, kmax, ielements

vmirror  ifn, kmin, kmax, ielements

vwrap  ifn, kmin, kmax, ielements
```

Vectoriel : Chemins de Retard.

```
kout vdelayk  iksig, kdel, imaxdel [, iskip, imode]

vecdelay  ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]

vport ifn, khtime, ielements [, ifnInit]
```

Vectoriel : Aléatoire.

```
vrandh  ifn,  krange, kcps, ielements [, idstoffset] [, iseed]
      [, isize] [, ioffset]

vrandi  ifn,  krange, kcps, ielements [, idstoffset] [, iseed]
      [, isize] [, ioffset]
```

Vectoriel : Automates Cellulaires.

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \
       iRuleFunc, ielements, irulelen [, iradius]
```

Système de Patch Zak.

```
zacl kfirst, klast

zakinit isizea, isizek

ares zamod asig, kzamod

ares zar kndx

ares zarg kndx, kgain

zaw asig, kndx

zawm asig, kndx [, imix]

ir zir indx

ziw isig, indx

ziwm isig, indx [, imix]

zkcl kfirst, klast

kres zkmod ksig, kzkmod

kres zkr kndx

zkw ksig, kndx
```

```
zkwm ksig, kndx [, imix]
```

Accueil de Plugin : DSSI et LADSPA.

```
dssiactivate ihandle, ktoggle

aout1 [, aout2, aout3, aout4] dssiaudio ihandle, ain1 [,ain2, ain3, ain4]

dssictls ihandle, iport, kvalue, ktrigger

ihandle dssiinit ilibraryname, iplugindex [, iverbose]

dssilist
```

Accueil de Plugin : VST.

```
aout1,aout2 vstaudio instance, [ain1, ain2]
aout1,aout2 vstaudiog instance, [ain1, ain2]

vstbankload instance, ipath

vstedit instance

vstinfo instance

instance vstinit ilibrarypath [,iverbose]

vstmidiout instance, kstatus, kchan, kdata1, kdata2

vstnote instance, kchan, knote, kveloc, kdur

vstparamset instance, kparam, kvalue
kvalue vstparamget instance, kparam

vstprogset instance, kprogram
```

OSC.

```
ihandle OSCinit iport

kans OSClisten ihandle, idest, itype [, xdata1, xdata2, ...]

OSCsend kwhen, ihost, iport, idestination, itype [, kdata1, kdata2, ...]
```

Réseau.

```
remoteport iportnum

asig sockrecv iport, ilength
asigl, asigr sockrecvs iport, ilength
asig strecv Sipaddr, iport

socksend asig, Sipaddr, iport, ilength
socksends asigl, asigr, Sipaddr, iport,
    ilength
stsend asig, Sipaddr, iport
```

Opcodes pour le Traitement à Distance.

```
insglobalisource, instrnum [,instrnum...]
```

```
insremotidestination, isource, instrnum [,instrnum...]
```

```
midglobalisource, instrnum [,instrnum...]
```

```
midremotidestination, isource, instrnum [,instrnum...]
```

Opcodes Mixer.

```
MixerClear
```

```
kgain MixerGetLevel isend, ibuss
```

```
asignal MixerReceive ibuss, ichannel
```

```
MixerSend asignal, isend, ibuss, ichannel
```

```
MixerSetLevel isend, ibuss, kgain
```

Opcodes Python.

```
pyassign "variable", kvalue
pyassigni "variable", ivalue
pylassign "variable", kvalue
pylassigni "variable", ivalue
pyassignt ktrigger, "variable", kvalue
pylassignt ktrigger, "variable", kvalue
```

```
kresult pyeval "expression"
iresult pyevali "expression"
kresult pyleval "expression"
iresult pylevali "expression"
kresult pyevalt ktrigger, "expression"
kresult pylevalt ktrigger, "expression"
```

```
pyexec "filename"
pyexeci "filename"
pylexec "filename"
pylexeci "filename"
pyexect ktrigger, "filename"
plyexect ktrigger, "filename"
```

```
pyinit
```

```
pyrun "statement"
pyruni "statement"
pylrun "statement"
pylruni "statement"
pyrunt ktrigger, "statement"
pylrunt ktrigger, "statement"
```

Opcodes pour le Traitement d'Image.

```
iimagenum imagecreate iwidth, iheight
```

```
imagefree iimagenum
```

```
ared agreen ablue imagegetpixel iimagenum, ax, ay
kred kgreen kblue imagegetpixel iimagenum, kx, ky

iimagenum imageload filename

imagesave iimagenum, filename

imagegetpixel iimagenum, ax, ay, ared, agreen, ablue
imagegetpixel iimagenum, kx, ky, kred, kgreen, kblue

iwidth iheight imagesize iimagenum
```

Divers.

```
ires system_i itrig, Scmd, [inowait]
kres system ktrig, Scmd, [knowait]
```

Utilitaires.

```
csound -U atsa [options] nomfichier_entree nomfichier_sortie

cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]

csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]

csound -U cvanal [options] nomfichier_entree nomfichier_sortie
cvanal [options] nomfichier_entree nomfichier_sortie

dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree

envext [-options] fichierson
csound -U envext [-options] fichierson

extractor [OPTIONS ... ] fichierentree

het_export fichier_het fichier_textecsv
csound -U het_export fichier_het fichier_textecsv

het_import fichier_textecsv fichier_het
csound -U het_import fichier_textecsv fichier_het

csound -U hetro [options] nomfichier_entree nomfichier_sortie
hetro [options] nomfichier_entree nomfichier_sortie

csound -U lpanal [options] nomfichier_entree nomfichier_sortie
lpanal [options] nomfichier_entree nomfichier_sortie

makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]

mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...

pv_export fichier_pv fichier_texte_csv
csound -U pv_export fichier_pv fichier_texte_csv

pv_import fichier_texte_csv fichier_pv
csound -U pv_import fichier_texte_csv fichier_pv

csound -U pvanal [options] nomfic_entree nomfic_sortie
pvanal [options] nomfic_entree nomfic_sortie

csound -U pvlook [options] fichier_entree
pvlook [options] fichier_entree

scale [OPTIONS ... ] fichier

csound -U sdif2ad [options] fichier_entree fichier_sortie
```

```
csound -U sndinfo [options] fichierson ...
sndinfo [options] fichierson ...

srconv [options] fichier_entree
```

Annexe A. Conversion de Hauteur

Tableau A.1. Conversion de Hauteur

Note (anglais)	Note (français)	Hz	cpspch
C-1	do-2	8.176	3.00
C#-1	do#-2	8.662	3.01
D-1	ré-2	9.177	3.02
D#-1	ré#-2	9.723	3.03
E-1	mi-2	10.301	3.04
F-1	fa-2	10.913	3.05
F#-1	fa#-2	11.562	3.06
G-1	sol-2	12.250	3.07
G#-1	sol#-2	12.978	3.08
A-1	la-2	13.750	3.09
A#-1	la#-2	14.568	3.10
B-1	si-2	15.434	3.11
C0	do-1	16.352	4.00
C#0	do#-1	17.324	4.01
D0	ré-1	18.354	4.02
D#0	ré#-1	19.445	4.03
E0	mi-1	20.602	4.04
F0	fa-1	21.827	4.05
F#0	fa#-1	23.125	4.06
G0	sol-1	24.500	4.07
G#0	sol#-1	25.957	4.08
A0	la-1	27.500	4.09
A#0	la#-1	29.135	4.10
B0	si-1	30.868	4.11
C1	do0	32.703	5.00
C#1	do#0	34.648	5.01
D1	ré0	36.708	5.02
D#1	ré#0	38.891	5.03
E1	mi0	41.203	5.04
F1	fa0	43.654	5.05
F#1	fa#0	46.249	5.06
G1	sol0	48.999	5.07
G#1	sol#0	51.913	5.08
A1	la0	55.000	5.09
A#1	la#0	58.270	5.10
B1	si0	61.735	5.11

Note (anglais)	Note (français)	Hz	cpspch
C2	do1	65.406	6.00
C#2	do#1	69.296	6.01
D2	ré1	73.416	6.02
D#2	ré#1	77.782	6.03
E2	mi1	82.407	6.04
F2	fa1	87.307	6.05
F#2	fa#1	92.499	6.06
G2	sol1	97.999	6.07
G#2	sol#1	103.826	6.08
A2	la1	110.000	6.09
A#2	la#1	116.541	6.10
B2	si1	123.471	6.11
C3	do2	130.813	7.00
C#3	do#2	138.591	7.01
D3	ré2	146.832	7.02
D#3	ré#2	155.563	7.03
E3	mi2	164.814	7.04
F3	fa2	174.614	7.05
F#3	fa#2	184.997	7.06
G3	sol2	195.998	7.07
G#3	sol#2	207.652	7.08
A3	la2	220.000	7.09
A#3	la#2	233.082	7.10
B3	si2	246.942	7.11
C4	do3	261.626	8.00
C#4	do#3	277.183	8.01
D4	ré3	293.665	8.02
D#4	ré#3	311.127	8.03
E4	mi3	329.628	8.04
F4	fa3	349.228	8.05
F#4	fa#3	369.994	8.06
G4	sol3	391.995	8.07
G#4	sol#3	415.305	8.08
A4	la3	440.000	8.09
A#4	la#3	466.164	8.10
B4	si3	493.883	8.11
C5	do4	523.251	9.00
C#5	do#4	554.365	9.01
D5	ré4	587.330	9.02
D#5	ré#4	622.254	9.03
E5	mi4	659.255	9.04

Note (anglais)	Note (français)	Hz	cpspch
F5	fa4	698.456	9.05
F#5	fa#4	739.989	9.06
G5	sol4	783.991	9.07
G#5	sol#4	830.609	9.08
A5	la4	880.000	9.09
A#5	la#4	932.328	9.10
B5	si4	987.767	9.11
C6	do5	1046.502	10.00
C#6	do#5	1108.731	10.01
D6	ré5	1174.659	10.02
D#6	ré#5	1244.508	10.03
E6	mi5	1318.510	10.04
F6	fa5	1396.913	10.05
F#6	fa#5	1479.978	10.06
G6	sol5	1567.982	10.07
G#6	sol#5	1661.219	10.08
A6	la5	1760.000	10.09
A#6	la#5	1864.655	10.10
B6	si5	1975.533	10.11
C7	do6	2093.005	11.00
C#7	do#6	2217.461	11.01
D7	ré6	2349.318	11.02
D#7	ré#6	2489.016	11.03
E7	mi6	2637.020	11.04
F7	fa6	2793.826	11.05
F#7	fa#6	2959.955	11.06
G7	sol6	3135.963	11.07
G#7	sol#6	3322.438	11.08
A7	la6	3520.000	11.09
A#7	la#6	3729.310	11.10
B7	si6	3951.066	11.11
C8	do7	4186.009	12.00
C#8	do#7	4434.922	12.01
D8	ré7	4698.636	12.02
D#8	ré#7	4978.032	12.03
E8	mi7	5274.041	12.04
F8	fa7	5587.652	12.05
F#8	fa#7	5919.911	12.06
G8	sol7	6271.927	12.07
G#8	sol#7	6644.875	12.08
A8	la7	7040.000	12.09

Note (anglais)	Note (français)	Hz	cpspch
A#8	la#7	7458.620	12.10
B8	si7	7902.133	12.11
C9	do8	8372.018	13.00
C#9	do#8	8869.844	13.01
D9	ré8	9397.273	13.02
D#9	ré#8	9956.063	13.03
E9	mi8	10548.08	13.04
F9	fa8	11175.30	13.05
F#9	fa#8	11839.82	13.06
G9	sol8	12543.85	13.07

Annexe B. Valeurs d'Intensité du Son

Tableau B.1. Valeurs d'Intensité du Son (pour un ton pur à 1000 Hz)

Dynamiques	Intensité (W/m ^ 2)	Niveau (dB)
douleur	1	120
fff	10 ^ -2	100
f	10 ^ -4	80
p	10 ^ -6	60
ppp	10 ^ -8	40
seuil d'audibilité	10 ^ -12	0

Annexe C. Valeurs de Formant

Tableau C.1. alto « a »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2800	3500	4950
amp (dB)	0	-4	-20	-36	-60
larg. bande (Hz)	80	90	120	130	140

Tableau C.2. alto « e »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1600	2700	3300	4950
amp (dB)	0	-24	-30	-35	-60
larg. bande (Hz)	60	80	120	150	200

Tableau C.3. alto « i »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	1700	2700	3700	4950
amp (dB)	0	-20	-30	-36	-60
larg. bande (Hz)	50	100	120	150	200

Tableau C.4. alto « o »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3500	4950
amp (dB)	0	-9	-16	-28	-55
larg. bande (Hz)	70	80	100	130	135

Tableau C.5. alto « u »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2530	3500	4950
amp (dB)	0	-12	-30	-40	-64
larg. bande (Hz)	50	60	170	180	200

Tableau C.6. basse « a »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	600	1040	2250	2450	2750
amp (dB)	0	-7	-9	-9	-20
larg. bande (Hz)	60	70	110	120	130

Tableau C.7. basse « e »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1620	2400	2800	3100
amp (dB)	0	-12	-9	-12	-18
larg. bande (Hz)	40	80	100	120	120

Tableau C.8. basse « i »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	250	1750	2600	3050	3340
amp (dB)	0	-30	-16	-22	-28
larg. bande (Hz)	60	90	100	120	120

Tableau C.9. basse « o »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	750	2400	2600	2900
amp (dB)	0	-11	-21	-20	-40
larg. bande (Hz)	40	80	100	120	120

Tableau C.10. basse « u »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2400	2675	2950
amp (dB)	0	-20	-32	-28	-36
larg. bande (Hz)	40	80	100	120	120

Tableau C.11. haute-contre « a »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	660	1120	2750	3000	3350
amp (dB)	0	-6	-23	-24	-38
larg. bande (Hz)	80	90	120	130	140

Tableau C.12. haute-contre « e »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	440	1800	2700	3000	3300
amp (dB)	0	-14	-18	-20	-20
larg. bande (Hz)	70	80	100	120	120

Tableau C.13. haute-contre « i »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	1850	2900	3350	3590
amp (dB)	0	-24	-24	-36	-36
larg. bande (Hz)	40	90	100	120	120

Tableau C.14. haute-contre « o »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	430	820	2700	3000	3300
amp (dB)	0	-10	-26	-22	-34
larg. bande (Hz)	40	80	100	120	120

Tableau C.15. haute-contre « u »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	370	630	2750	3000	3400
amp (dB)	0	-20	-23	-30	-34
larg. bande (Hz)	40	60	100	120	120

Tableau C.16. soprano « a »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2900	3900	4950
amp (dB)	0	-6	-32	-20	-50
larg. bande (Hz)	80	90	120	130	140

Tableau C.17. soprano « e »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	2000	2800	3600	4950

Valeurs	f1	f2	f3	f4	f5
amp (dB)	0	-20	-15	-40	-56
larg. bande (Hz)	60	100	120	150	200

Tableau C.18. soprano « i »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	2140	2950	3900	4950
amp (dB)	0	-12	-26	-26	-44
larg. bande (Hz)	60	90	100	120	120

Tableau C.19. soprano « o »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3800	4950
amp (dB)	0	-11	-22	-22	-50
larg. bande (Hz)	40	80	100	120	120

Tableau C.20. soprano « u »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2700	3800	4950
amp (dB)	0	-16	-35	-40	-60
larg. bande (Hz)	50	60	170	180	200

Tableau C.21. ténor « a »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	650	1080	2650	2900	3250
amp (dB)	0	-6	-7	-8	-22
larg. bande (Hz)	80	90	120	130	140

Tableau C.22. ténor « e »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1700	2600	3200	3580
amp (dB)	0	-14	-12	-14	-20
larg. bande (Hz)	70	80	100	120	120

Tableau C.23. ténor « i »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	290	1870	2800	3250	3540
amp (dB)	0	-15	-18	-20	-30
larg. bande (Hz)	40	90	100	120	120

Tableau C.24. ténor « o »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	800	2600	2800	3000
amp (dB)	0	-10	-12	-12	-26
larg. bande (Hz)	70	80	100	130	135

Tableau C.25. ténor « u »

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2700	2900	3300
amp (dB)	0	-20	-17	-14	-26
larg. bande (Hz)	40	60	100	120	120

Annexe D. Rapports de Fréquence Modale

Contribution de Scott Lindroth

John Bower, un étudiant de Scott Lindroth, a dressé cette liste de fréquences modales pour différents objets et matériaux. Certains modes fonctionnent mieux que d'autres, et la plupart ne donnent des résultats plausibles que dans un intervalle de fréquences particulier. Caveat emptor.

En général, les objets en bois ne sonneront pas "bois" à moins qu'un composant aléatoire ne soit présent dans le son (essayez les guides d'onde en bandes). Néanmoins, certains des objets en bois font aussi de merveilleux instruments métalliques.

Ces rapports peuvent être utiles avec des opcodes comme *mode* ou *streson*.

Tableau D.1. Rapports de Fréquence Modale

Instrument	Rapports de Fréquence Modale
Dahina (tabla)	[1, 2.89, 4.95, 6.99, 8.01, 9.02]
Bayan (tabla)	[1, 2.0, 3.01, 4.01, 4.69, 5.63]
Plaque en bois de Cèdre Rouge	[1, 1.47, 2.09, 2.56]
Plaque en bois de Séquoia	[1, 1.47, 2.11, 2.57]
Plaque en bois de Sapin de Douglas	[1, 1.42, 2.11, 2.47]
Barre uniforme en bois	[1, 2.572, 4.644, 6.984, 9.723, 12]
Barre uniforme en aluminum	[1, 2.756, 5.423, 8.988, 13.448, 18.680]
Xylophone	[1, 3.932, 9.538, 16.688, 24.566, 31.147]
Vibraphone 1	[1, 3.984, 10.668, 17.979, 23.679, 33.642]
Vibraphone 2	[1, 3.997, 9.469, 15.566, 20.863, 29.440]
Plaques de Chladni	([62, 107, 360, 460, 863] Hz +-2Hz) [1, 1.72581, 5.80645, 7.41935, 13.91935] rapports
Bol tibétain (180mm)	([221, 614, 1145, 1804, 2577, 3456, 4419] Hz) 934g, 180mm [1, 2.77828, 5.18099, 8.16289, 11.66063, 15.63801, 19.99 rapports
Bol tibétain (152 mm)	([314, 836, 1519, 2360, 3341, 4462, 5696] Hz) 563g, 152mm [1, 2.66242, 4.83757, 7.51592, 10.64012, 14.21019, 18.14027] rapports
Bol tibétain (140 mm)	([528, 1460, 2704, 4122, 5694] Hz) 557g, 140mm [1, 2.76515, 5.12121, 7.80681, 10.78409] rapports

Instrument	**Rapports de Fréquence Modale**
Ver de vin	[1, 2.32, 4.25, 6.63, 9.38]
Petite cloche à main	([1312.0, 1314.5, 2353.3, 2362.9, 3306.5, 3309.4, 3923.8, 3928.2, 4966.6, 4993.7, 5994.4, 6003.0, 6598.9, 6619.7, 7971.7, 7753.2, 8413.1, 8453.3, 9292.4, 9305.2, 9602.3, 9912.4] Hz) [1, 1.0019054878049, 1.7936737804878, 1.8009908536585, 2.5201981707317, 2.5224085365854, 2.9907012195122, 2.9940548780488, 3.7855182926829, 3.8061737804878, 4.5689024390244, 4.5754573170732, 5.0296493902439, 5.0455030487805, 6.0759908536585, 5.9094512195122, 6.4124237804878, 6.4430640243902, 7.0826219512195, 7.0923780487805, 7.3188262195122, 7.5551829268293] rapports
Sphère en spinelle de diamètre 3.6675mm	([977.25, 1003.16, 1390.13, 1414.93, 1432.84, 1465.34, 1748.48, 1834.20, 1919.90, 1933.64, 1987.20, 2096.48, 2107.10, 2202.08, 2238.40, 2280.10, 0 /*2290.53 calculated*/, 2400.88, 2435.85, 2507.80, 2546.30, 2608.55, 2652.35, 2691.70, 2708.00] Hz) [1, 1.026513174725, 1.4224916858532, 1.4478690202098, 1.4661959580455, 1.499452545408, 1.7891839345101, 1.8768994627782, 1.9645945254541, 1.9786543873113, 2.0334612432847, 2.1452852391916, 2.1561524686621, 2.2533435661294, 2.2905090816065, 2.3331798413917, 0, 2.4567715528268, 2.4925556408289, 2.5661806088514, 2.6055768738808, 2.6692760296751, 2.7140956766436, 2.7543617293425, 2.7710411870043] rapports
Couvercle de pot	[1, 3.2, 6.23, 6.27, 9.92, 14.15] rapports

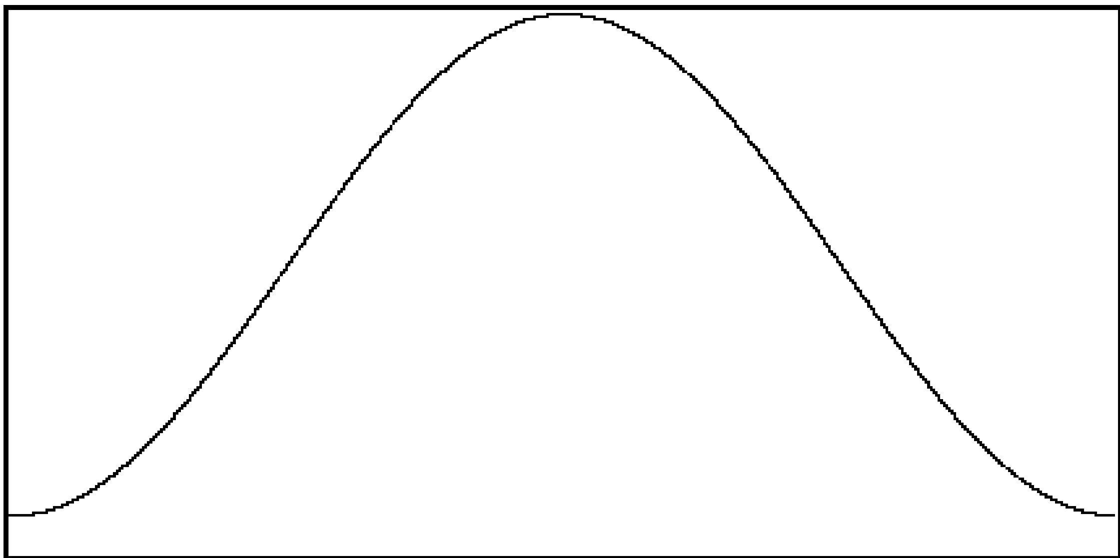
Annexe E. Fonctions Fenêtres

Les fonctions fenêtres sont utilisées pour l'analyse, et comme enveloppes de forme d'onde, particulièrement dans la synthèse granulaire. Les fonctions fenêtre sont intégrées à certains opcodes, mais d'autres opcodes nécessitent une table de fonction pour générer la fenêtre. *GEN20* est utilisé à cet effet. Le diagramme de chaque fenêtre ci-dessous est accompagné de l'instruction f utilisée pour la générer.

Hamming.

Exemple E.1. Instruction pour la fonction fenêtre de Hamming

```
f81   0   8192   20   1   1
```

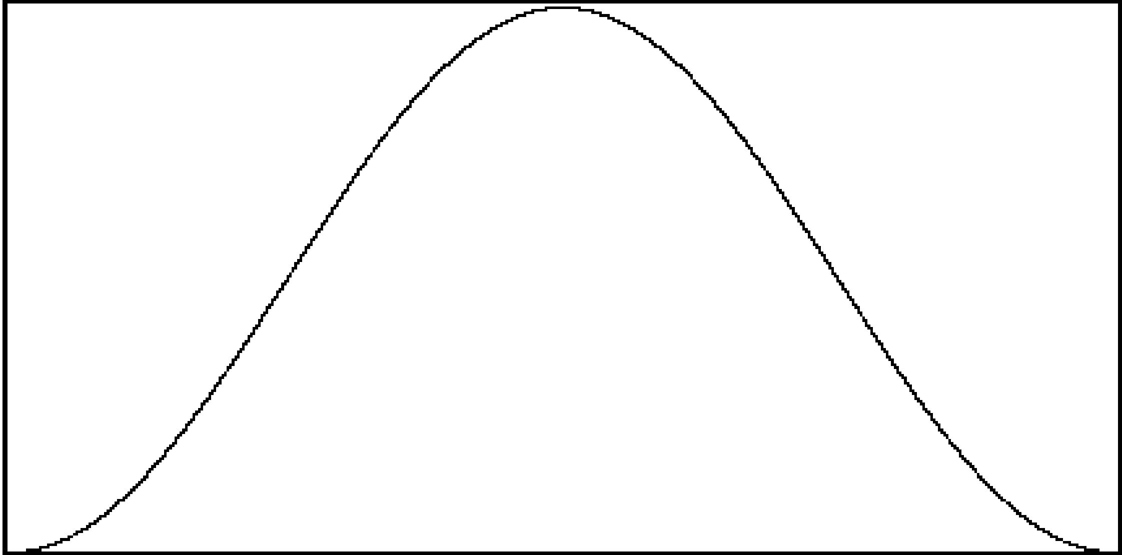


Fonction Fenêtre de Hamming.

Hanning.

Exemple E.2. Instruction pour la fonction fenêtre de Hanning

```
f82   0   8192   20   2   1
```

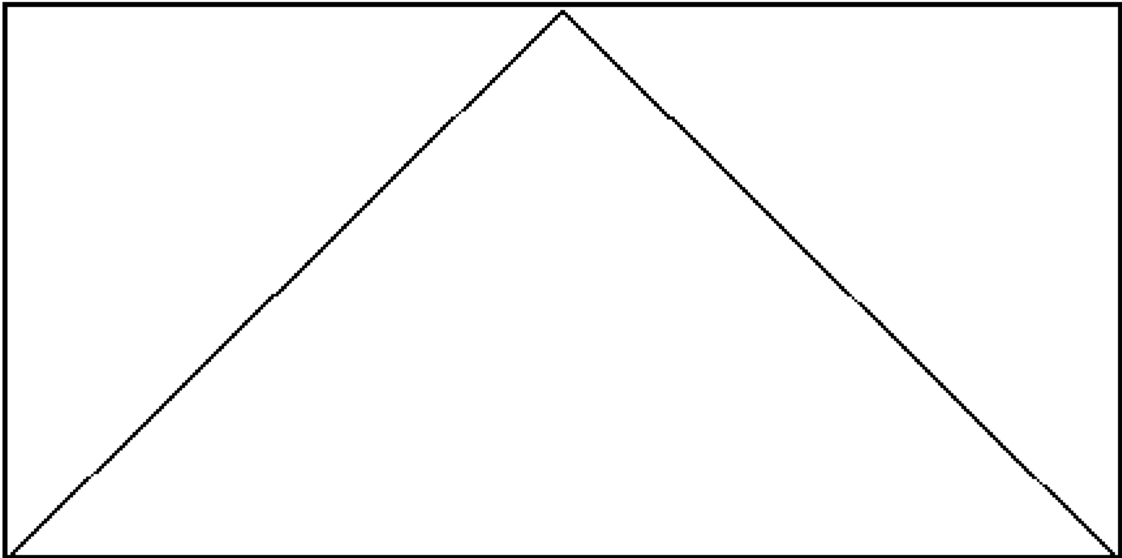


Fonction Fenêtre de Hanning

Bartlett.

Exemple E.3. Instruction pour la fonction fenêtre de Bartlett

```
f83   0   8192   20   3   1
```

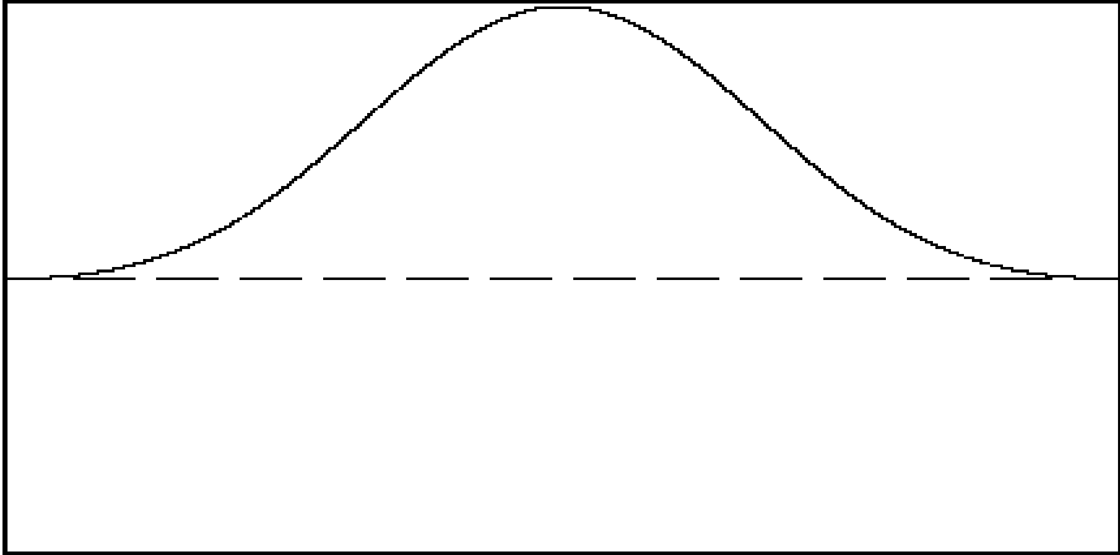


Fonction Fenêtre de Bartlett

Blackman.

Exemple E.4. Instruction pour la fonction fenêtre de Blackman

```
f84   0   8192   20   4   1
```

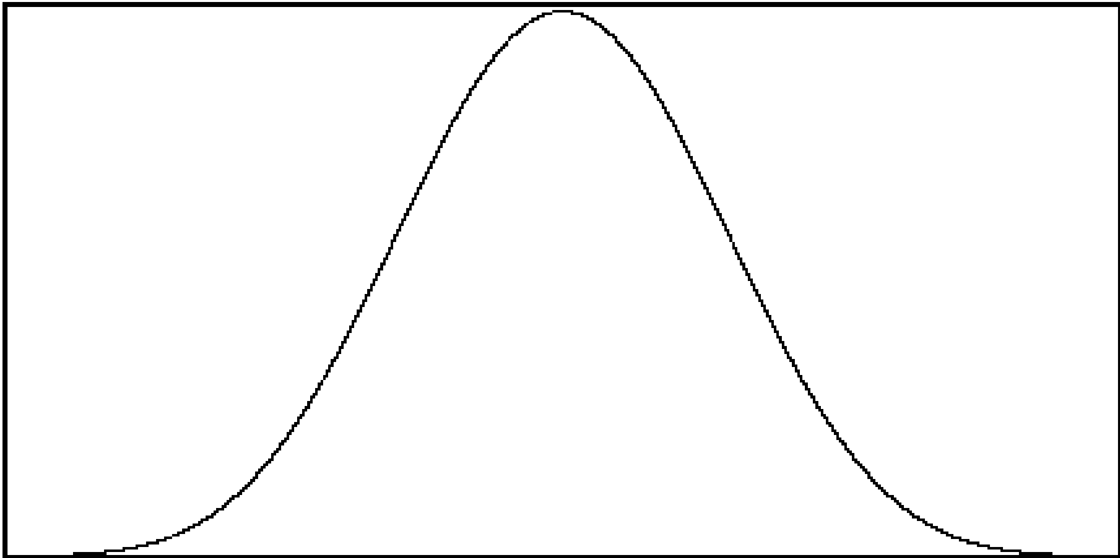


Fonction Fenêtre de Blackman

Blackman-Harris.

Exemple E.5. Instruction pour la fonction fenêtre de Blackman-Harris

```
f85   0   8192   20   5   1
```

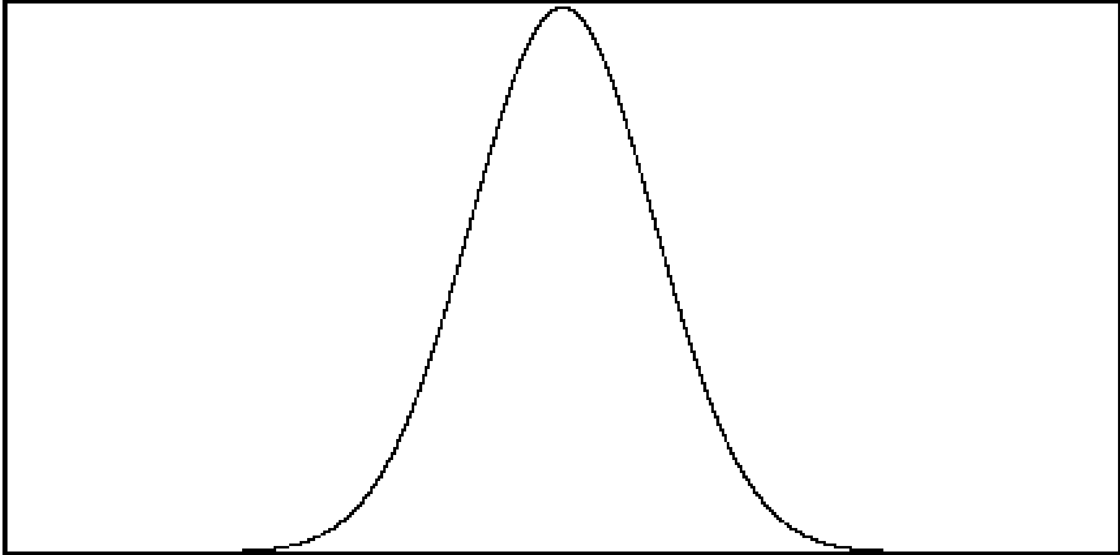


Fonction Fenêtre de Blackman-Harris

Gaussienne.

Exemple E.6. Instruction pour la fonction fenêtre Gaussienne

```
f86   0   8192   20   6   1
```



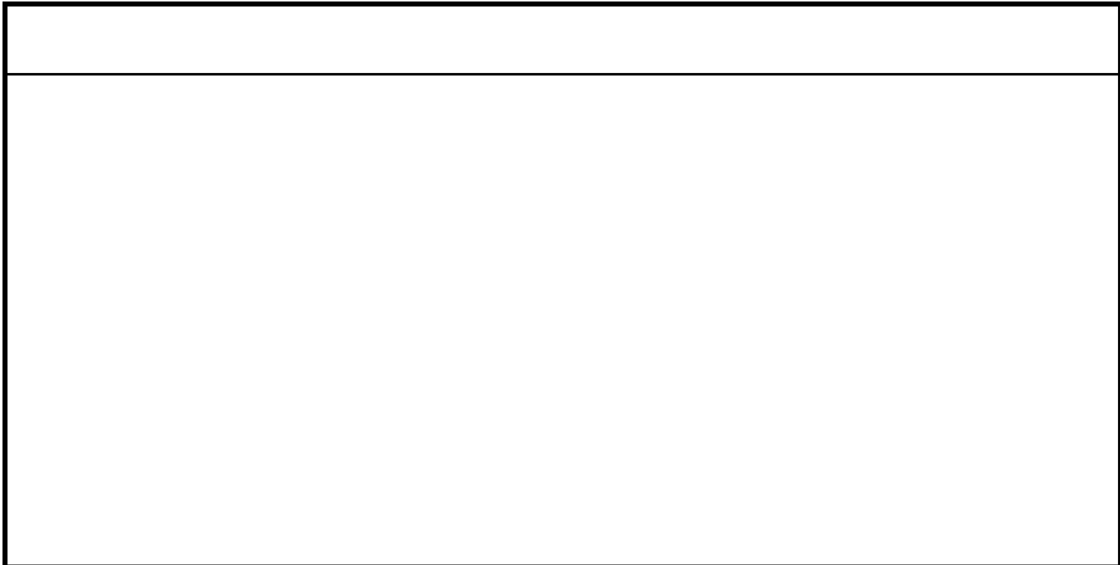
Fonction Fenêtre Gaussienne

Rectangle.

Exemple E.7. Instruction pour la fonction fenêtre Rectangle

```
f88   0   8192   -20   8   .1
```

Note : l'échelle verticale est exagérée dans ce diagramme.

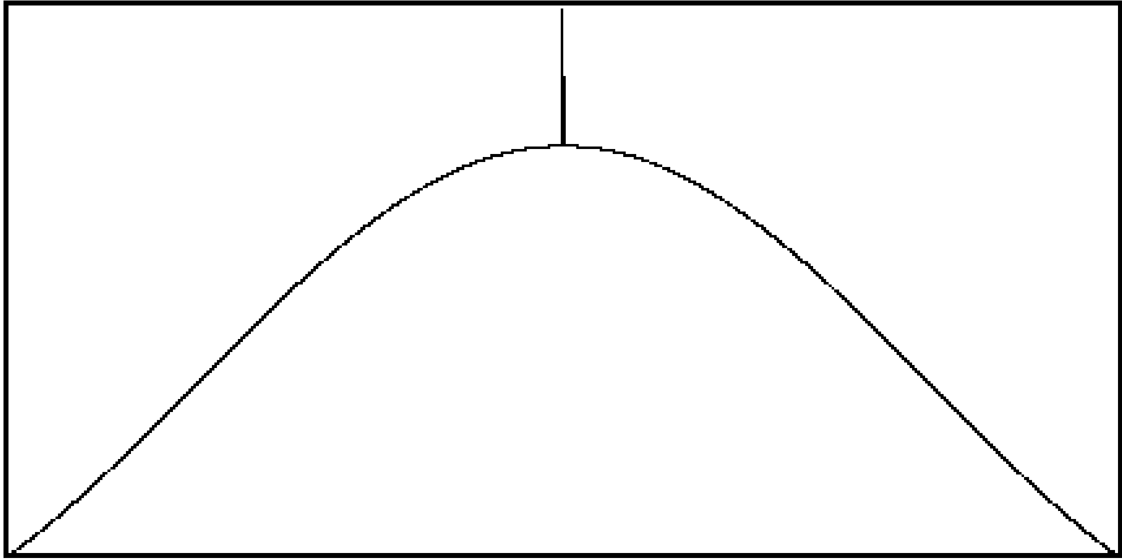


Fonction Fenêtre Rectangle

Sync.

Exemple E.8. Instruction pour la fonction fenêtre Sync

```
f89   0   4096   -20   9   .75
```



Fonction Fenêtre Sync

Annexe F. Format de Fichier SoundFont2

A partir de la version 4.07 de Csound, *Csound supporte le format de fichier de sons échantillonnés SoundFont2*. SoundFont2 (ou SF2) est un standard répandu qui permet l'encodage de banques de sons basés sur des tables d'onde dans un fichier binaire. Afin de comprendre l'usage de ces opcodes, l'utilisateur doit avoir une certaine connaissance du format SF2, c'est pourquoi une brève description de ce format suit.

Le format SF2 comprend des objets générateurs et modulateurs. Tous les opcodes actuels de Csound concernant SF2 ne supportent que la fonction générateur.

Il y a plusieurs niveaux de générateurs ayant une structure hiérarchique. Le type de générateur le plus élémentaire est le « sample » (son échantillonné). Les samples peuvent être bouclés ou non, et sont associés avec un numéro de note MIDI, appelé la touche de base. Quand un sample est associé à un intervalle de numéros de notes MIDI, un intervalle de vélocités, une transposition (accord grossier et fin), un accord d'échelle, un facteur de pondération de niveau, le sample et ses associations constituent un « split » (division). Un ensemble de splits, avec un nom, constituent un « instrument ». Quand un instrument est associé avec un intervalle de touches, un intervalle de vélocités, un facteur de pondération de niveau, et une transposition, l'instrument et ses associations constituent un « layer » (couche). Un ensemble de layers, avec un nom, constituent un « preset ». Les presets sont normalement les structures de génération sonore finales prêtes pour l'utilisateur. Ils génèrent le son selon les réglages de leurs composants des niveaux inférieurs.

Les données des sons échantillonnés et les données de structure sont incorporées dans le même fichier binaire SF2. Un fichier SF2 unique peut contenir au maximum 128 banques de 128 programmes de preset, soit un total de 16384 presets dans un fichier SF2. Le nombre maximum de layers, instruments, splits et samples n'est probablement limité que par la mémoire de l'ordinateur.

Annexe G. Csound Double (64 bit) ou Float (32 bit)

On peut construire Csound pour utiliser des nombres en virgule flottante DOUBLES sur 64 bit pour le traitement en interne au lieu des habituels nombres en virgule flottante FLOATS sur 32 bit. Cette plus grande précision pour le traitement interne produit un son bien plus "propre" mais au prix d'un temps de traitement plus long. Parce que csound met bien plus de temps pour ses calculs s'il a été compilé pour des doubles, il est utilisé typiquement en fin de travail pour produire la version finale d'une oeuvre. Si vous utilisez csound pour une sortie en temps réel, il vaut mieux utiliser une version 32 bit (float), qui fournit une sortie plus rapidement. Pour un rendu différé, vous pouvez utiliser l'une ou l'autre version, mais pour le master final, la version 64 bit produira une sortie de meilleure qualité.

La résolution choisie doit être la même que celle du type de variable des échantillons audio. Dans Csound "float" il s'agit de nombres en virgule flottante simple précision sur 32 bit. La mantisse occupe 24 bit. Dans Csound "double, il s'agit de nombres en virgule flottante double précision sur 64 bit.La mantisse occupe 52 bit. Chaque chiffre décimal nécessite 3 ou 4 bit. Ainsi, il y a une précision de 7 chiffres en "float" et de 16 chiffres en "double".

Pour chaque multiplication ou division, selon que les opérandes sont entiers, rationnels, décimaux périodiques ou irrationnels, le résultat peut présenter une erreur d'arrondi. S'il y a une erreur d'arrondi, il y a au plus une perte de précision d'un bit par opération (en plus des erreurs d'arrondi dues à la représentation binaire des nombres rationnels ou réels).

Pour les échantillons de type float, si le signal ne déborde pas la mantisse, le rapport signal-bruit vaut 6,02 fois 24, soit 144 dB. Au pire, chaque opération ajoutera 6,02 dB de bruit dû à l'erreur d'arrondi. Notre audition réagit à un ambitus dynamique effectif de 120 à 130 dB, mais nous apprécions que notre musique soit compressée dans un intervalle dynamique d'AU PLUS 60 dB (et habituellement beaucoup moins, disons 20 dB). Ceci nous donne (144 - 60) / 6,02 = environ 10 opérations défavorables avant que nous puissions entendre une dégradation. En pratique, nous pouvons enchaîner plusieurs fois ce nombre d'opérations avant d'entendre une dégradation ou du bruit.

Pour les échantillons de type double, si le signal ne déborde pas la mantisse, le rapport signal-bruit vaut 6,02 fois 52, soit 313 dB. Au pire, chaque opération ajoutera 6,02 dB de bruit dû à l'erreur d'arrondi. Ceci nous donne (313 - 60) / 6,02 = environ 42, en pratique plusieurs fois ce nombre d'opérations avant qu'il n'y ait une dégradation audible ou du bruit.

Mais si l'on relève le nombre d'opérations arithmétiques dans des instruments typiques de Csound ou d'autres synthétiseurs logiciels, les instruments très complexes entrent définitivement dans la zone des dégradations audibles sur de bons haut-parleurs avec float, et il n'est donc pas surprenant que dans certains cas, un test ABX à l'aveugle confirme des différences audibles *occasionnelles* entre de la musique synthétisée avec Csound "float" et la même musique synthétisée avec Csound "double". De même, il est courant de constater des différences audibles entre les implémentations numérique et analogique des mêmes algorithmes de synthèse.

Avec Csound double "l'espace de traitement numérique du signal" est nettement plus large, ce qui le rend plus adéquat pour toute musique dont l'écoute est critique. La version float ne devrait être utilisée que si son avantage en vitesse (environ 15 %) est critique pour l'exécution en temps-réel.

Notes sur l'utilisation de Csound construit pour la double précision

1. Les fichiers *hetro*, d'analyse PVOC-EX et *pvanal* générés pour Csound 32 bit (float) fonctionneront

avec Csound 64 bit (double précision).

2. Les fichiers *lpanal* et *cvanal* générés pour Csound ne fonctionneront pas avec Csound64.

Glossaire

G

Point de Garde

Un point de garde est la dernière position d'une table de fonction. Si la longueur est, disons 1024, la table aura 1024+1 (1025) points : le point supplémentaire est le point de garde.

Dans tous les cas, pour une table de 1024 points, le premier point aura l'index 0 et le dernier l'index 1023 (l'index 1024 n'est pas réellement utilisé).

Il y a un point de garde car certains opcodes lisent les valeurs de la table par interpolation ; dans ce cas, si l'index de lecture est par exemple 1023,5, nous aurons besoin de la position 1024 pour l'interpolation.

Il y a deux manières de remplir ce point (écrire sa valeur) :

1. La manière par défaut : en copiant la valeur du 1er point de la table
2. Le point de garde étendu : en prolongeant le contour de la table (en continuant le calcul pour un point supplémentaire)

En général le premier mode est utilisé pour les applications cycliques, comme un oscillateur (qui lit la table en boucle continue). Le second usage est pour les lectures à passage unique, comme les enveloppes, où il faut interpoler le dernier point correctement en suivant le contour de la table (on ne boucle pas sur le début de la table).