[1] [] 1 [Generated on Sun Sep 19 09:41:16 2004 for csound by Doxygen] []Generated on Sun Sep 19 09:41:16 2004 for csound by Doxygen
.

Csound and CsoundVST API Reference Manual

By
Michael Gogins
gogins@pipeline.com

16th July 2005

Contents

[2]

Part I.

Licenses

1. Csound and CsoundVST

2. Manual

Part II.

API Reference

3. The Csound Application Programming Interfaces

The Csound Application Programming Interface (API) reference is contained in the chapters following this one. The Csound API actually consists of several APIs:

- *The Csound C API.* Include `csound.h` (page 590) and link with `libcsound.a`.
- *The CsoundVST C API.* Include `csoundvst_api.h` (page 562) and link with `libCsoundVST.a`.
- *The CsoundVST C++ API.* Include `CsoundVST.hpp` (page 561) and link with `libCsoundVST.a`. The `CsoundVST` class (6.31) contains an instance of the `CppSound` class (6.28), which provides a C++ wrapper for the C API as well as the CsoundFile class (6.30) for loading, saving, and editing Csound orchestra and score files, and a basic graphical user interface for editing Csound files and running Csound.

  The C++ API also provides a class hierarchy for doing algorithmic composition using Michael Gogins' concept of music graphs.
- *The CsoundVST Python API.* Import the `CsoundVST` Python extension module. Because the Python API provides a complete Python wrapper for the entire C++ API, the C++ API reference also serves as a reference to the Python API.

3.1. An Example Using the Csound API

The Csound command–line program is itself built using the Csound API. Its code reads in full as follows:

```
#include "csound.h"

int main(int argc, char **argv)
{
    // Create Csound.
    void *csound = csoundCreate(0);
    // One complete performance cycle.
    int result = csoundCompile(csound, argc, argv);
    if(!result)
    {
        while(csoundPerformKsmps(csound) == 0){}
        csoundCleanup(csound);
    }
    // Destroy Csound.
    csoundDestroy(csound);
    return result;
}
```

3.2. An Example Using the CsoundVST C++ API

CsoundVST extends the Csound API with C++. There is a C++ class for the Csound API proper, another C++ class for manipulating Csound files in code, and additional classes for algorithmic composition based on music space. All these C++ classes also have a Python interface in the `CsoundVST` Python extension module.

You can build CsoundVST into your own software using the `_CsoundVST` shared library and `CsoundVST.hpp` header file. For example, the CsoundVST stand-alone graphical user interface program is made this way:

```
int main(int argc, char **argv)
{
    CsoundVST *csoundVST = CreateCsoundVST();
    AEffEditor *editor = csoundVST->getEditor();
    editor->open(0);
    if(argc == 2) {
      csoundVST->openFile(argv[1]);
    }
    return csoundVST->run();
}
```

There is also a high-level C API for CsoundVST, declared in `frontends/CsoundVST/csoundvst_api.h`. Any program able to interface with C calling convention functions can use this API. For example, a ***Mathematica*** 5.0 notebook can use the .NET/Link package's `DefineDLLFunction` to access the CsoundVST API to create an instance of CsoundVST, load an orchestra, generate a score using the power of ***Mathematica***, and render that score.

4. Csound and CsoundVST Directory Documentation

4.1. frontends/CsoundVST/ Directory Reference

Files

- file Cell.hpp
- file Composition.hpp
- file Conversions.hpp
- file Counterpoint.hpp
- file CounterpointNode.hpp
- file CppSound.hpp
- file CsoundFile.hpp
- file CsoundVST.hpp
- file csoundvst_api.h
- file CsoundVstFltk.hpp
- file Event.hpp
- file Exception.hpp
- file Hocket.hpp
- file ImageToScore.hpp
- file Lindenmayer.hpp
- file MCRM.hpp
- file Midifile.hpp
- file MusicModel.hpp
- file Node.hpp
- file Random.hpp
- file Rescale.hpp
- file Score.hpp
- file ScoreNode.hpp
- file Sequence.hpp
- file Shell.hpp
- file Silence.hpp
- file StrangeAttractor.hpp
- file System.hpp

4.2. frontends/flcsound/ Directory Reference

Files

- file canvas.hpp
- file curve.hpp
- file plots.hpp
- file synthesizer.hpp

4.3. Opcodes/fluidOpcodes/ Directory Reference

Files

- file fluidOpcodes.hpp

4.4. frontends/ Directory Reference

Directories

- directory CsoundVST
- directory flcsound

4.5. H/ Directory Reference

Files

- file csdl.h
- file csound.h
- file csoundCore.h
- file OpcodeBase.hpp

4.6. Opcodes/stk/include/ Directory Reference

Files

- file Envelope.h
- file Filter.h

4.7. Opcodes/Loris/ Directory Reference

Directories

- directory src

4.8. Opcodes/ Directory Reference

Directories

- directory fluidOpcodes
- directory Loris
- directory stk

Files

- file filter.h

4.9. Opcodes/Loris/src/ Directory Reference

Files

- file AiffData.h
- file AiffFile.h
- file Analyzer.h
- file AssociateBandwidth.h
- file Breakpoint.h
- file BreakpointEnvelope.h
- file BreakpointUtils.h
- file Channelizer.h
- file Dilator.h
- file Distiller.h
- file Endian.h
- file Envelope.h
- file Exception.h
- file Filter.h
- file FourierTransform.h
- file FrequencyReference.h
- file ImportLemur.h
- file KaiserWindow.h
- file Marker.h
- file Morpher.h
- file NoiseGenerator.h
- file Notifier.h
- file Oscillator.h
- file Partial.h
- file PartialBuilder.h
- file PartialList.h
- file PartialPtrs.h
- file PartialUtils.h
- file ReassignedSpectrum.h
- file Resampler.h
- file SdifFile.h
- file Sieve.h
- file SpcFile.h
- file SpectralPeaks.h
- file SpectralPeakSelector.h
- file Synthesizer.h

4.10. Opcodes/stk/ Directory Reference

Directories

- directory include

5. Csound and CsoundVST Namespace Documentation

5.1. boost::numeric Namespace Reference

5.1.1. Detailed Description

C S O U N D V S T

A VST plugin version of Csound, with Python scripting.

L I C E N S E

This software is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

5.2. csound Namespace Reference

5.2.1. Detailed Description

C S O U N D V S T

A VST plugin version of Csound, with Python scripting.

L I C E N S E

Classes

- class Cell
- class Composition
- class Conversions
- class CounterpointNode
- class Event
- class Exception
- class Hocket
- class ImageToScore
- class Lindenmayer
- class MCRM
- class Chunk
- class MidiHeader
- class MidiEvent
- class MidiTrack
- class TempoMap
- class MidiFile
- class MusicModel
- class Node
- class Random
- class Rescale
- class Score
- class ScoreNode
- class Sequence
- class Shell
- class StrangeAttractor
- class Logger
- class System
- class ThreadLock

Typedefs

- typedef unsigned char csound_u_char
- typedef Node * NodePtr
- typedef void(* MessageCallbackType)(void *userdata, int attribute, const char *format, va_list valist)

Functions

- bool operator< (const Event &a, const Event &b)
- bool operator< (const MidiEvent &a, MidiEvent &b)

5.2.2. Typedef Documentation

5.2.2.1. typedef unsigned char csound::csound_u_char

Definition at line 50 of file Midifile.hpp.

5.2.2.2. typedef void(* csound::MessageCallbackType)(void *userdata, int attribute, const char *format, va_list valist)

Definition at line 42 of file System.hpp.

5.2.2.3. typedef Node* csound::NodePtr

Definition at line 84 of file Node.hpp.

5.2.3. Function Documentation

5.2.3.1. bool operator< (const MidiEvent & *a*, MidiEvent & *b*)

5.2.3.2. bool operator< (const Event & *a*, const Event & *b*)

5.3. Loris Namespace Reference

Classes

- struct CkHeader
- struct ContainerCk
- struct CommonCk
- struct SoundDataCk
- struct MarkerCk
- struct InstrumentCk
- struct SosEnvelopesCk
- class AiffFile
- class Analyzer
- class AssociateBandwidth
- class Breakpoint
- class BreakpointEnvelope
- class Channelizer
- class Dilator
- class Distiller
- class BigEndian
- class Envelope
- class Exception
- class AssertionFailure
- class IndexOutOfBounds
- class InvalidObject
- class InvalidIterator
- class InvalidArgument

  Class of exceptions thrown when a function argument is found to be invalid.

- class RuntimeError
- class FileIOException

  Class of exceptions thrown when file input or output fails.

- class Filter
- class FourierTransform
- class FrequencyReference
- class ImportLemur
- class ImportException
- class KaiserWindow
- class Marker
- class Morpher
- class NoiseGenerator
- class Oscillator
- class Partial
- class Partial_Iterator
- class Partial_ConstIterator
- class InvalidPartial
- class PartialBuilder
- class ReassignedSpectrum
- class Resampler
- class SdifFile
- class Sieve
- class SpcFile
- class SpectralPeakSelector
- class Synthesizer

Namespaces

- namespace BreakpointUtils
- namespace PartialUtils

Typedefs

- typedef unsigned long ID
- typedef char Byte
- typedef std::list< Loris::Partial > PartialList
- typedef std::list< Loris::Partial >::iterator PartialListIterator
- typedef std::list< Loris::Partial >::const_iterator PartialListConstIterator
- typedef std::vector< Partial * > PartialPtrs
- typedef std::vector< Partial * >::iterator PartialPtrsIterator
- typedef std::vector< Partial * >::const_iterator PartialPtrsConstIterator
- typedef std::vector< const Partial * > ConstPartialPtrs
- typedef std::vector< const Partial * >::iterator ConstPartialPtrsIterator
- typedef std::vector< const Partial * >::const_iterator ConstPartialPtrsConstIterator
- typedef std::vector< std::pair< double, Breakpoint > > Peaks

Enumerations

- enum {

  ContainerId = 0x464f524d, AiffType = 0x41494646, CommonId = 0x434f4d4d, ApplicationSpecificId = 0x4150504c,

  SosEnvelopesId = 0x534f5365, SoundDataId = 0x53534e44, InstrumentId = 0x494e5354, MarkerId = 0x4d41524b }

Functions

- std::istream & readChunkHeader (std::istream &s, CkHeader &h)
- std::istream & readApplicationSpecifcData (std::istream &s, SosEnvelopesCk &ck, unsigned long chunkSize)
- std::istream & readCommonData (std::istream &s, CommonCk &ck, unsigned long chunkSize)
- std::istream & readContainer (std::istream &s, ContainerCk &ck, unsigned long chunkSize)
- std::istream & readInstrumentData (std::istream &s, InstrumentCk &ck, unsigned long chunkSize)
- std::istream & readMarkerData (std::istream &s, MarkerCk &ck, unsigned long chunkSize)
- std::istream & readSampleData (std::istream &s, SoundDataCk &ck, unsigned long chunkSize)
- void configureCommonCk (CommonCk &ck, unsigned long nFrames, unsigned int nChans, unsigned int bps, double srate)
- void configureContainer (ContainerCk &ck, unsigned long dataSize)
- void configureInstrumentCk (InstrumentCk &ck, double midiNoteNum)
- void configureMarkerCk (MarkerCk &ck, const std::vector< Marker > &markers, double srate)
- void configureSoundDataCk (SoundDataCk &ck, const std::vector< double > &samples, unsigned int bps)
- std::ostream & writeCommonData (std::ostream &s, const CommonCk &ck)
- std::ostream & writeContainer (std::ostream &s, const ContainerCk &ck)

- std::ostream & writeInstrumentData (std::ostream &s, const InstrumentCk &ck)
- std::ostream & writeMarkerData (std::ostream &s, const MarkerCk &ck)
- std::ostream & writeSampleData (std::ostream &s, const SoundDataCk &ck)
- void convertBytesToSamples (const std::vector< Byte > &bytes, std::vector< double > &samples, unsigned int bps)
- void convertSamplesToBytes (const std::vector< double > &samples, std::vector< Byte > &bytes, unsigned int bps)
- template<typename Iter> void fillPartialPtrs (Iter begin, Iter end, PartialPtrs &fillme)
- template<typename Iter> void fillPartialPtrs (Iter begin, Iter end, ConstPartialPtrs &fillme)

5.3.1. Typedef Documentation

5.3.1.1. typedef char Loris::Byte

Definition at line 93 of file AiffData.h.

5.3.1.2. typedef std::vector< const Partial * > Loris::ConstPartialPtrs

Definition at line 72 of file PartialPtrs.h.

5.3.1.3. typedef std::vector< const Partial * >::const_iterator Loris::ConstPartialPtrsConstIterator

Definition at line 74 of file PartialPtrs.h.

5.3.1.4. typedef std::vector< const Partial * >::iterator Loris::ConstPartialPtrsIterator

Definition at line 73 of file PartialPtrs.h.

5.3.1.5. typedef unsigned long Loris::ID

Definition at line 92 of file AiffData.h.

5.3.1.6. typedef std::list< Loris::Partial > Loris::PartialList

Definition at line 55 of file PartialList.h.

5.3.1.7. typedef std::list< Loris::Partial >::const_iterator Loris::PartialListConstIterator

Definition at line 57 of file PartialList.h.

5.3.1.8. typedef std::list< Loris::Partial >::iterator Loris::PartialListIterator

Definition at line 56 of file PartialList.h.

5.3.1.9. typedef std::vector< Partial * > Loris::PartialPtrs

Definition at line 68 of file PartialPtrs.h.

5.3.1.10. typedef std::vector< Partial * >::const_iterator Loris::PartialPtrsConstIterator

Definition at line 70 of file PartialPtrs.h.

5.3.1.11. typedef std::vector< Partial * >::iterator Loris::PartialPtrsIterator

Definition at line 69 of file PartialPtrs.h.

5.3.1.12. typedef std::vector< std::pair< double, Breakpoint > > Loris::Peaks

Definition at line 45 of file SpectralPeaks.h.

5.3.2. Enumeration Type Documentation

5.3.2.1. anonymous enum

Enumeration values:

ContainerId

AiffType

CommonId

ApplicationSpecificId

SosEnvelopesId

SoundDataId

InstrumentId

MarkerId

Definition at line 80 of file AiffData.h.

5.3.3. Function Documentation

5.3.3.1. void configureCommonCk (CommonCk & *ck*, unsigned long *nFrames*, unsigned int *nChans*, unsigned int *bps*, double *srate*)

5.3.3.2. void configureContainer (ContainerCk & *ck*, unsigned long *dataSize*)

5.3.3.3. void configureInstrumentCk (InstrumentCk & *ck*, double *midiNoteNum*)

5.3.3.4. void configureMarkerCk (MarkerCk & *ck*, const std::vector< Marker > & *markers*, double *srate*)

5.3.3.5. void Loris::configureSoundDataCk (SoundDataCk & *ck*, const std::vector< double > & *samples*, unsigned int *bps*)

5.3.3.6. void convertBytesToSamples (const std::vector< Byte > & *bytes*, std::vector< double > & *samples*, unsigned int *bps*)

5.3.3.7. void convertSamplesToBytes (const std::vector< double > & *samples*, std::vector< Byte > & *bytes*, unsigned int *bps*)

5.3.3.8. template<typename Iter> void fillPartialPtrs (Iter *begin*, Iter *end*, ConstPartialPtrs & *fillme*)

Definition at line 96 of file PartialPtrs.h.

5.3.3.9. template<typename Iter> void fillPartialPtrs (Iter *begin*, Iter *end*, PartialPtrs & *fillme*)

Definition at line 87 of file PartialPtrs.h.

Referenced by Loris::Sieve::sift().

5.3.3.10. std::istream& readApplicationSpecifcData (std::istream & *s*, SosEnvelopesCk & *ck*, unsigned long *chunkSize*)

5.3.3.11. std::istream& readChunkHeader (std::istream & *s*, CkHeader & *h*)

5.3.3.12. std::istream& readCommonData (std::istream & *s*, CommonCk & *ck*, unsigned long *chunkSize*)

5.3.3.13. std::istream& readContainer (std::istream & *s*, ContainerCk & *ck*, unsigned long *chunkSize*)

5.3.3.14. std::istream& readInstrumentData (std::istream & *s*, InstrumentCk & *ck*, unsigned long *chunkSize*)

5.3.3.15. std::istream& readMarkerData (std::istream & *s*, MarkerCk & *ck*, unsigned long *chunkSize*)

5.3.3.16. std::istream& readSampleData (std::istream & *s*, SoundDataCk & *ck*, unsigned long *chunkSize*)

5.3.3.17. std::ostream& writeCommonData (std::ostream & *s*, const CommonCk & *ck*)

5.3.3.18. std::ostream& writeContainer (std::ostream & *s*, const ContainerCk & *ck*)

5.3.3.19. std::ostream& writeInstrumentData (std::ostream & *s*, const InstrumentCk & *ck*)

5.3.3.20. std::ostream& writeMarkerData (std::ostream & *s*, const MarkerCk & *ck*)

5.3.3.21. std::ostream& writeSampleData (std::ostream & *s*, const SoundDataCk & *ck*)

5.4. Loris::BreakpointUtils Namespace Reference

Classes

- struct frequency_between
- struct less_frequency
- struct greater_amplitude

Functions

- void addNoiseEnergy (Breakpoint &bp, double enoise)
- Breakpoint makeNullBefore (const Breakpoint &bp, double fadeTime)
- Breakpoint makeNullAfter (const Breakpoint &bp, double fadeTime)

5.4.1. Function Documentation

5.4.1.1. void addNoiseEnergy (Breakpoint & *bp*, double *enoise*) [inline]

Definition at line 78 of file BreakpointUtils.h.

5.4.1.2. Breakpoint makeNullAfter (const Breakpoint & *bp*, double *fadeTime*)

5.4.1.3. Breakpoint makeNullBefore (const Breakpoint & *bp*, double *fadeTime*)

5.5. Loris::PartialUtils Namespace Reference

Classes

- class PartialMutator
- class AmplitudeScaler
- class BandwidthScaler
- class FrequencyScaler
- class NoiseRatioScaler
- class PitchShifter
- class Cropper
- class TimeShifter
- class isLabelEqual
- class isLabelGreater
- class isLabelLess
- class compareLabelLess
- class compareDurationLess
- class compareDurationGreater

Functions

- template<class Arg> void scaleAmplitude (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleAmplitude (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleBandwidth (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleBandwidth (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleFrequency (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleFrequency (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleNoiseRatio (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleNoiseRatio (Iter b, Iter e, const Arg &arg)
- template<class Arg> void shiftPitch (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void shiftPitch (Iter b, Iter e, const Arg &arg)
- void crop (Partial &p, double t1, double t2)
- template<class Iter> void crop (Iter b, Iter e, double t1, double t2)
- void shiftTime (Partial &p, double arg)
- template<class Iter> void shiftTime (Iter b, Iter e, double arg)
- template<typename Iterator> std::pair< double, double > timeSpan (Iterator begin, Iterator end)

5.5.1. Function Documentation

5.5.1.1. template<class Iter> void crop (Iter *b*, Iter *e*, double *t1*, double *t2*)

Definition at line 346 of file PartialUtils.h.

5.5.1.2. void crop (Partial & *p*, double *t1*, double *t2*) [inline]

Definition at line 339 of file PartialUtils.h.

5.5.1.3. template<class Iter, class Arg> void scaleAmplitude (Iter *b*, Iter *e*, const Arg & *arg*)

Scale the amplitude of a sequence of Partials according to an envelope representing a amplitude scale value or envelope.

Parameters:

b is the beginning of a sequence of Partials to mutate.

e is the end of a sequence of Partials to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 148 of file PartialUtils.h.

5.5.1.4. template<class Arg> void scaleAmplitude (Partial & *p*, const Arg & *arg*)

Scale the amplitude of the specified Partial according to an envelope representing a amplitude scale value or envelope.

Parameters:

p is a Partial to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 130 of file PartialUtils.h.

5.5.1.5. template<class Iter, class Arg> void scaleBandwidth (Iter *b*, Iter *e*, const Arg & *arg*)

Scale the bandwidth of a sequence of Partials according to an envelope representing a amplitude scale value or envelope.

Parameters:

b is the beginning of a sequence of Partials to mutate.

e is the end of a sequence of Partials to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 208 of file PartialUtils.h.

5.5.1.6. template<class Arg> void scaleBandwidth (Partial & *p*, const Arg & *arg*)

Scale the bandwidth of the specified Partial according to an envelope representing a amplitude scale value or envelope.

Parameters:

p is a Partial to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 190 of file PartialUtils.h.

5.5.1.7. template<class Iter, class Arg> void scaleFrequency (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 240 of file PartialUtils.h.

5.5.1.8. template<class Arg> void scaleFrequency (Partial & *p*, const Arg & *arg*)

Definition at line 233 of file PartialUtils.h.

5.5.1.9. template<class Iter, class Arg> void scaleNoiseRatio (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 273 of file PartialUtils.h.

5.5.1.10. template<class Arg> void scaleNoiseRatio (Partial & *p*, const Arg & *arg*)

Definition at line 266 of file PartialUtils.h.

5.5.1.11. template<class Iter, class Arg> void shiftPitch (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 306 of file PartialUtils.h.

5.5.1.12. template<class Arg> void shiftPitch (Partial & *p*, const Arg & *arg*)

Definition at line 299 of file PartialUtils.h.

5.5.1.13. template<class Iter> void shiftTime (Iter *b*, Iter *e*, double *arg*)

Definition at line 380 of file PartialUtils.h.

5.5.1.14. void shiftTime (Partial & *p*, double *arg*) [inline]

Definition at line 373 of file PartialUtils.h.

5.5.1.15. template<typename Iterator> std::pair< double, double > timeSpan (Iterator *begin*, Iterator *end*)

Definition at line 396 of file PartialUtils.h.

6. Csound and CsoundVST Class Documentation

6.1. AIFFDAT Struct Reference

`#include <csoundCore.h>`

Public Attributes

- MYFLT natcps
- MYFLT gainfac
- short loopmode1
- short loopmode2
- long begin1
- long end1
- long begin2
- long end2
- MYFLT fmaxamps [AIFF_MAXCHAN+1]

6.1.1. Member Data Documentation

6.1.1.1. long AIFFDAT::begin1

Definition at line 366 of file csoundCore.h.

6.1.1.2. long AIFFDAT::begin2

Definition at line 367 of file csoundCore.h.

6.1.1.3. long AIFFDAT::end1

Definition at line 366 of file csoundCore.h.

6.1.1.4. long AIFFDAT::end2

Definition at line 367 of file csoundCore.h.

6.1.1.5. MYFLT AIFFDAT::fmaxamps[AIFF_MAXCHAN+1]

Definition at line 368 of file csoundCore.h.

6.1.1.6. MYFLT AIFFDAT::gainfac

Definition at line 363 of file csoundCore.h.

6.1.1.7. short AIFFDAT::loopmode1

Definition at line 364 of file csoundCore.h.

6.1.1.8. short AIFFDAT::loopmode2

Definition at line 365 of file csoundCore.h.

6.1.1.9. MYFLT AIFFDAT::natcps

Definition at line 362 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.2. Loris::AiffFile Class Reference

`#include <AiffFile.h>`

6.2.1. Detailed Description

Class AiffFile represents sample data in a AIFF-format samples file, and manages file I/O and sample conversion. Since the sound analysis and synthesis algorithms in Loris and the reassigned bandwidth-enhanced representation are monaural, AiffFile manages only monaural (single channel) AIFF-format samples files.

Definition at line 58 of file AiffFile.h.

Public Types

- typedef std::vector< double > samples_type
- typedef samples_type::size_type size_type
- typedef std::vector< Marker > markers_type

Public Member Functions

- AiffFile (const std::string &filename)
- template<typename Iter> AiffFile (Iter begin_partials, Iter end_partials, double samplerate, double fadeTime=.001)
- AiffFile (double samplerate, size_type numFrames=0)
- AiffFile (const double *buffer, size_type bufferlength, double samplerate)
- AiffFile (const std::vector< double > &vec, double samplerate)
- AiffFile (const AiffFile &other)
- AiffFile & operator= (const AiffFile &rhs)
- markers_type & markers (void)
- const markers_type & markers (void) const
- double midiNoteNumber (void) const
- size_type numFrames (void) const
- double sampleRate (void) const
- samples_type & samples (void)
- const samples_type & samples (void) const
- void addPartial (const Loris::Partial &p, double fadeTime=.001)
- template<typename Iter> void addPartials (Iter begin_partials, Iter end_partials, double fadeTime=.001)
- void setMidiNoteNumber (double nn)
- void write (const std::string &filename, unsigned int bps=16)

Private Member Functions

- void configureSynthesizer (double fadeTime)
- void readAiffData (const std::string &filename)

Private Attributes

- double notenum_
- double rate_
- markers_type markers_
- samples_type samples_
- std::auto_ptr< Synthesizer > psynth_

6.2.2. Member Typedef Documentation

6.2.2.1. typedef std::vector< Marker > Loris::AiffFile::markers_type

The type of AIFF marker storage in an AiffFile.

Definition at line 72 of file AiffFile.h.

6.2.2.2. typedef std::vector< double > Loris::AiffFile::samples_type

The type of the sample storage in an AiffFile.

Definition at line 66 of file AiffFile.h.

6.2.2.3. typedef samples_type::size_type Loris::AiffFile::size_type

The type of all size parameters for AiffFile.

Definition at line 69 of file AiffFile.h.

6.2.3. Constructor & Destructor Documentation

6.2.3.1. Loris::AiffFile::AiffFile (const std::string & *filename*) [explicit]

Initialize an instance of AiffFile by importing sample data from the file having the specified filename or path.

Parameters:

filename is the name or path of an AIFF samples file

6.2.3.2. template<typename Iter> Loris::AiffFile::AiffFile (Iter *begin_partials*, Iter *end_partials*, double *samplerate*, double *fadeTime* = .001)

Initialize an instance of AiffFile with samples rendered from a sequnence of Partials. The Partials in the specified half-open (STL-style) range are rendered at the specified sample rate, using the (optionally) specified Partial fade time (see Synthesizer.h for an examplanation of fade time).

Parameters:

begin_partials is the beginning of a sequence of Partials

end_partials is (one-past) the end of a sequence of Partials

samplerate is the rate at which Partials are rendered

fadeTime is the Partial fade time for rendering the Partials on the specified range. If unspecified, the default fade time is 1 ms.

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only PartialList::const_iterator arguments.

Definition at line 272 of file AiffFile.h.

References addPartials().

6.2.3.3. Loris::AiffFile::AiffFile (double *samplerate*, size_type *numFrames* = 0) [explicit]

Initialize an instance of AiffFile having the specified sample rate, preallocating numFrames samples, initialized to zero.

Parameters:

samplerate is the rate at which Partials are rendered

numFrames is the initial number of (zero) samples. If unspecified, no samples are preallocated.

6.2.3.4. Loris::AiffFile::AiffFile (const double * *buffer*, size_type *bufferlength*, double *samplerate*)

Initialize an instance of AiffFile from a buffer of sample data, with the specified sample rate.

Parameters:

buffer is a pointer to a buffer of floating point samples.

bufferlength is the number of samples in the buffer.

samplerate is the sample rate of the samples in the buffer.

6.2.3.5. Loris::AiffFile::AiffFile (const std::vector< double > & *vec*, double *samplerate*)

Initialize an instance of AiffFile from a vector of sample data, with the specified sample rate.

Parameters:

vec is a vector of floating point samples.

samplerate is the sample rate of the samples in the vector.

6.2.3.6. Loris::AiffFile::AiffFile (const AiffFile & *other*)

Initialize this and AiffFile that is an exact copy, having all the same sample data, as another AiffFile.

Parameters:

other is the AiffFile to copy

6.2.4. Member Function Documentation

6.2.4.1. void Loris::AiffFile::addPartial (const Loris::Partial & *p*, double *fadeTime* = .001)

Render the specified Partial using the (optionally) specified Partial fade time, and accumulate the resulting samples into the sample vector for this AiffFile.

Parameters:

p is the partial to render into this AiffFile

fadeTime is the Partial fade time for rendering the Partials on the specified range. If unspecified, the default fade time is 1 ms.

6.2.4.2. template<typename Iter> void Loris::AiffFile::addPartials (Iter *begin_partials*, Iter *end_partials*, double *fadeTime* = .001)

Accumulate samples rendered from a sequence of Partials. The Partials in the specified half-open (STL-style) range are rendered at this AiffFile's sample rate, using the (optionally) specified Partial fade time (see Synthesizer.h for an examplanation of fade time).

Parameters:

begin_partials is the beginning of a sequence of Partials

end_partials is (one-past) the end of a sequence of Partials

fadeTime is the Partial fade time for rendering the Partials on the specified range. If unspecified, the default fade time is 1 ms.

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only PartialList::const_iterator arguments.

Definition at line 308 of file AiffFile.h.

Referenced by AiffFile().

6.2.4.3. void Loris::AiffFile::configureSynthesizer (double *fadeTime*) [private]

6.2.4.4. const markers_type& Loris::AiffFile::markers (void) const

Return a const reference to the Marker (see Marker.h) container for this AiffFile.

6.2.4.5. markers_type& Loris::AiffFile::markers (void)

Return a reference to the Marker (see Marker.h) container for this AiffFile.

6.2.4.6. double Loris::AiffFile::midiNoteNumber (void) const

Return the fractional MIDI note number assigned to this AiffFile. If the sound has no definable pitch, note number 60.0 is used.

6.2.4.7. size_type Loris::AiffFile::numFrames (void) const

Return the number of sample frames represented in this AiffFile. A sample frame contains one sample per channel for a single sample interval (e.g. mono and stereo samples files having a sample rate of 44100 Hz both have 44100 sample frames per second of audio samples).

6.2.4.8. AiffFile& Loris::AiffFile::operator= (const AiffFile & *rhs*)

Assignment operator: change this AiffFile to be an exact copy of the specified AiffFile, rhs, that is, having the same sample data.

Parameters:

rhs is the AiffFile to replicate

6.2.4.9. void Loris::AiffFile::readAiffData (const std::string & *filename*) [private]

6.2.4.10. double Loris::AiffFile::sampleRate (void) const

Return the sampling freqency in Hz for the sample data in this AiffFile.

6.2.4.11. const samples_type& Loris::AiffFile::samples (void) const

Return a const reference (or const reference) to the vector containing the floating-point sample data for this AiffFile.

6.2.4.12. samples_type& Loris::AiffFile::samples (void)

Return a reference (or const reference) to the vector containing the floating-point sample data for this AiffFile.

6.2.4.13. void Loris::AiffFile::setMidiNoteNumber (double *nn*)

Set the fractional MIDI note number assigned to this AiffFile. If the sound has no definable pitch, use note number 60.0 (the default).

Parameters:

nn is a fractional MIDI note number, 60 is middle C.

6.2.4.14. void Loris::AiffFile::write (const std::string & *filename*, unsigned int *bps* = 16)

Export the sample data represented by this AiffFile to the file having the specified filename or path. Export signed integer samples of the specified size, in bits (8, 16, 24, or 32).

Parameters:

filename is the name or path of the AIFF samples file to be created or overwritten.

bps is the number of bits per sample to store in the samples file (8, 16, 24, or 32).If unspeicified, 16 bits

6.2.5. Member Data Documentation

6.2.5.1. markers_type Loris::AiffFile::markers_ [private]

Definition at line 235 of file AiffFile.h.

6.2.5.2. double Loris::AiffFile::notenum_ [private]

Definition at line 234 of file AiffFile.h.

6.2.5.3. std::auto_ptr< Synthesizer > Loris::AiffFile::psynth_ [private]

Definition at line 238 of file AiffFile.h.

6.2.5.4. double Loris::AiffFile::rate_ [private]

Definition at line 234 of file AiffFile.h.

6.2.5.5. samples_type Loris::AiffFile::samples_ [private]

Definition at line 236 of file AiffFile.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/AiffFile.h

6.3. Loris::PartialUtils::AmplitudeScaler Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::AmplitudeScaler::

6.3.1. Detailed Description

Scale the amplitude of the specified Partial according to an envelope representing a time-varying amplitude scale value.

Definition at line 103 of file PartialUtils.h.

Public Member Functions

- AmplitudeScaler (double x)
- AmplitudeScaler (const Envelope &e)
- void operator() (Partial &p) const

Protected Attributes

- Envelope * env

6.3.2. Constructor & Destructor Documentation

6.3.2.1. Loris::PartialUtils::AmplitudeScaler::AmplitudeScaler (double *x*) [inline]

Construct a new AmplitudeScaler from a constant scale factor.

Definition at line 108 of file PartialUtils.h.

6.3.2.2. Loris::PartialUtils::AmplitudeScaler::AmplitudeScaler (const Envelope & *e*) [inline]

Construct a new AmplitudeScaler from an Envelope representing a time-varying scale factor.

Definition at line 112 of file PartialUtils.h.

6.3.3. Member Function Documentation

6.3.3.1. void Loris::PartialUtils::AmplitudeScaler::operator() (Partial & *p*) const [virtual]

Function call operator: apply a scale factor to the specified Partial.

Implements Loris::PartialUtils::PartialMutator.

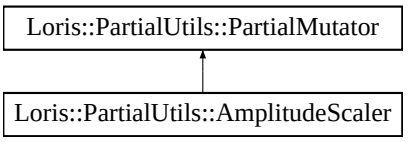
6.3.4. Member Data Documentation

6.3.4.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected, inherited]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::AmplitudeScaler

6.4. Loris::Analyzer Class Reference

`#include <Analyzer.h>`

6.4.1. Detailed Description

Class Analyzer represents a configuration of parameters for performing Reassigned Bandwidth-Enhanced Additive Analysis of sampled sounds. The analysis process yields a collection of Partials, each having a trio of synchronous, non-uniformly- sampled breakpoint envelopes representing the time-varying frequency, amplitude, and noisiness of a single bandwidth- enhanced sinusoid. These Partials are accumulated in the Analyzer.

The core analysis parameter is the frequency resolution, the minimum instantaneous frequency spacing between partials. All other parameters are initially configured according to this parameter (and the analysis window width, if specified). Subsequent parameter mutations are independent.

For more information about Reassigned Bandwidth-Enhanced Analysis and the Reassigned Bandwidth-Enhanced Additive Sound Model, refer to the Loris website: www.cerlsoundgroup.org/Loris/.

Definition at line 68 of file Analyzer.h.

Public Member Functions

- Analyzer (double resolutionHz)
- Analyzer (double resolutionHz, double windowWidthHz)
- Analyzer (const Analyzer &other)
- ~Analyzer (void)
- Analyzer & operator= (const Analyzer &rhs)
- void configure (double resolutionHz, double windowWidthHz)
- void analyze (const std::vector< double > &vec, double srate)
- void analyze (const double *bufBegin, const double *bufEnd, double srate)
- void analyze (const std::vector< double > &vec, double srate, const Envelope &reference)
- void analyze (const double *bufBegin, const double *bufEnd, double srate, const Envelope &reference)
- double ampFloor (void) const
- bool associateBandwidth (void) const
- double bwRegionWidth (void) const
- double cropTime (void) const
- double freqDrift (void) const
- double freqFloor (void) const
- double freqResolution (void) const
- double hopTime (void) const
- double sidelobeLevel (void) const
- double windowWidth (void) const
- void setAmpFloor (double x)
- void setBwRegionWidth (double x)
- void setCropTime (double x)
- void setFreqDrift (double x)
- void setFreqFloor (double x)
- void setFreqResolution (double x)
- void setHopTime (double x)
- void setSidelobeLevel (double x)
- void setWindowWidth (double x)

- PartialList & partials (void)
- const PartialList & partials (void) const

Private Attributes

- std::auto_ptr< Analyzer_imp > _imp

6.4.2. Constructor & Destructor Documentation

6.4.2.1. Loris::Analyzer::Analyzer (double *resolutionHz*) [explicit]

insulating implementation class

Construct a new Analyzer configured with the given frequency resolution (minimum instantaneous frequency difference between Partials). All other Analyzer parameters are computed from the specified frequency resolution.

Parameters:

resolutionHz is the frequency resolution in Hz.

6.4.2.2. Loris::Analyzer::Analyzer (double *resolutionHz*, double *windowWidthHz*)

Construct a new Analyzer configured with the given frequency resolution (minimum instantaneous frequency difference between Partials) and analysis window width (main lobe, zero-to-zero). All other Analyzer parameters are computed from the specified resolution and window width.

Parameters:

resolutionHz is the frequency resolution in Hz.

windowWidthHz is the main lobe width of the Kaiser analysis window in Hz.

6.4.2.3. Loris::Analyzer::Analyzer (const Analyzer & *other*)

Construct a new Analyzer having identical parameter configuration to another Analyzer. The list of collected Partials is not copied.

Parameters:

other is the Analyzer to copy.

6.4.2.4. Loris::Analyzer::~Analyzer (void)

Destroy this Analyzer.

6.4.3. Member Function Documentation

6.4.3.1. double Loris::Analyzer::ampFloor (void) const

Return the amplitude floor (lowest detected spectral amplitude), in (negative) dB, for this Analyzer.

6.4.3.2. void Loris::Analyzer::analyze (const double * *bufBegin*, const double * *bufEnd*, double *srate*, const Envelope & *reference*)

Analyze a range of (mono) samples at the given sample rate (in Hz) and append the extracted Partials to Analyzer's PartialList (std::list of Partials). Use the specified envelope as a frequency reference for Partial tracking.

Parameters:

bufBegin is a pointer to a buffer of floating point samples

bufEnd is (one-past) the end of a buffer of floating point samples

srate is the sample rate of the samples in the buffer

reference is an Envelope having the approximate frequency contour expected of the resulting Partials.

6.4.3.3. void Loris::Analyzer::analyze (const std::vector< double > & *vec*, double *srate*, const Envelope & *reference*)

Analyze a vector of (mono) samples at the given sample rate (in Hz) and append the extracted Partials to Analyzer's PartialList (std::list of Partials). Use the specified envelope as a frequency reference for Partial tracking.

Parameters:

vec is a vector of floating point samples

srate is the sample rate of the samples in the vector

reference is an Envelope having the approximate frequency contour expected of the resulting Partials.

6.4.3.4. void Loris::Analyzer::analyze (const double * *bufBegin*, const double * *bufEnd*, double *srate*)

Analyze a range of (mono) samples at the given sample rate (in Hz) and collect the resulting Partials.

Parameters:

bufBegin is a pointer to a buffer of floating point samples

bufEnd is (one-past) the end of a buffer of floating point samples

srate is the sample rate of the samples in the buffer

6.4.3.5. void Loris::Analyzer::analyze (const std::vector< double > & *vec*, double *srate*)

Analyze a vector of (mono) samples at the given sample rate (in Hz) and append the extracted Partials to Analyzer's PartialList (std::list of Partials).

Parameters:

vec is a vector of floating point samples

srate is the sample rate of the samples in the vector

6.4.3.6. bool Loris::Analyzer::associateBandwidth (void) const

Return true if this Analyzer is configured to peform bandwidth association to distribute noise energy among extracted Partials, and false if noise energy will be collected in noise Partials, labeled -1 in this Analyzer's PartialList.

6.4.3.7. double Loris::Analyzer::bwRegionWidth (void) const

Return the width (in Hz) of the Bandwidth Association regions used by this Analyzer. If zero, bandwidth enhancement is disabled.

6.4.3.8. void Loris::Analyzer::configure (double *resolutionHz*, double *windowWidthHz*)

Configure this Analyzer with the given frequency resolution (minimum instantaneous frequency difference between Partials) and analysis window width (main lobe, zero-to-zero, in Hz). All other Analyzer parameters are (re-)computed from the frequency resolution and window width.

Parameters:

resolutionHz is the frequency resolution in Hz.

windowWidthHz is the main lobe width of the Kaiser analysis window in Hz.

There are three categories of analysis parameters:

- the resolution, and params that are usually related to (or identical to) the resolution (frequency floor and drift)
- the window width and params that are usually related to (or identical to) the window width (hop and crop times)
- independent parameters (bw region width and amp floor)

6.4.3.9. double Loris::Analyzer::cropTime (void) const

Return the crop time (maximum temporal displacement of a time- frequency data point from the time-domain center of the analysis window, beyond which data points are considered "unreliable") for this Analyzer.

6.4.3.10. double Loris::Analyzer::freqDrift (void) const

Return the maximum allowable frequency difference between consecutive Breakpoints in a Partial envelope for this Analyzer.

6.4.3.11. double Loris::Analyzer::freqFloor (void) const

Return the frequency floor (minimum instantaneous Partial frequency), in Hz, for this Analyzer.

6.4.3.12. double Loris::Analyzer::freqResolution (void) const

Return the frequency resolution (minimum instantaneous frequency difference between Partials) for this Analyzer.

6.4.3.13. double Loris::Analyzer::hopTime (void) const

Return the hop time (which corresponds approximately to the average density of Partial envelope Breakpoint data) for this Analyzer.

6.4.3.14. Analyzer& Loris::Analyzer::operator= (const Analyzer & *rhs*)

Construct a new Analyzer having identical parameter configuration to another Analyzer. The list of collected Partials is not copied.

Parameters:

rhs is the Analyzer to copy.

6.4.3.15. const PartialList& Loris::Analyzer::partials (void) const

Return an immutable (const) reference to this Analyzer's list of analyzed Partials.

6.4.3.16. PartialList& Loris::Analyzer::partials (void)

Return a mutable reference to this Analyzer's list of analyzed Partials.

6.4.3.17. void Loris::Analyzer::setAmpFloor (double *x*)

Set the amplitude floor (lowest detected spectral amplitude), in (negative) dB, for this Analyzer.

Parameters:

x is the new value of this parameter.

6.4.3.18. void Loris::Analyzer::setBwRegionWidth (double *x*)

Set the width (in Hz) of the Bandwidth Association regions used by this Analyzer. If zero, bandwidth enhancement is disabled.

Parameters:

x is the new value of this parameter.

6.4.3.19. void Loris::Analyzer::setCropTime (double *x*)

Set the crop time (maximum temporal displacement of a time- frequency data point from the time-domain center of the analysis window, beyond which data points are considered "unreliable") for this Analyzer.

Parameters:

x is the new value of this parameter.

6.4.3.20. void Loris::Analyzer::setFreqDrift (double *x*)

Set the maximum allowable frequency difference between consecutive Breakpoints in a Partial envelope for this Analyzer.

Parameters:
x is the new value of this parameter.

6.4.3.21. void Loris::Analyzer::setFreqFloor (double *x*)

Set the frequency floor (minimum instantaneous Partial frequency), in Hz, for this Analyzer.

Parameters:
x is the new value of this parameter.

6.4.3.22. void Loris::Analyzer::setFreqResolution (double *x*)

Set the frequency resolution (minimum instantaneous frequency difference between Partials) for this Analyzer. (Does not cause other parameters to be recomputed.)

Parameters:
x is the new value of this parameter.

6.4.3.23. void Loris::Analyzer::setHopTime (double *x*)

Set the hop time (which corresponds approximately to the average density of Partial envelope Breakpoint data) for this Analyzer.

Parameters:
x is the new value of this parameter.

6.4.3.24. void Loris::Analyzer::setSidelobeLevel (double *x*)

Set the sidelobe attenutation level for the Kaiser analysis window in positive dB. More negative numbers (e.g. -90) give very good sidelobe rejection but cause the window to be longer in time. Less negative numbers raise the level of the sidelobes, increasing the likelihood of frequency-domain interference, but allow the window to be shorter in time.

Parameters:
x is the new value of this parameter.

6.4.3.25. void Loris::Analyzer::setWindowWidth (double *x*)

Set the frequency-domain main lobe width (measured between zero-crossings) of the analysis window used by this Analyzer.

Parameters:
x is the new value of this parameter.

6.4.3.26. double Loris::Analyzer::sidelobeLevel (void) const

Return the sidelobe attenutation level for the Kaiser analysis window in positive dB. Larger numbers (e.g. 90) give very good sidelobe rejection but cause the window to be longer in time. Smaller numbers (like 60) raise the level of the sidelobes, increasing the likelihood of frequency-domain interference, but allow the window to be shorter in time.

6.4.3.27. double Loris::Analyzer::windowWidth (void) const

Return the frequency-domain main lobe width (measured between zero-crossings) of the analysis window used by this Analyzer.

6.4.4. Member Data Documentation

6.4.4.1. std::auto_ptr< Analyzer_imp > Loris::Analyzer::_imp [private]

Definition at line 71 of file Analyzer.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Analyzer.h

6.5. arglst Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int count
- char * arg [1]

6.5.1. Member Data Documentation

6.5.1.1. char* arglst::arg[1]

Definition at line 156 of file csoundCore.h.

6.5.1.2. int arglst::count

Definition at line 155 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.6. argoffs Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int count
- int indx [1]

6.6.1. Member Data Documentation

6.6.1.1. int argoffs::count

Definition at line 160 of file csoundCore.h.

6.6.1.2. int argoffs::indx[1]

Definition at line 161 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.7. Loris::AssertionFailure Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::AssertionFailure::

6.7.1. Detailed Description

Class of exceptions thrown when an assertion (usually representing an invariant condition, and usually detected by the Assert macro) is violated.

Definition at line 112 of file Exception.h.

Public Member Functions

- AssertionFailure (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.7.2. Constructor & Destructor Documentation

6.7.2.1. Loris::AssertionFailure::AssertionFailure (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 125 of file Exception.h.

6.7.3. Member Function Documentation

6.7.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.7.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.7.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.7.4. Member Data Documentation

6.7.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

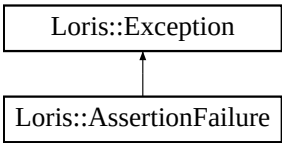
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::AssertionFailure

6.8. Loris::AssociateBandwidth Class Reference

`#include <AssociateBandwidth.h>`

Public Member Functions

- AssociateBandwidth (double regionWidth, double srate)
- ~AssociateBandwidth (void)
- void associateBandwidth (Peaks::iterator begin, Peaks::iterator rejected, Peaks::iterator end)

Private Member Functions

- double computeNoiseEnergy (double freq, double amp)
- void accumulateNoise (double freq, double amp)
- void accumulateSinusoid (double freq, double amp)
- void associate (Breakpoint &bp)
- void reset (void)

Private Attributes

- std::vector< double > _weights
- std::vector< double > _surplus
- double _regionRate

6.8.1. Constructor & Destructor Documentation

6.8.1.1. Loris::AssociateBandwidth::AssociateBandwidth (double *regionWidth*, double *srate*)

6.8.1.2. Loris::AssociateBandwidth::~AssociateBandwidth (void)

6.8.2. Member Function Documentation

6.8.2.1. void Loris::AssociateBandwidth::accumulateNoise (double *freq*, double *amp*) [private]

6.8.2.2. void Loris::AssociateBandwidth::accumulateSinusoid (double *freq*, double *amp*) [private]

6.8.2.3. void Loris::AssociateBandwidth::associate (Breakpoint & *bp*) [private]

6.8.2.4. void Loris::AssociateBandwidth::associateBandwidth (Peaks::iterator *begin*, Peaks::iterator *rejected*, Peaks::iterator *end*)

6.8.2.5. double Loris::AssociateBandwidth::computeNoiseEnergy (double *freq*, double *amp*) [private]

6.8.2.6. void Loris::AssociateBandwidth::reset (void) [private]

6.8.3. Member Data Documentation

6.8.3.1. double Loris::AssociateBandwidth::_regionRate [private]

Definition at line 68 of file AssociateBandwidth.h.

6.8.3.2. std::vector< double > Loris::AssociateBandwidth::_surplus [private]

Definition at line 63 of file AssociateBandwidth.h.

6.8.3.3. std::vector< double > Loris::AssociateBandwidth::_weights [private]

Definition at line 60 of file AssociateBandwidth.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/AssociateBandwidth.h

6.9. auxch Struct Reference

`#include <csoundCore.h>`

Public Attributes

- auxch * nxtchp
- long size
- void * auxp
- void * endp

6.9.1. Member Data Documentation

6.9.1.1. void* auxch::auxp

Definition at line 226 of file csoundCore.h.

6.9.1.2. void * auxch::endp

Definition at line 226 of file csoundCore.h.

6.9.1.3. struct auxch* auxch::nxtchp

Definition at line 224 of file csoundCore.h.

6.9.1.4. long auxch::size

Definition at line 225 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.10. Loris::PartialUtils::BandwidthScaler Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::BandwidthScaler::

6.10.1. Detailed Description

Scale the bandwidth of the specified Partial according to an envelope representing a time-varying bandwidth scale value.

Definition at line 163 of file PartialUtils.h.

Public Member Functions

- BandwidthScaler (double x)
- BandwidthScaler (const Envelope &e)
- void operator() (Partial &p) const

Protected Attributes

- Envelope * env

6.10.2. Constructor & Destructor Documentation

6.10.2.1. Loris::PartialUtils::BandwidthScaler::BandwidthScaler (double *x*) [inline]

Construct a new BandwidthScaler from a constant scale factor.

Definition at line 168 of file PartialUtils.h.

6.10.2.2. Loris::PartialUtils::BandwidthScaler::BandwidthScaler (const Envelope & *e*) [inline]

Construct a new BandwidthScaler from an Envelope representing a time-varying scale factor.

Definition at line 172 of file PartialUtils.h.

6.10.3. Member Function Documentation

6.10.3.1. void Loris::PartialUtils::BandwidthScaler::operator() (Partial & *p*) const [virtual]

Function call operator: apply a scale factor to the specified Partial.

Implements Loris::PartialUtils::PartialMutator.

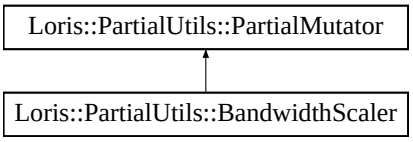
6.10.4. Member Data Documentation

6.10.4.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected, inherited]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::BandwidthScaler

6.11. Loris::BigEndian Class Reference

`#include <Endian.h>`

Static Public Member Functions

- static void read (std::istream &s, long howmany, int size, char *putemHere)
- static void write (std::ostream &s, long howmany, int size, const char *stuff)

6.11.1. Member Function Documentation

6.11.1.1. static void Loris::BigEndian::read (std::istream & *s*, long *howmany*, int *size*, char * *putemHere*) [static]

6.11.1.2. static void Loris::BigEndian::write (std::ostream & *s*, long *howmany*, int *size*, const char * *stuff*) [static]

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Endian.h

6.12. Loris::Breakpoint Class Reference

`#include <Breakpoint.h>`

6.12.1. Detailed Description

Class Breakpoint represents a single breakpoint in the Partial parameter (frequency, amplitude, bandwidth) envelope. Instantaneous phase is also stored, but is only used at the onset of a partial, or when it makes a transition from zero to nonzero amplitude.

Loris Partials represent reassigned bandwidth-enhanced model components. A Partial consists of a chain of Breakpoints describing the time-varying frequency, amplitude, and bandwidth (noisiness) of the component. For more information about Reassigned Bandwidth-Enhanced Analysis and the Reassigned Bandwidth-Enhanced Additive Sound Model, refer to the Loris website: www.cerlsoundgroup.org/Loris/.

Breakpoint is a leaf class, do not subclass.

Definition at line 58 of file Breakpoint.h.

Public Member Functions

- Breakpoint (void)
- Breakpoint (double f, double a, double b, double p=0.)
- double amplitude (void) const
- double bandwidth (void) const
- double frequency (void) const
- double phase (void) const
- void setAmplitude (double x)
- void setBandwidth (double x)
- void setFrequency (double x)
- void setPhase (double x)
- void addNoiseEnergy (double enoise)

Private Attributes

- double _frequency
- double _amplitude
- double _bandwidth
- double _phase

6.12.2. Constructor & Destructor Documentation

6.12.2.1. Loris::Breakpoint::Breakpoint (void)

radians

Construct a new Breakpoint with all parameters initialized to 0 (needed for STL containability).

6.12.2.2. Loris::Breakpoint::Breakpoint (double *f*, double *a*, double *b*, double *p* = 0.)

Construct a new Breakpoint with the specified parameters.

Parameters:

f is the intial frequency.

a is the initial amplitude.

b is the initial bandwidth.

p is the initial phase, if specified (if unspecified, 0 is assumed).

6.12.3. Member Function Documentation

6.12.3.1. void Loris::Breakpoint::addNoiseEnergy (double *enoise*)

Add noise (bandwidth) energy to this Breakpoint by computing new amplitude and bandwidth values. enoise may be negative, but noise energy cannot be removed (negative energy added) in excess of the current noise energy.

Parameters:

enoise is the amount of noise energy to add to this Breakpoint.

6.12.3.2. double Loris::Breakpoint::amplitude (void) const [inline]

Return the amplitude of this Breakpoint.

Definition at line 88 of file Breakpoint.h.

References _amplitude.

Referenced by Loris::BreakpointUtils::greater_amplitude::operator()().

6.12.3.3. double Loris::Breakpoint::bandwidth (void) const [inline]

Return the bandwidth (noisiness) coefficient of this Breakpoint.

Definition at line 91 of file Breakpoint.h.

References _bandwidth.

6.12.3.4. double Loris::Breakpoint::frequency (void) const [inline]

Return the frequency of this Breakpoint.

Definition at line 94 of file Breakpoint.h.

References _frequency.

Referenced by Loris::BreakpointUtils::less_frequency::operator()().

6.12.3.5. double Loris::Breakpoint::phase (void) const [inline]

Return the phase of this Breakpoint.

Definition at line 97 of file Breakpoint.h.

References _phase.

6.12.3.6. void Loris::Breakpoint::setAmplitude (double *x*) [inline]

Set the amplitude of this Breakpoint.

Parameters:
x is the new amplitude

Definition at line 103 of file Breakpoint.h.

References _amplitude.

6.12.3.7. void Loris::Breakpoint::setBandwidth (double *x*) [inline]

Set the bandwidth (noisiness) coefficient of this Breakpoint.

Parameters:
x is the new bandwidth

Definition at line 108 of file Breakpoint.h.

References _bandwidth.

6.12.3.8. void Loris::Breakpoint::setFrequency (double *x*) [inline]

Set the frequency of this Breakpoint.

Parameters:
x is the new frequency.

Definition at line 113 of file Breakpoint.h.

References _frequency.

6.12.3.9. void Loris::Breakpoint::setPhase (double *x*) [inline]

Set the phase of this Breakpoint.

Parameters:
x is the new phase.

Definition at line 118 of file Breakpoint.h.

References _phase.

6.12.4. Member Data Documentation

6.12.4.1. double Loris::Breakpoint::_amplitude [private]

in Hertz

Definition at line 62 of file Breakpoint.h.

Referenced by amplitude(), and setAmplitude().

6.12.4.2. double Loris::Breakpoint::_bandwidth [private]

absolute

Definition at line 63 of file Breakpoint.h.

Referenced by bandwidth(), and setBandwidth().

6.12.4.3. double Loris::Breakpoint::_frequency [private]

Definition at line 61 of file Breakpoint.h.

Referenced by frequency(), and setFrequency().

6.12.4.4. double Loris::Breakpoint::_phase [private]

fraction of total energy that is noise energy

Definition at line 64 of file Breakpoint.h.

Referenced by phase(), and setPhase().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Breakpoint.h

6.13. Loris::BreakpointEnvelope Class Reference

`#include <BreakpointEnvelope.h>`

Inheritance diagram for Loris::BreakpointEnvelope::

6.13.1. Detailed Description

A BreakpointEnvelope represents a linear segment breakpoint function with infinite extension at each end (that is, evalutaing the envelope past either end of the breakpoint function yields the value at the nearest end point).

BreakpointEnvelope implements the Envelope interface, described by the abstract class Envelope.

BreakpointEnvelope inherits the types

- `size_type`
- `value_type`
- `iterator`
- `const_iterator`

and the member functions

- size_type size(void) const
- bool empty(void) const
- iterator begin(void)
- const_iterator begin(void) const
- iterator end(void)
- const_iterator end(void) const

from std::map< double, double >.

Definition at line 69 of file BreakpointEnvelope.h.

Public Member Functions

- BreakpointEnvelope (void)
- BreakpointEnvelope (double initialValue)
- virtual BreakpointEnvelope * clone (void) const
- virtual double valueAt (double t) const
- void insert (double time, double value)
- void insertBreakpoint (double time, double value)

6.13.2. Constructor & Destructor Documentation

6.13.2.1. Loris::BreakpointEnvelope::BreakpointEnvelope (void)

Construct a new BreakpointEnvelope having no breakpoints (and an implicit value of 0 everywhere).

6.13.2.2. Loris::BreakpointEnvelope::BreakpointEnvelope (double *initialValue*) [explicit]

Construct and return a new BreakpointEnvelope having a single breakpoint at 0 (and an implicit value everywhere) of initialValue.

Parameters:

initialValue is the value of this BreakpointEnvelope at time 0.

6.13.3. Member Function Documentation

6.13.3.1. virtual BreakpointEnvelope* Loris::BreakpointEnvelope::clone (void) const [virtual]

Return an exact copy of this BreakpointEnvelope (polymorphic copy, following the Prototype pattern).

Implements Loris::Envelope.

6.13.3.2. void Loris::BreakpointEnvelope::insert (double *time*, double *value*)

Insert a breakpoint representing the specified (time, value) pair into this BreakpointEnvelope. If there is already a breakpoint at the specified time, it will be replaced with the new breakpoint.

Parameters:

time is the time at which to insert a new breakpoint

value is the value of the new breakpoint

Referenced by insertBreakpoint().

6.13.3.3. void Loris::BreakpointEnvelope::insertBreakpoint (double *time*, double *value*) [inline]

Insert a breakpoint representing the specified (time, value) pair into this BreakpointEnvelope. Same as insert, retained for backwards-compatibility.

Definition at line 116 of file BreakpointEnvelope.h.

References insert().

6.13.3.4. virtual double Loris::BreakpointEnvelope::valueAt (double *t*) const [virtual]

Return the linearly-interpolated value of this BreakpointEnvelope at the specified time.

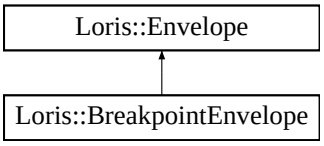
Parameters:

t is the time at which to evaluate this BreakpointEnvelope.

Implements Loris::Envelope.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/BreakpointEnvelope.h


Loris::Envelope
Loris::BreakpointEnvelope

6.14. Canvas Class Reference

`#include <canvas.hpp>`

Public Member Functions

- Canvas (int x, int y, int w, int h, const char *label=0)
- void set_curve (Curve *)

Private Member Functions

- void draw ()

Private Attributes

- Curve * m_curve

6.14.1. Constructor & Destructor Documentation

6.14.1.1. Canvas::Canvas (int *x*, int *y*, int *w*, int *h*, const char * *label* = 0)

6.14.2. Member Function Documentation

6.14.2.1. void Canvas::draw () [private]

6.14.2.2. void Canvas::set_curve (Curve *)

6.14.3. Member Data Documentation

6.14.3.1. Curve* Canvas::m_curve [private]

Definition at line 30 of file canvas.hpp.

The documentation for this class was generated from the following file:

- frontends/flcsound/canvas.hpp

6.15. csound::Cell Class Reference

`#include <Cell.hpp>`

Inheritance diagram for csound::Cell::

6.15.1. Detailed Description

Score node that simplifies building up repetitive and overlapping motivic cells, such as used in Minimalism.

Definition at line 41 of file Cell.hpp.

Public Member Functions

- Cell ()
- virtual ~Cell ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- int repeatCount
- bool relativeDuration
- double durationSeconds
- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- Score score
- ublas::matrix< double > localCoordinates

6.15.2. Constructor & Destructor Documentation

6.15.2.1. csound::Cell::Cell ()

6.15.2.2. virtual csound::Cell::~Cell () [virtual]

6.15.3. Member Function Documentation

6.15.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.15.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.15.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.15.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.15.3.5. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.15.3.6. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.15.3.7. virtual void csound::Cell::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::ScoreNode.

6.15.3.8. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.15.3.9. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.15.4. Member Data Documentation

6.15.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.15.4.2. double csound::Cell::durationSeconds

Definition at line 47 of file Cell.hpp.

6.15.4.3. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.15.4.4. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.15.4.5. bool csound::Cell::relativeDuration

Definition at line 46 of file Cell.hpp.

6.15.4.6. int csound::Cell::repeatCount

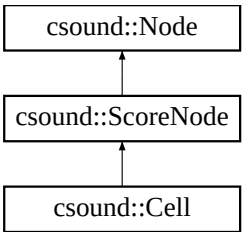
Definition at line 45 of file Cell.hpp.

6.15.4.7. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Cell.hpp


csound::Node
csound::ScoreNode
csound::Cell

6.16. Loris::Channelizer Class Reference

`#include <Channelizer.h>`

6.16.1. Detailed Description

Class Channelizer represents an algorithm for automatic labeling of a sequence of Partials. Partials must be labeled in preparation for morphing (see Morpher) to establish correspondences between Partials in the morph source and target sounds.

Channelized partials are labeled according to their adherence to a harmonic frequency structure with a time-varying fundamental frequency. The frequency spectrum is partitioned into non-overlapping channels having time-varying center frequencies that are harmonic (integer) multiples of a specified reference frequency envelope, and each channel is identified by a unique label equal to its harmonic number. Each Partial is assigned the label corresponding to the channel containing the greatest portion of its (the Partial's) energy.

A reference frequency Envelope for channelization and the channel number to which it corresponds (1 for an Envelope that tracks the Partial at the fundamental frequency) must be specified. The reference Envelope can be constructed explcitly, point by point (using, for example, the Breakpoint-Envelope class), or constructed automatically using the FrequencyReference class.

Channelizer is a leaf class, do not subclass.

Definition at line 75 of file Channelizer.h.

Public Member Functions

- Channelizer (const Envelope &refChanFreq, int refChanLabel)
- Channelizer (const Channelizer &other)
- Channelizer & operator= (const Channelizer &rhs)
- ~Channelizer (void)
- void channelize (Partial &partial) const
- template<typename Iter> void channelize (Iter begin, Iter end) const
- template<typename Iter> void operator() (Iter begin, Iter end) const

Static Public Member Functions

- template<typename Iter> static void channelize (Iter begin, Iter end, const Envelope &refChanFreq, int refChanLabel)

Private Attributes

- std::auto_ptr< Envelope > _refChannelFreq
- int _refChannelLabel

6.16.2. Constructor & Destructor Documentation

6.16.2.1. Loris::Channelizer::Channelizer (const Envelope & *refChanFreq*, int *refChanLabel*)

Exceptions:

InvalidArgument if refChanLabel is not positive.

Construct a new Channelizer using the specified reference Envelope to represent the a numbered channel.

Parameters:

refChanFreq is an Envelope representing the center frequency of a channel.

refChanLabel is the corresponding channel number (i.e. 1 if refChanFreq is the lowest-frequency channel, and all other channels are harmonics of refChanFreq, or 2 if refChanFreq tracks the second harmonic, etc.).

6.16.2.2. Loris::Channelizer::Channelizer (const Channelizer & *other*)

Construct a new Channelizer that is an exact copy of another. The copy represents the same set of frequency channels, constructed from the same reference Envelope and channel number.

Parameters:

other is the Channelizer to copy

6.16.2.3. Loris::Channelizer::~Channelizer (void)

Destroy this Channelizer.

6.16.3. Member Function Documentation

6.16.3.1. template<typename Iter> void Loris::Channelizer::channelize (Iter *begin*, Iter *end*, const Envelope & *refChanFreq*, int *refChanLabel*) [static]

Static member that constructs an instance and applies it to a sequence of Partials. Construct a Channelizer using the specified Envelope and reference label, and use it to channelize a sequence of Partials.

Parameters:

begin is the beginning of a sequence of Partials to channelize.

end is the end of a sequence of Partials to channelize.

refChanFreq is an Envelope representing the center frequency of a channel.

refChanLabel is the corresponding channel number (i.e. 1 if refChanFreq is the lowest-frequency channel, and all other channels are harmonics of refChanFreq, or 2 if refChanFreq tracks the second harmonic, etc.).

Exceptions:

InvalidArgument if refChanLabel is not positive.

If compiled with NO_TEMPLATE_MEMBERS defined, then begin and end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 240 of file Channelizer.h.

References channelize().

6.16.3.2. template<typename Iter> void Loris::Channelizer::channelize (Iter *begin*, Iter *end*) const

Assign each Partial in the specified half-open (STL-style) range the label corresponding to the frequency channel containing the greatest portion of its (the Partial's) energy.

Parameters:

begin is the beginning of the range of Partials to channelize

end is (one-past) the end of the range of Partials to channelize

If compiled with NO_TEMPLATE_MEMBERS defined, then begin and end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 201 of file Channelizer.h.

6.16.3.3. void Loris::Channelizer::channelize (Partial & *partial*) const

Label a Partial with the number of the frequency channel containing the greatest portion of its (the Partial's) energy.

Parameters:

partial is the Partial to label.

Referenced by channelize(), and operator()().

6.16.3.4. template<typename Iter> void Loris::Channelizer::operator() (Iter *begin*, Iter *end*) const [inline]

Function call operator: same as channelize().

Definition at line 142 of file Channelizer.h.

References channelize().

6.16.3.5. Channelizer& Loris::Channelizer::operator= (const Channelizer & *rhs*)

Assignment operator: make this Channelizer an exact copy of another. This Channelizer is made to represent the same set of frequency channels, constructed from the same reference Envelope and channel number as rhs.

Parameters:

rhs is the Channelizer to copy

6.16.4. Member Data Documentation

6.16.4.1. std::auto_ptr< Envelope > Loris::Channelizer::_refChannelFreq [private]

Definition at line 78 of file Channelizer.h.

6.16.4.2. int Loris::Channelizer::_refChannelLabel [private]

Definition at line 79 of file Channelizer.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Channelizer.h

6.17. csound::Chunk Class Reference

`#include <Midifile.hpp>`

Inheritance diagram for csound::Chunk::

Public Member Functions

- Chunk (const char *_id)
- virtual ~Chunk (void)
- virtual void read (std::istream &stream)
- virtual void write (std::ostream &stream)
- virtual void markChunkSize (std::ostream &stream)
- virtual void markChunkStart (std::ostream &stream)
- virtual void markChunkEnd (std::ostream &stream)

Public Attributes

- int id
- int chunkSize
- int chunkSizePosition
- int chunkStart
- int chunkEnd

6.17.1. Constructor & Destructor Documentation

6.17.1.1. csound::Chunk::Chunk (const char * *_id*)

6.17.1.2. virtual csound::Chunk::~Chunk (void) [virtual]

6.17.2. Member Function Documentation

6.17.2.1. virtual void csound::Chunk::markChunkEnd (std::ostream & *stream*) [virtual]

6.17.2.2. virtual void csound::Chunk::markChunkSize (std::ostream & *stream*) [virtual]

6.17.2.3. virtual void csound::Chunk::markChunkStart (std::ostream & *stream*) [virtual]

6.17.2.4. virtual void csound::Chunk::read (std::istream & *stream*) [virtual]

Reimplemented in csound::MidiHeader.

6.17.2.5. virtual void csound::Chunk::write (std::ostream & *stream*) [virtual]

Reimplemented in csound::MidiHeader.

6.17.3. Member Data Documentation

6.17.3.1. int csound::Chunk::chunkEnd

Definition at line 60 of file Midifile.hpp.

6.17.3.2. int csound::Chunk::chunkSize

Definition at line 57 of file Midifile.hpp.

6.17.3.3. int csound::Chunk::chunkSizePosition

Definition at line 58 of file Midifile.hpp.

6.17.3.4. int csound::Chunk::chunkStart

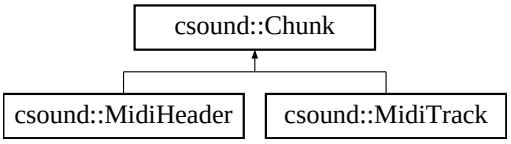
Definition at line 59 of file Midifile.hpp.

6.17.3.5. int csound::Chunk::id

Definition at line 56 of file Midifile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp


csound::Chunk
csound::MidiHeader
csound::MidiTrack

6.18. Loris::CkHeader Struct Reference

`#include <AiffData.h>`

Public Member Functions

- CkHeader (void)

Public Attributes

- ID id
- Uint_32 size

6.18.1. Constructor & Destructor Documentation

6.18.1.1. Loris::CkHeader::CkHeader (void) [inline]

Definition at line 103 of file AiffData.h.

References id, and size.

6.18.2. Member Data Documentation

6.18.2.1. ID Loris::CkHeader::id

Definition at line 97 of file AiffData.h.

Referenced by CkHeader().

6.18.2.2. Uint_32 Loris::CkHeader::size

Definition at line 98 of file AiffData.h.

Referenced by CkHeader().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.19. Loris::CommonCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- Int_16 channels
- Int_32 sampleFrames
- Int_16 bitsPerSample
- double srate

6.19.1. Member Data Documentation

6.19.1.1. Int_16 Loris::CommonCk::bitsPerSample

Definition at line 117 of file AiffData.h.

6.19.1.2. Int_16 Loris::CommonCk::channels

Definition at line 115 of file AiffData.h.

6.19.1.3. CkHeader Loris::CommonCk::header

Definition at line 114 of file AiffData.h.

6.19.1.4. Int_32 Loris::CommonCk::sampleFrames

Definition at line 116 of file AiffData.h.

6.19.1.5. double Loris::CommonCk::srate

Definition at line 118 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.20. Loris::PartialUtils::compareDurationGreater Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- bool operator() (const Partial &lhs, const Partial &rhs) const
- bool operator() (const Partial *lhs, const Partial *rhs) const

6.20.1. Member Function Documentation

6.20.1.1. bool Loris::PartialUtils::compareDurationGreater::operator() (const Partial * *lhs*, const Partial * *rhs*) const [inline]

Definition at line 523 of file PartialUtils.h.

6.20.1.2. bool Loris::PartialUtils::compareDurationGreater::operator() (const Partial & *lhs*, const Partial & *rhs*) const [inline]

Definition at line 520 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.21. Loris::PartialUtils::compareDurationLess Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- bool operator() (const Partial &lhs, const Partial &rhs) const
- bool operator() (const Partial *lhs, const Partial *rhs) const

6.21.1. Member Function Documentation

6.21.1.1. bool Loris::PartialUtils::compareDurationLess::operator() (const Partial * *lhs*, const Partial * *rhs*) const [inline]

Definition at line 508 of file PartialUtils.h.

6.21.1.2. bool Loris::PartialUtils::compareDurationLess::operator() (const Partial & *lhs*, const Partial & *rhs*) const [inline]

Definition at line 505 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.22. Loris::PartialUtils::compareLabelLess Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- bool operator() (const Partial &lhs, const Partial &rhs) const
- bool operator() (const Partial *lhs, const Partial *rhs) const

6.22.1. Member Function Documentation

6.22.1.1. bool Loris::PartialUtils::compareLabelLess::operator() (const Partial * *lhs*, const Partial * *rhs*) const [inline]

Definition at line 492 of file PartialUtils.h.

6.22.1.2. bool Loris::PartialUtils::compareLabelLess::operator() (const Partial & *lhs*, const Partial & *rhs*) const [inline]

Definition at line 489 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.23. csound::Composition Class Reference

`#include <Composition.hpp>`

Inheritance diagram for csound::Composition::

6.23.1. Detailed Description

Base class for user-derived musical compositions. Contains a Score object for collecting generated Events such as notes and control messages, and an Orchestra object for rendering the generated scores.

Definition at line 43 of file Composition.hpp.

Public Member Functions

- Composition ()
- virtual ~Composition ()
- virtual void generate ()
- virtual void createCsoundScore (std::string addToScore="")
- virtual void render ()
- virtual void perform ()
- virtual void clear ()
- virtual Score & getScore ()
- virtual void setCppSound (CppSound *orchestra)
- virtual CppSound * getCppSound ()
- virtual void write (const char *text)
- virtual void setTonesPerOctave (double tonesPerOctave)
- virtual double getTonesPerOctave () const
- virtual void setConformPitches (bool conformPitches)
- virtual bool getConformPitches () const

Protected Attributes

- Score score
- double tonesPerOctave
- bool conformPitches
- CppSound cppSound_
- CppSound * cppSound

6.23.2. Constructor & Destructor Documentation

6.23.2.1. csound::Composition::Composition ()

6.23.2.2. virtual csound::Composition::~Composition () [virtual]

6.23.3. Member Function Documentation

6.23.3.1. virtual void csound::Composition::clear () [virtual]

Clear all contents of this. Probably should be overridden in derived classes.

Reimplemented in csound::MusicModel.

6.23.3.2. virtual void csound::Composition::createCsoundScore (std::string *addToScore* = "") [virtual]

Translate the generated score to a Csound score and export it for performance.

6.23.3.3. virtual void csound::Composition::generate () [virtual]

Generate performance events and store them in the score. Must be overidden in derived classes.

Reimplemented in csound::MusicModel.

6.23.3.4. virtual bool csound::Composition::getConformPitches () const [virtual]

6.23.3.5. virtual CppSound* csound::Composition::getCppSound () [virtual]

Return the self-contained Orchestra.

6.23.3.6. virtual Score& csound::Composition::getScore () [virtual]

Return the self-contained Score.

6.23.3.7. virtual double csound::Composition::getTonesPerOctave () const [virtual]

6.23.3.8. virtual void csound::Composition::perform () [virtual]

Uses csound to perform the current score.

6.23.3.9. virtual void csound::Composition::render () [virtual]

Convenience function that erases the existing score, appends optional text to it, invokes generate(), invokes createCsoundScore(), and invokes perform().

6.23.3.10. virtual void csound::Composition::setConformPitches (bool *conformPitches*) [virtual]

6.23.3.11. virtual void csound::Composition::setCppSound (CppSound * *orchestra*) [virtual]

Sets the self-contained Orchestra.

6.23.3.12. virtual void csound::Composition::setTonesPerOctave (double *tonesPerOctave*) [virtual]

6.23.3.13. virtual void csound::Composition::write (const char * *text*) [virtual]

Write as if to stdout or stderr.

6.23.4. Member Data Documentation

6.23.4.1. bool csound::Composition::conformPitches [protected]

Definition at line 48 of file Composition.hpp.

6.23.4.2. CppSound* csound::Composition::cppSound [protected]

Definition at line 50 of file Composition.hpp.

6.23.4.3. CppSound csound::Composition::cppSound_ [protected]

Definition at line 49 of file Composition.hpp.

6.23.4.4. Score csound::Composition::score [protected]

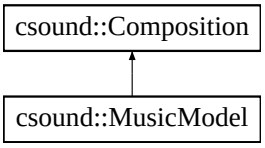
Definition at line 46 of file Composition.hpp.

6.23.4.5. double csound::Composition::tonesPerOctave [protected]

Definition at line 47 of file Composition.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Composition.hpp


csound::Composition
csound::MusicModel

6.24. Loris::ContainerCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- ID formType

6.24.1. Member Data Documentation

6.24.1.1. ID Loris::ContainerCk::formType

Definition at line 109 of file AiffData.h.

6.24.1.2. CkHeader Loris::ContainerCk::header

Definition at line 108 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.25. csound::Conversions Class Reference

`#include <Conversions.hpp>`

6.25.1. Detailed Description

Conversions to and from various music and signal processing units. Note that: silence::Event represents loudness in MIDI units (0 to 127). silence::Orchestra represents loudness in gain (0 to 1). silence::WaveSoundfileOut represents loudness in amplitude (0 to 1 for float samples, 0 to 32767 for short samples). Loudness can also be represented in positive decibels (0 to 84 for short samples, 0 to whatever for float samples). For float samples, decibels are assumed to be equivalent to MIDI velocity; otherwise, MIDI velocity is rescaled according to the maximum dynamic range supported by the sample size. All loudness conversions are driven by sample word size, which must be set before use; the default is 4 (float samples).

Definition at line 55 of file Conversions.hpp.

Static Public Member Functions

- static double getPI (void)
- static double get2PI (void)
- static double getMiddleCHz (void)
- static double getNORM_7 (void)
- static bool initialize ()
- static int getSampleSize (void)
- static double getMaximumAmplitude (int size)
- static double getMaximumDynamicRange ()
- static double amplitudeToDecibels (double amplitude)
- static double amplitudeToGain (double Amplitude)
- static double decibelsToAmplitude (double decibels)
- static double decibelsToMidi (double decibels)
- static double gainToAmplitude (double Gain)
- static double midiToDecibels (double Midi)
- static double midiToAmplitude (double Midi)
- static double amplitudeToMidi (double Amplitude)
- static double midiToGain (double Midi)
- static double leftPan (double x)
- static const double round (double value)
- static double temper (double octave, double tonesPerOctave)
- static double phaseToTableLengths (double Phase, double TableSampleCount)
- static double hzToMidi (double Hz, bool rounded)
- static double hzToOctave (double Hz)
- static double hzToSamplingIncrement (double Hz, double SR, double SamplesPerCycle)
- static double midiToHz (double Midi)
- static double midiToOctave (double Midi)
- static double midiToSamplingIncrement (double Midi, double SR, double SamplesPerCycle)
- static double octaveToHz (double Octave)
- static double octaveToMidi (double Octave, bool rounded)
- static double octaveToSamplingIncrement (double Octave, double SR, double SamplesPerCycle)
- static double rightPan (double x)
- static int swapInt (int Source)

- static short swapShort (short Source)
- static bool stringToBool (std::string value)
- static std::string boolToString (bool value)
- static int stringToInt (std::string value)
- static std::string intToString (int value)
- static double stringToDouble (std::string value)
- static std::string doubleToString (double value)
- static double midiToPitchClass (double midiKey)
- static double pitchClassSetToMidi (double pitchClassSet)
- static double midiToPitchClassSet (double midiKey)
- static double pitchClassToMidi (double pitchClass)
- static double findClosestPitchClass (double pitchClassSet, double pitchClass, double tones=12.0)
- static double midiToRoundedOctave (double midiKey)
- static std::string & trim (std::string &value)
- static std::string & trimQuotes (std::string &value)
- static double modulus (double a, double b)
- static double nameToPitchClassSet (std::string name)
- static std::string pitchClassSetToName (double pitchClassSet)

Static Public Attributes

- static const double PI_
- static const double TWO_PI_
- static const double middleCHz
- static const double log10d20
- static const double log10scale
- static const double NORM_7_
- static const double floatMaximumAmplitude
- static int sampleSize

Static Private Member Functions

- static void subfill (std::string root, char *cname, double pcs)
- static void fill (char *cname, char *cpitches)
- static std::string listPitchClassSets ()

Static Private Attributes

- static bool initialized_
- static std::map< std::string, double > pitchClassSetsForNames
- static std::map< double, std::string > namesForPitchClassSets

6.25.2. Member Function Documentation

6.25.2.1. static double csound::Conversions::amplitudeToDecibels (double *amplitude*) [static]

6.25.2.2. static double csound::Conversions::amplitudeToGain (double *Amplitude*) [static]

6.25.2.3. static double csound::Conversions::amplitudeToMidi (double *Amplitude*) [static]

6.25.2.4. static std::string csound::Conversions::boolToString (bool *value*) [static]

6.25.2.5. static double csound::Conversions::decibelsToAmplitude (double *decibels*) [static]

6.25.2.6. static double csound::Conversions::decibelsToMidi (double *decibels*) [static]

6.25.2.7. static std::string csound::Conversions::doubleToString (double *value*) [static]

6.25.2.8. static void csound::Conversions::fill (char * *cname*, char * *cpitches*) [static, private]

6.25.2.9. static double csound::Conversions::findClosestPitchClass (double *pitchClassSet*, double *pitchClass*, double *tones* = 12.0) [static]

6.25.2.10. static double csound::Conversions::gainToAmplitude (double *Gain*) [static]

6.25.2.11. static double csound::Conversions::get2PI (void) [static]

6.25.2.12. static double csound::Conversions::getMaximumAmplitude (int *size*) [static]

Returns the maximum soundfile amplitude for the sample size, assuming either float or twos' complement integer samples.

6.25.2.13. static double csound::Conversions::getMaximumDynamicRange () [static]

6.25.2.14. static double csound::Conversions::getMiddleCHz (void) [static]

6.25.2.15. static double csound::Conversions::getNORM_7 (void) [static]

6.25.2.16. static double csound::Conversions::getPI (void) [static]

6.25.2.17. static int csound::Conversions::getSampleSize (void) [static]

Returns the maximum soundfile amplitude for the sample size.

6.25.2.18. static double csound::Conversions::hzToMidi (double *Hz*, bool *rounded*) [static]

6.25.2.19. static double csound::Conversions::hzToOctave (double *Hz*) [static]

6.25.2.20. static double csound::Conversions::hzToSamplingIncrement (double *Hz*, double *SR*, double *SamplesPerCycle*) [static]

6.25.2.21. static bool csound::Conversions::initialize () [static]

6.25.2.22. static std::string csound::Conversions::intToString (int *value*) [static]

6.25.2.23. static double csound::Conversions::leftPan (double *x*) [static]

6.25.2.24. static std::string csound::Conversions::listPitchClassSets () [static, private]

6.25.2.25. static double csound::Conversions::midiToAmplitude (double *Midi*) [static]

6.25.2.26. static double csound::Conversions::midiToDecibels (double *Midi*) [static]

6.25.2.27. static double csound::Conversions::midiToGain (double *Midi*) [static]

6.25.2.28. static double csound::Conversions::midiToHz (double *Midi*) [static]

6.25.2.29. static double csound::Conversions::midiToOctave (double *Midi*) [static]

6.25.2.30. static double csound::Conversions::midiToPitchClass (double *midiKey*) [static]

6.25.2.31. static double csound::Conversions::midiToPitchClassSet (double *midiKey*) [static]

6.25.2.32. static double csound::Conversions::midiToRoundedOctave (double *midiKey*) [static]

6.25.2.33. static double csound::Conversions::midiToSamplingIncrement (double *Midi*, double *SR*, double *SamplesPerCycle*) [static]

6.25.2.34. static double csound::Conversions::modulus (double *a*, double *b*) [static]

True modulus accounting for sign.

6.25.2.35. static double csound::Conversions::nameToPitchClassSet (std::string *name*) [static]

Return the pitch-class set number (sum of powers of 2 by pitch-class) for the jazz-style scale or chord name.

6.25.2.36. static double csound::Conversions::octaveToHz (double *Octave*) [static]

6.25.2.37. static double csound::Conversions::octaveToMidi (double *Octave*, bool *rounded*) [static]

6.25.2.38. static double csound::Conversions::octaveToSamplingIncrement (double *Octave*, double *SR*, double *SamplesPerCycle*) [static]

6.25.2.39. static double csound::Conversions::phaseToTableLengths (double *Phase*, double *TableSampleCount*) [static]

6.25.2.40. static double csound::Conversions::pitchClassSetToMidi (double *pitchClassSet*) [static]

6.25.2.41. static std::string csound::Conversions::pitchClassSetToName (double *pitchClassSet*) [static]

Return the jazz-style scale or chord name for the pitch-class set number (sum of powers of 2 by pitch-class).

6.25.2.42. static double csound::Conversions::pitchClassToMidi (double *pitchClass*) [static]

6.25.2.43. static double csound::Conversions::rightPan (double *x*) [static]

6.25.2.44. static const double csound::Conversions::round (double *value*) [static]

6.25.2.45. static bool csound::Conversions::stringToBool (std::string *value*) [static]

6.25.2.46. static double csound::Conversions::stringToDouble (std::string *value*) [static]

6.25.2.47. static int csound::Conversions::stringToInt (std::string *value*) [static]

6.25.2.48. static void csound::Conversions::subfill (std::string *root*, char * *cname*, double *pcs*) [static, private]

6.25.2.49. static int csound::Conversions::swapInt (int *Source*) [static]

6.25.2.50. static short csound::Conversions::swapShort (short *Source*) [static]

6.25.2.51. static double csound::Conversions::temper (double *octave*, double *tonesPerOctave*) [static]

6.25.2.52. static std::string& csound::Conversions::trim (std::string & *value*) [static]

6.25.2.53. static std::string& csound::Conversions::trimQuotes (std::string & *value*) [static]

6.25.3. Member Data Documentation

6.25.3.1. const double csound::Conversions::floatMaximumAmplitude [static]

Definition at line 71 of file Conversions.hpp.

6.25.3.2. bool csound::Conversions::initialized_ [static, private]

Definition at line 57 of file Conversions.hpp.

6.25.3.3. const double csound::Conversions::log10d20 [static]

Definition at line 68 of file Conversions.hpp.

6.25.3.4. const double csound::Conversions::log10scale [static]

Definition at line 69 of file Conversions.hpp.

6.25.3.5. const double csound::Conversions::middleCHz [static]

Definition at line 67 of file Conversions.hpp.

6.25.3.6. std::map<double, std::string> csound::Conversions::namesForPitchClassSets [static, private]

Definition at line 59 of file Conversions.hpp.

6.25.3.7. const double csound::Conversions::NORM_7_ [static]

Definition at line 70 of file Conversions.hpp.

6.25.3.8. const double csound::Conversions::PI_ [static]

Definition at line 65 of file Conversions.hpp.

6.25.3.9. std::map<std::string, double> csound::Conversions::pitchClassSetsForNames [static, private]

Definition at line 58 of file Conversions.hpp.

6.25.3.10. int csound::Conversions::sampleSize [static]

Definition at line 72 of file Conversions.hpp.

6.25.3.11. const double csound::Conversions::TWO_PI_ [static]

Definition at line 66 of file Conversions.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Conversions.hpp

6.26. Counterpoint Class Reference

`#include <Counterpoint.hpp>`

Inheritance diagram for Counterpoint::

Public Types

- enum { MostNotes_ = 128, MostVoices_ = 12 }
- enum {
  Unison = 0, MinorSecond = 1, MajorSecond = 2, MinorThird = 3,
  MajorThird = 4, Fourth = 5, Tritone = 6, Fifth = 7,
  MinorSixth = 8, MajorSixth = 9, MinorSeventh = 10, MajorSeventh = 11,
  Octave = 12 }
- enum {
  Aeolian = 1, Dorian = 2, Phrygian = 3, Lydian = 4,
  Mixolydian = 5, Ionian = 6, Locrian = 7 }
- enum { DirectMotion = 1, ContraryMotion = 2, ObliqueMotion = 3, NoMotion = 4 }
- enum {
  WholeNote = 8, HalfNote = 4, DottedHalfNote = 6, QuarterNote = 2,
  DottedQuarterNote = 3, EighthNote = 1 }
- enum {
  One = 0, Two = 2, Three = 3, Four = 4,
  Five = 5, Six = 6, Eight = 8 }
- enum { infinity = 1000000, Bad = 100, RealBad = 200 }
- enum { INTERVALS_WITH_BASS_SIZE = 8 }
- enum { NumFields = 16, Field = (MostVoices_+1), EndF = (Field∗NumFields) }

Public Member Functions

- void message (const char ∗format,...)
- void message (const char ∗format, va_list valist)
- virtual void initialize (int mostnotes, int mostvoices)
- virtual void clear ()
- Counterpoint ()
- virtual ∼Counterpoint ()
- int ABS (int i)
- int MIN (int a, int b)
- int MAX (int a, int b)
- void ARRBLT (int ∗dest, int ∗source, int num)
- int InMode (int Pitch, int Mode)
- int BadMelody (int Intv)
- int ASkip (int Interval)
- int AStep (int Interval)
- int AThird (int Interval)
- int ASeventh (int Interval)
- int AnOctave (int Interval)
- int ATenth (int Interval)
- int MotionType (int Pitch1, int Pitch2, int Pitch3, int Pitch4)

- int DirectMotionToPerfectConsonance (int Pitch1, int Pitch2, int Pitch3, int Pitch4)
- int ConsecutiveSkipsInSameDirection (int Pitch1, int Pitch2, int Pitch3)
- int OutOfRange (int Pitch)
- int ExtremeRange (int Pitch)
- int Us (int n, int v)
- int LastNote (int n, int v)
- int FirstNote (int n, int v)
- int NextToLastNote (int n, int v)
- void SetUs (int n, int p, int v)
- int TotalRange (int Cn, int Cp, int v)
- int Cantus (int n, int v)
- int VIndex (int Time, int VNum)
- int Other (int Cn, int v, int v1)
- int Bass (int Cn, int v)
- int Beat8 (int n)
- int DownBeat (int n, int v)
- int UpBeat (int n, int v)
- int PitchRepeats (int Cn, int Cp, int v)
- int Size (int MelInt)
- int TooMuchOfInterval (int Cn, int Cp, int v)
- int ADissonance (int Interval, int Cn, int Cp, int v, int Species)
- int Doubled (int Pitch, int Cn, int v)
- int SpecialSpeciesCheck (int Cn, int Cp, int v, int Other0, int Other1, int Other2, int NumParts, int Species, int MelInt, int Interval, int ActInt, int LastIntClass, int Pitch, int LastMelInt, int CurLim)
- void AddInterval (int n)
- int OtherVoiceCheck (int Cn, int Cp, int v, int NumParts, int Species, int CurLim)
- int Check (int Cn, int Cp, int v, int NumParts, int Species, int CurLim)
- int SaveIndx (int indx, int *Sp)
- void SaveResults (int CurrentPenalty, int Penalty, int v1, int Species)
- int Look (int CurPen, int CurVoice, int NumParts, int Species, int Lim, int *Pens, int *Is, int *CurNotes)
- void BestFitFirst (int CurTime, int CurrentPenalty, int NumParts, int Species, int BrLim)
- void FillRhyPat ()
- float RANDOM (float amp)
- void UsedRhy (int n)
- int CurRhy (int n)
- void CleanRhy ()
- int GoodRhy ()
- void counterpoint (int OurMode, int *StartPitches, int CurV, int CantusFirmusLength, int Species, int *cantus)
- void AnySpecies (int OurMode, int *StartPitches, int CurV, int CantusFirmusLength, int Species)
- void fillCantus (int c0, int c1, int c2, int c3, int c4, int c5, int c6, int c7, int c8, int c9, int c10, int c11, int c12, int c13, int c14)
- void toCsoundScore (std::string filename, double secondsPerPulse)
- void winners (int v1, int *data, int *best, int *best1, int *best2, int *durs)

Public Attributes

- boost::variate_generator< boost::mt19937, boost::uniform_real<> > * uniform_real_generator
- void(* messageCallback)(void *userdata, int attribute, const char *format, va_list valist)
- int MostNotes
- int MostVoices
- long randx
- boost::numeric::ublas::matrix< int > Ctrpt
- boost::numeric::ublas::matrix< int > Onset
- boost::numeric::ublas::matrix< int > Dur
- boost::numeric::ublas::vector< int > TotalNotes
- boost::numeric::ublas::matrix< int > BestFit
- boost::numeric::ublas::matrix< int > BestFit1
- boost::numeric::ublas::matrix< int > BestFit2
- boost::numeric::ublas::vector< int > vbs
- boost::numeric::ublas::matrix< int > RhyPat
- boost::numeric::ublas::vector< int > RhyNotes
- int Fits [3]
- int LowestSemitone
- int HighestSemitone
- int BasePitch
- int Mode
- int TotalTime
- int UnisonPenalty
- int DirectToFifthPenalty
- int DirectToOctavePenalty
- int ParallelFifthPenalty
- int ParallelUnisonPenalty
- int EndOnPerfectPenalty
- int NoLeadingTonePenalty
- int DissonancePenalty
- int OutOfRangePenalty
- int OutOfModePenalty
- int TwoSkipsPenalty
- int DirectMotionPenalty
- int PerfectConsonancePenalty
- int CompoundPenalty
- int TenthToOctavePenalty
- int SkipTo8vePenalty
- int SkipFromUnisonPenalty
- int SkipPrecededBySameDirectionPenalty
- int FifthPrecededBySameDirectionPenalty
- int SixthPrecededBySameDirectionPenalty
- int SkipFollowedBySameDirectionPenalty
- int FifthFollowedBySameDirectionPenalty
- int SixthFollowedBySameDirectionPenalty
- int TwoSkipsNotInTriadPenalty
- int BadMelodyPenalty
- int ExtremeRangePenalty
- int LydianCadentialTritonePenalty
- int LowerNeighborPenalty

- int UpperNeighborPenalty
- int OverTwelfthPenalty
- int OverOctavePenalty
- int SixthLeapPenalty
- int OctaveLeapPenalty
- int BadCadencePenalty
- int DirectPerfectOnDownbeatPenalty
- int RepetitionOnUpbeatPenalty
- int DissonanceNotFillingThirdPenalty
- int UnisonDownbeatPenalty
- int TwoRepeatedNotesPenalty
- int ThreeRepeatedNotesPenalty
- int FourRepeatedNotesPenalty
- int LeapAtCadencePenalty
- int NotaCambiataPenalty
- int NotBestCadencePenalty
- int UnisonOnBeat4Penalty
- int NotaLigaturePenalty
- int LesserLigaturePenalty
- int UnresolvedLigaturePenalty
- int NoTimeForaLigaturePenalty
- int EighthJumpPenalty
- int HalfUntiedPenalty
- int UnisonUpbeatPenalty
- int MelodicBoredomPenalty
- int SkipToDownBeatPenalty
- int ThreeSkipsPenalty
- int DownBeatUnisonPenalty
- int VerticalTritonePenalty
- int MelodicTritonePenalty
- int AscendingSixthPenalty
- int RepeatedPitchPenalty
- int NotContraryToOthersPenalty
- int NotTriadPenalty
- int InnerVoicesInDirectToPerfectPenalty
- int InnerVoicesInDirectToTritonePenalty
- int SixFiveChordPenalty
- int UnpreparedSixFivePenalty
- int UnresolvedSixFivePenalty
- int AugmentedIntervalPenalty
- int ThirdDoubledPenalty
- int DoubledLeadingTonePenalty
- int DoubledSixthPenalty
- int DoubledFifthPenalty
- int TripledBassPenalty
- int UpperVoicesTooFarApartPenalty
- int UnresolvedLeadingTonePenalty
- int AllVoicesSkipPenalty
- int DirectToTritonePenalty
- int CrossBelowBassPenalty
- int CrossAboveCantusPenalty
- int NoMotionAgainstOctavePenalty

- int IntervalsWithBass [INTERVALS_WITH_BASS_SIZE]
- int BestFitPenalty
- int MaxPenalty
- int Branches
- int AllDone
- float PenaltyRatio

Static Public Attributes

- static int PerfectConsonance [13]
- static int ImperfectConsonance [13]
- static int Dissonance [13]
- static int _Ionian [12]
- static int _Dorian [12]
- static int _Phrygian [12]
- static int _Lydian [12]
- static int _Mixolydian [12]
- static int _Aeolian [12]
- static int _Locrian [12]
- static int BadMelodyInterval [13]
- static int Indx [17]
- static boost::mt19937 mersenneTwister

6.26.1. Member Enumeration Documentation

6.26.1.1. anonymous enum

Enumeration values:

MostNotes_

MostVoices_

Definition at line 100 of file Counterpoint.hpp.

6.26.1.2. anonymous enum

Enumeration values:

Unison

MinorSecond

MajorSecond

MinorThird

MajorThird

Fourth

Tritone

Fifth

MinorSixth

MajorSixth

MinorSeventh

MajorSeventh

Octave

Definition at line 215 of file Counterpoint.hpp.

6.26.1.3. anonymous enum

Enumeration values:

Aeolian

Dorian

Phrygian

Lydian

Mixolydian

Ionian

Locrian

Definition at line 235 of file Counterpoint.hpp.

6.26.1.4. anonymous enum

Enumeration values:

DirectMotion

ContraryMotion

ObliqueMotion

NoMotion

Definition at line 283 of file Counterpoint.hpp.

6.26.1.5. anonymous enum

Enumeration values:

WholeNote

HalfNote

DottedHalfNote

QuarterNote

DottedQuarterNote

EighthNote

Definition at line 366 of file Counterpoint.hpp.

6.26.1.6. anonymous enum

Enumeration values:

One

Two

Three

Four

Five

Six

Eight

Definition at line 388 of file Counterpoint.hpp.

6.26.1.7. anonymous enum

Enumeration values:

infinity

Bad

RealBad

Definition at line 503 of file Counterpoint.hpp.

6.26.1.8. anonymous enum

Enumeration values:

INTERVALS_WITH_BASS_SIZE

Definition at line 754 of file Counterpoint.hpp.

6.26.1.9. anonymous enum

Enumeration values:

NumFields

Field

EndF

Definition at line 1207 of file Counterpoint.hpp.

6.26.2. Constructor & Destructor Documentation

6.26.2.1. Counterpoint::Counterpoint () [inline]

Definition at line 121 of file Counterpoint.hpp.

References AllVoicesSkipPenalty, AscendingSixthPenalty, AugmentedIntervalPenalty, Bad, BadCadencePenalty, BadMelodyPenalty, CompoundPenalty, CrossAboveCantusPenalty, CrossBelowBassPenalty, DirectMotionPenalty, DirectPerfectOnDownbeatPenalty, DirectToFifthPenalty, DirectToOctavePenalty, DirectToTritonePenalty, DissonanceNotFillingThirdPenalty, DissonancePenalty, DoubledFifthPenalty, DoubledLeadingTonePenalty, DoubledSixthPenalty, DownBeatUnisonPenalty, EighthJumpPenalty, EndOnPerfectPenalty, ExtremeRangePenalty, FifthFollowedBySameDirectionPenalty, FifthPrecededBySameDirectionPenalty, FourRepeatedNotesPenalty, HalfUntiedPenalty, HighestSemitone, infinity, initialize(), InnerVoicesInDirectToPerfectPenalty, InnerVoicesInDirectToTritonePenalty, LeapAtCadencePenalty, LesserLigaturePenalty, LowerNeighborPenalty, LowestSemitone, LydianCadentialTritonePenalty, MelodicBoredomPenalty, MelodicTritonePenalty, mersenneTwister, messageCallback, MostNotes_, MostVoices_, NoLeadingTonePenalty, NoMotionAgainstOctavePenalty, NotaCambiataPenalty, NotaLigaturePenalty, NotBestCadencePenalty, NotContraryToOthersPenalty, NoTimeForaLigaturePenalty, NotTriadPenalty, OctaveLeapPenalty, OutOfModePenalty, OutOfRangePenalty, OverOctavePenalty, OverTwelfthPenalty, ParallelFifthPenalty, ParallelUnisonPenalty, PerfectConsonancePenalty, RealBad, RepeatedPitchPenalty, RepetitionOnUpbeatPenalty, SixFiveChordPenalty, SixthFollowedBySameDirectionPenalty, SixthLeapPenalty, SixthPrecededBySameDirectionPenalty, SkipFollowedBySameDirectionPenalty, SkipFromUnisonPenalty, SkipPrecededBySameDirectionPenalty, SkipTo8vePenalty, SkipToDownBeatPenalty, TenthToOctavePenalty, ThirdDoubledPenalty, ThreeRepeatedNotesPenalty, ThreeSkipsPenalty, TripledBassPenalty, TwoRepeatedNotesPenalty, TwoSkipsNotInTriadPenalty, TwoSkipsPenalty, uniform_real_generator, UnisonDownbeatPenalty, UnisonOnBeat4Penalty, UnisonPenalty, UnisonUpbeatPenalty, UnpreparedSixFivePenalty, UnresolvedLeadingTonePenalty, UnresolvedLigaturePenalty,

UnresolvedSixFivePenalty, UpperNeighborPenalty, UpperVoicesTooFarApartPenalty, and VerticalTritonePenalty.

6.26.2.2. virtual Counterpoint::~Counterpoint () [inline, virtual]

Definition at line 207 of file Counterpoint.hpp.

References uniform_real_generator.

6.26.3. Member Function Documentation

6.26.3.1. int Counterpoint::ABS (int *i*) [inline]

Definition at line 211 of file Counterpoint.hpp.

Referenced by AnOctave(), ASkip(), AStep(), ATenth(), BadMelody(), DirectMotionToPerfectConsonance(), and Size().

6.26.3.2. void Counterpoint::AddInterval (int *n*) [inline]

Definition at line 762 of file Counterpoint.hpp.

6.26.3.3. int Counterpoint::ADissonance (int *Interval*, int *Cn*, int *Cp*, int *v*, int *Species*) [inline]

Definition at line 433 of file Counterpoint.hpp.

6.26.3.4. int Counterpoint::AnOctave (int *Interval*) [inline]

Definition at line 280 of file Counterpoint.hpp.

References ABS(), and Unison.

6.26.3.5. void Counterpoint::AnySpecies (int *OurMode*, int * *StartPitches*, int *CurV*, int *CantusFirmusLength*, int *Species*) [inline]

Definition at line 1471 of file Counterpoint.hpp.

6.26.3.6. void Counterpoint::ARRBLT (int * *dest*, int * *source*, int *num*) [inline]

Definition at line 214 of file Counterpoint.hpp.

6.26.3.7. int Counterpoint::ASeventh (int *Interval*) [inline]

Definition at line 279 of file Counterpoint.hpp.

References MajorSeventh, and MinorSeventh.

6.26.3.8. int Counterpoint::ASkip (int *Interval*) [inline]

Definition at line 275 of file Counterpoint.hpp.

References ABS(), and MajorSecond.

Referenced by ConsecutiveSkipsInSameDirection().

6.26.3.9. int Counterpoint::AStep (int *Interval*) [inline]

Definition at line 276 of file Counterpoint.hpp.

References ABS(), MajorSecond, and MinorSecond.

6.26.3.10. int Counterpoint::ATenth (int *Interval*) [inline]

Definition at line 281 of file Counterpoint.hpp.

References ABS(), and AThird().

6.26.3.11. int Counterpoint::AThird (int *Interval*) [inline]

Definition at line 278 of file Counterpoint.hpp.

References MajorThird, and MinorThird.

Referenced by ATenth().

6.26.3.12. int Counterpoint::BadMelody (int *Intv*) [inline]

Definition at line 273 of file Counterpoint.hpp.

References ABS(), BadMelodyInterval, MinorSixth, and Octave.

6.26.3.13. int Counterpoint::Bass (int *Cn*, int *v*) [inline]

Definition at line 358 of file Counterpoint.hpp.

References Cantus(), MIN(), and Other().

6.26.3.14. int Counterpoint::Beat8 (int *n*) [inline]

Definition at line 376 of file Counterpoint.hpp.

Referenced by DownBeat().

6.26.3.15. void Counterpoint::BestFitFirst (int *CurTime*, int *CurrentPenalty*, int *NumParts*, int *Species*, int *BrLim*) [inline]

Definition at line 1344 of file Counterpoint.hpp.

6.26.3.16. int Counterpoint::Cantus (int *n*, int *v*) [inline]

Definition at line 346 of file Counterpoint.hpp.

References Ctrpt, and Onset.

Referenced by Bass().

6.26.3.17. int Counterpoint::Check (int *Cn*, int *Cp*, int *v*, int *NumParts*, int *Species*, int *CurLim*) [inline]

Definition at line 902 of file Counterpoint.hpp.

6.26.3.18. void Counterpoint::CleanRhy () [inline]

Definition at line 1459 of file Counterpoint.hpp.

6.26.3.19. virtual void Counterpoint::clear () [virtual]

6.26.3.20. int Counterpoint::ConsecutiveSkipsInSameDirection (int *Pitch1*, int *Pitch2*, int *Pitch3*) [inline]

Definition at line 312 of file Counterpoint.hpp.

References ASkip().

6.26.3.21. void Counterpoint::counterpoint (int *OurMode*, int * *StartPitches*, int *CurV*, int *CantusFirmusLength*, int *Species*, int * *cantus*)

6.26.3.22. int Counterpoint::CurRhy (int *n*) [inline]

Definition at line 1458 of file Counterpoint.hpp.

6.26.3.23. int Counterpoint::DirectMotionToPerfectConsonance (int *Pitch1*, int *Pitch2*, int *Pitch3*, int *Pitch4*) [inline]

Definition at line 307 of file Counterpoint.hpp.

References ABS(), DirectMotion, MotionType(), and PerfectConsonance.

6.26.3.24. int Counterpoint::Doubled (int *Pitch*, int *Cn*, int *v*) [inline]

Definition at line 493 of file Counterpoint.hpp.

6.26.3.25. int Counterpoint::DownBeat (int *n*, int *v*) [inline]

Definition at line 377 of file Counterpoint.hpp.

References Beat8(), and Onset.

Referenced by UpBeat().

6.26.3.26. int Counterpoint::ExtremeRange (int *Pitch*) [inline]

Definition at line 322 of file Counterpoint.hpp.

References HighestSemitone, and LowestSemitone.

6.26.3.27. void Counterpoint::fillCantus (int *c0*, int *c1*, int *c2*, int *c3*, int *c4*, int *c5*, int *c6*, int *c7*, int *c8*, int *c9*, int *c10*, int *c11*, int *c12*, int *c13*, int *c14*) [inline]

Definition at line 1548 of file Counterpoint.hpp.

6.26.3.28. void Counterpoint::FillRhyPat () [inline]

Definition at line 1421 of file Counterpoint.hpp.

6.26.3.29. int Counterpoint::FirstNote (int *n*, int *v*) [inline]

Definition at line 328 of file Counterpoint.hpp.

6.26.3.30. int Counterpoint::GoodRhy () [inline]

Definition at line 1460 of file Counterpoint.hpp.

6.26.3.31. virtual void Counterpoint::initialize (int *mostnotes*, int *mostvoices*) [virtual]

Referenced by Counterpoint().

6.26.3.32. int Counterpoint::InMode (int *Pitch*, int *Mode*) [inline]

Definition at line 253 of file Counterpoint.hpp.

References _Aeolian, _Dorian, _Ionian, _Locrian, _Lydian, _Mixolydian, _Phrygian, Aeolian, Dorian, Ionian, Locrian, Lydian, Mixolydian, and Phrygian.

6.26.3.33. int Counterpoint::LastNote (int *n*, int *v*) [inline]

Definition at line 327 of file Counterpoint.hpp.

References TotalNotes.

6.26.3.34. int Counterpoint::Look (int *CurPen*, int *CurVoice*, int *NumParts*, int *Species*, int *Lim*, int * *Pens*, int * *Is*, int * *CurNotes*) [inline]

Definition at line 1305 of file Counterpoint.hpp.

6.26.3.35. int Counterpoint::MAX (int *a*, int *b*) [inline]

Definition at line 213 of file Counterpoint.hpp.

Referenced by TotalRange().

6.26.3.36. void Counterpoint::message (const char * *format*, va_list *valist*) [inline]

Definition at line 87 of file Counterpoint.hpp.

References messageCallback.

6.26.3.37. void Counterpoint::message (const char * *format*, ...) [inline]

Definition at line 80 of file Counterpoint.hpp.

6.26.3.38. int Counterpoint::MIN (int *a*, int *b*) [inline]

Definition at line 212 of file Counterpoint.hpp.

Referenced by Bass(), and TotalRange().

6.26.3.39. int Counterpoint::MotionType (int *Pitch1*, int *Pitch2*, int *Pitch3*, int *Pitch4*) [inline]

Definition at line 291 of file Counterpoint.hpp.

References ContraryMotion, DirectMotion, NoMotion, and ObliqueMotion.

Referenced by DirectMotionToPerfectConsonance().

6.26.3.40. int Counterpoint::NextToLastNote (int *n*, int *v*) [inline]

Definition at line 329 of file Counterpoint.hpp.

References TotalNotes.

6.26.3.41. int Counterpoint::Other (int *Cn*, int *v*, int *v1*) [inline]

Definition at line 356 of file Counterpoint.hpp.

References Ctrpt, Onset, and VIndex().

Referenced by Bass().

6.26.3.42. int Counterpoint::OtherVoiceCheck (int *Cn*, int *Cp*, int *v*, int *NumParts*, int *Species*, int *CurLim*) [inline]

Definition at line 778 of file Counterpoint.hpp.

6.26.3.43. int Counterpoint::OutOfRange (int *Pitch*) [inline]

Definition at line 321 of file Counterpoint.hpp.

References HighestSemitone, and LowestSemitone.

6.26.3.44. int Counterpoint::PitchRepeats (int *Cn*, int *Cp*, int *v*) [inline]

Definition at line 380 of file Counterpoint.hpp.

References Us().

6.26.3.45. float Counterpoint::RANDOM (float *amp*) [inline]

Definition at line 1452 of file Counterpoint.hpp.

6.26.3.46. int Counterpoint::SaveIndx (int *indx*, int * *Sp*) [inline]

Definition at line 1213 of file Counterpoint.hpp.

6.26.3.47. void Counterpoint::SaveResults (int *CurrentPenalty*, int *Penalty*, int *v1*, int *Species*) [inline]

Definition at line 1237 of file Counterpoint.hpp.

6.26.3.48. void Counterpoint::SetUs (int *n*, int *p*, int *v*) [inline]

Definition at line 330 of file Counterpoint.hpp.

References Ctrpt.

6.26.3.49. int Counterpoint::Size (int *MelInt*) [inline]

Definition at line 398 of file Counterpoint.hpp.

References ABS(), Eight, Fifth, Five, Four, Fourth, MajorSecond, MajorThird, MinorSecond, MinorSixth, MinorThird, Octave, One, Six, Three, Two, and Unison.

Referenced by TooMuchOfInterval().

6.26.3.50. int Counterpoint::SpecialSpeciesCheck (int *Cn*, int *Cp*, int *v*, int *Other0*, int *Other1*, int *Other2*, int *NumParts*, int *Species*, int *MelInt*, int *Interval*, int *ActInt*, int *LastIntClass*, int *Pitch*, int *LastMelInt*, int *CurLim*) [inline]

Definition at line 593 of file Counterpoint.hpp.

6.26.3.51. void Counterpoint::toCsoundScore (std::string *filename*, double *secondsPerPulse*)

6.26.3.52. int Counterpoint::TooMuchOfInterval (int *Cn*, int *Cp*, int *v*) [inline]

Definition at line 417 of file Counterpoint.hpp.

References Ctrpt, and Size().

6.26.3.53. int Counterpoint::TotalRange (int *Cn*, int *Cp*, int *v*) [inline]

Definition at line 332 of file Counterpoint.hpp.

References MAX(), MIN(), and Us().

6.26.3.54. int Counterpoint::UpBeat (int *n*, int *v*) [inline]

Definition at line 378 of file Counterpoint.hpp.

References DownBeat().

6.26.3.55. int Counterpoint::Us (int *n*, int *v*) [inline]

Definition at line 326 of file Counterpoint.hpp.

References Ctrpt.

Referenced by PitchRepeats(), and TotalRange().

6.26.3.56. void Counterpoint::UsedRhy (int *n*) [inline]

Definition at line 1457 of file Counterpoint.hpp.

6.26.3.57. int Counterpoint::VIndex (int *Time*, int *VNum*) [inline]

Definition at line 348 of file Counterpoint.hpp.

References Dur, Onset, and TotalNotes.

Referenced by Other().

6.26.3.58. void Counterpoint::winners (int *v1*, int * *data*, int * *best*, int * *best1*, int * *best2*, int * *durs*) [inline]

Definition at line 1557 of file Counterpoint.hpp.

6.26.4. Member Data Documentation

6.26.4.1. int Counterpoint::_Aeolian[12] [static]

Definition at line 250 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.2. int Counterpoint::_Dorian[12] [static]

Definition at line 246 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.3. int Counterpoint::_Ionian[12] [static]

Definition at line 245 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.4. int Counterpoint::_Locrian[12] [static]

Definition at line 251 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.5. int Counterpoint::_Lydian[12] [static]

Definition at line 248 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.6. int Counterpoint::_Mixolydian[12] [static]

Definition at line 249 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.7. int Counterpoint::_Phrygian[12] [static]

Definition at line 247 of file Counterpoint.hpp.

Referenced by InMode().

6.26.4.8. int Counterpoint::AllDone

Definition at line 1205 of file Counterpoint.hpp.

6.26.4.9. int Counterpoint::AllVoicesSkipPenalty

Definition at line 585 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.10. int Counterpoint::AscendingSixthPenalty

Definition at line 568 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.11. int Counterpoint::AugmentedIntervalPenalty

Definition at line 577 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.12. int Counterpoint::BadCadencePenalty

Definition at line 543 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.13. int Counterpoint::BadMelodyInterval[13] [static]

Definition at line 272 of file Counterpoint.hpp.

Referenced by BadMelody().

6.26.4.14. int Counterpoint::BadMelodyPenalty

Definition at line 534 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.15. int Counterpoint::BasePitch

Definition at line 324 of file Counterpoint.hpp.

6.26.4.16. boost::numeric::ublas::matrix<int> Counterpoint::BestFit

Definition at line 110 of file Counterpoint.hpp.

6.26.4.17. boost::numeric::ublas::matrix<int> Counterpoint::BestFit1

Definition at line 111 of file Counterpoint.hpp.

6.26.4.18. boost::numeric::ublas::matrix<int> Counterpoint::BestFit2

Definition at line 112 of file Counterpoint.hpp.

6.26.4.19. int Counterpoint::BestFitPenalty

Definition at line 1205 of file Counterpoint.hpp.

6.26.4.20. int Counterpoint::Branches

Definition at line 1205 of file Counterpoint.hpp.

6.26.4.21. int Counterpoint::CompoundPenalty

Definition at line 523 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.22. int Counterpoint::CrossAboveCantusPenalty

Definition at line 590 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.23. int Counterpoint::CrossBelowBassPenalty

Definition at line 587 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.24. boost::numeric::ublas::matrix<int> Counterpoint::Ctrpt

Definition at line 106 of file Counterpoint.hpp.
Referenced by Cantus(), Other(), SetUs(), TooMuchOfInterval(), and Us().

6.26.4.25. int Counterpoint::DirectMotionPenalty

Definition at line 521 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.26. int Counterpoint::DirectPerfectOnDownbeatPenalty

Definition at line 544 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.27. int Counterpoint::DirectToFifthPenalty

Definition at line 511 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.28. int Counterpoint::DirectToOctavePenalty

Definition at line 512 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.29. int Counterpoint::DirectToTritonePenalty

Definition at line 586 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.30. int Counterpoint::Dissonance[13] [static]

Definition at line 233 of file Counterpoint.hpp.

6.26.4.31. int Counterpoint::DissonanceNotFillingThirdPenalty

Definition at line 546 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.32. int Counterpoint::DissonancePenalty

Definition at line 517 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.33. int Counterpoint::DoubledFifthPenalty

Definition at line 581 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.34. int Counterpoint::DoubledLeadingTonePenalty

Definition at line 579 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.35. int Counterpoint::DoubledSixthPenalty

Definition at line 580 of file Counterpoint.hpp.
Referenced by Counterpoint().

6.26.4.36. int Counterpoint::DownBeatUnisonPenalty

Definition at line 565 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.37. boost::numeric::ublas::matrix<int> Counterpoint::Dur

Definition at line 108 of file Counterpoint.hpp.

Referenced by VIndex().

6.26.4.38. int Counterpoint::EighthJumpPenalty

Definition at line 559 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.39. int Counterpoint::EndOnPerfectPenalty

Definition at line 515 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.40. int Counterpoint::ExtremeRangePenalty

Definition at line 535 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.41. int Counterpoint::FifthFollowedBySameDirectionPenalty

Definition at line 531 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.42. int Counterpoint::FifthPrecededBySameDirectionPenalty

Definition at line 528 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.43. int Counterpoint::Fits[3]

Definition at line 116 of file Counterpoint.hpp.

6.26.4.44. int Counterpoint::FourRepeatedNotesPenalty

Definition at line 550 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.45. int Counterpoint::HalfUntiedPenalty

Definition at line 560 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.46. int Counterpoint::HighestSemitone

Definition at line 319 of file Counterpoint.hpp.

Referenced by Counterpoint(), ExtremeRange(), and OutOfRange().

6.26.4.47. int Counterpoint::ImperfectConsonance[13] [static]

Definition at line 232 of file Counterpoint.hpp.

6.26.4.48. int Counterpoint::Indx[17] [static]

Definition at line 1303 of file Counterpoint.hpp.

6.26.4.49. int Counterpoint::InnerVoicesInDirectToPerfectPenalty

Definition at line 572 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.50. int Counterpoint::InnerVoicesInDirectToTritonePenalty

Definition at line 573 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.51. int Counterpoint::IntervalsWithBass[INTERVALS_WITH_BASS_SIZE]

Definition at line 759 of file Counterpoint.hpp.

6.26.4.52. int Counterpoint::LeapAtCadencePenalty

Definition at line 551 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.53. int Counterpoint::LesserLigaturePenalty

Definition at line 556 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.54. int Counterpoint::LowerNeighborPenalty

Definition at line 537 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.55. int Counterpoint::LowestSemitone

Definition at line 318 of file Counterpoint.hpp.

Referenced by Counterpoint(), ExtremeRange(), and OutOfRange().

6.26.4.56. int Counterpoint::LydianCadentialTritonePenalty

Definition at line 536 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.57. int Counterpoint::MaxPenalty

Definition at line 1205 of file Counterpoint.hpp.

6.26.4.58. int Counterpoint::MelodicBoredomPenalty

Definition at line 562 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.59. int Counterpoint::MelodicTritonePenalty

Definition at line 567 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.60. boost::mt19937 Counterpoint::mersenneTwister [static]

Definition at line 1450 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.61. void(* Counterpoint::messageCallback)(void *userdata, int attribute, const char *format, va_list valist)

Referenced by Counterpoint(), and message().

6.26.4.62. int Counterpoint::Mode

Definition at line 324 of file Counterpoint.hpp.

6.26.4.63. int Counterpoint::MostNotes

Definition at line 98 of file Counterpoint.hpp.

6.26.4.64. int Counterpoint::MostVoices

Definition at line 99 of file Counterpoint.hpp.

6.26.4.65. int Counterpoint::NoLeadingTonePenalty

Definition at line 516 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.66. int Counterpoint::NoMotionAgainstOctavePenalty

Definition at line 591 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.67. int Counterpoint::NotaCambiataPenalty

Definition at line 552 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.68. int Counterpoint::NotaLigaturePenalty

Definition at line 555 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.69. int Counterpoint::NotBestCadencePenalty

Definition at line 553 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.70. int Counterpoint::NotContraryToOthersPenalty

Definition at line 570 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.71. int Counterpoint::NoTimeForaLigaturePenalty

Definition at line 558 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.72. int Counterpoint::NotTriadPenalty

Definition at line 571 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.73. int Counterpoint::OctaveLeapPenalty

Definition at line 542 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.74. boost::numeric::ublas::matrix<int> Counterpoint::Onset

Definition at line 107 of file Counterpoint.hpp.

Referenced by Cantus(), DownBeat(), Other(), and VIndex().

6.26.4.75. int Counterpoint::OutOfModePenalty

Definition at line 519 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.76. int Counterpoint::OutOfRangePenalty

Definition at line 518 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.77. int Counterpoint::OverOctavePenalty

Definition at line 540 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.78. int Counterpoint::OverTwelfthPenalty

Definition at line 539 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.79. int Counterpoint::ParallelFifthPenalty

Definition at line 513 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.80. int Counterpoint::ParallelUnisonPenalty

Definition at line 514 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.81. float Counterpoint::PenaltyRatio

Definition at line 1206 of file Counterpoint.hpp.

6.26.4.82. int Counterpoint::PerfectConsonance[13] [static]

Definition at line 231 of file Counterpoint.hpp.

Referenced by DirectMotionToPerfectConsonance().

6.26.4.83. int Counterpoint::PerfectConsonancePenalty

Definition at line 522 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.84. long Counterpoint::randx

Definition at line 105 of file Counterpoint.hpp.

6.26.4.85. int Counterpoint::RepeatedPitchPenalty

Definition at line 569 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.86. int Counterpoint::RepetitionOnUpbeatPenalty

Definition at line 545 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.87. boost::numeric::ublas::vector<int> Counterpoint::RhyNotes

Definition at line 115 of file Counterpoint.hpp.

6.26.4.88. boost::numeric::ublas::matrix<int> Counterpoint::RhyPat

Definition at line 114 of file Counterpoint.hpp.

6.26.4.89. int Counterpoint::SixFiveChordPenalty

Definition at line 574 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.90. int Counterpoint::SixthFollowedBySameDirectionPenalty

Definition at line 532 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.91. int Counterpoint::SixthLeapPenalty

Definition at line 541 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.92. int Counterpoint::SixthPrecededBySameDirectionPenalty

Definition at line 529 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.93. int Counterpoint::SkipFollowedBySameDirectionPenalty

Definition at line 530 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.94. int Counterpoint::SkipFromUnisonPenalty

Definition at line 526 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.95. int Counterpoint::SkipPrecededBySameDirectionPenalty

Definition at line 527 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.96. int Counterpoint::SkipTo8vePenalty

Definition at line 525 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.97. int Counterpoint::SkipToDownBeatPenalty

Definition at line 563 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.98. int Counterpoint::TenthToOctavePenalty

Definition at line 524 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.99. int Counterpoint::ThirdDoubledPenalty

Definition at line 578 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.100. int Counterpoint::ThreeRepeatedNotesPenalty

Definition at line 549 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.101. int Counterpoint::ThreeSkipsPenalty

Definition at line 564 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.102. boost::numeric::ublas::vector<int> Counterpoint::TotalNotes

Definition at line 109 of file Counterpoint.hpp.

Referenced by LastNote(), NextToLastNote(), and VIndex().

6.26.4.103. int Counterpoint::TotalTime

Definition at line 324 of file Counterpoint.hpp.

6.26.4.104. int Counterpoint::TripledBassPenalty

Definition at line 582 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.105. int Counterpoint::TwoRepeatedNotesPenalty

Definition at line 548 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.106. int Counterpoint::TwoSkipsNotInTriadPenalty

Definition at line 533 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.107. int Counterpoint::TwoSkipsPenalty

Definition at line 520 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.108. boost::variate_generator<boost::mt19937, boost::uniform_real<> >* Counterpoint::uniform_real_generator

Definition at line 78 of file Counterpoint.hpp.

Referenced by Counterpoint(), and ~Counterpoint().

6.26.4.109. int Counterpoint::UnisonDownbeatPenalty

Definition at line 547 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.110. int Counterpoint::UnisonOnBeat4Penalty

Definition at line 554 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.111. int Counterpoint::UnisonPenalty

Definition at line 510 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.112. int Counterpoint::UnisonUpbeatPenalty

Definition at line 561 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.113. int Counterpoint::UnpreparedSixFivePenalty

Definition at line 575 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.114. int Counterpoint::UnresolvedLeadingTonePenalty

Definition at line 584 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.115. int Counterpoint::UnresolvedLigaturePenalty

Definition at line 557 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.116. int Counterpoint::UnresolvedSixFivePenalty

Definition at line 576 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.117. int Counterpoint::UpperNeighborPenalty

Definition at line 538 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.118. int Counterpoint::UpperVoicesTooFarApartPenalty

Definition at line 583 of file Counterpoint.hpp.

Referenced by Counterpoint().

6.26.4.119. boost::numeric::ublas::vector<int> Counterpoint::vbs

Definition at line 113 of file Counterpoint.hpp.

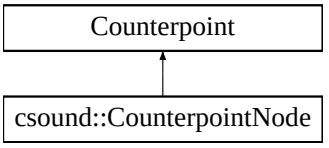
6.26.4.120. int Counterpoint::VerticalTritonePenalty

Definition at line 566 of file Counterpoint.hpp.

Referenced by Counterpoint().

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Counterpoint.hpp


Counterpoint
csound::CounterpointNode

6.27. csound::CounterpointNode Class Reference

`#include <CounterpointNode.hpp>`

Inheritance diagram for csound::CounterpointNode::

6.27.1. Detailed Description

Uses Bill Schottstaedt's species counterpoint generator code to either (a) generate a counterpoint in species 1, 2, or 3 for a cantus firmus selected from notes generated by child nodes, or (b) attempt to correct the voice leading for species 1, 2, or 3 counterpoint in notes generated by child nodes.

Definition at line 53 of file CounterpointNode.hpp.

Public Types

- enum { GenerateCounterpoint = 0, CorrectCounterpoint = 1 }
- enum { MostNotes_ = 128, MostVoices_ = 12 }
- enum {
  Unison = 0, MinorSecond = 1, MajorSecond = 2, MinorThird = 3,
  MajorThird = 4, Fourth = 5, Tritone = 6, Fifth = 7,
  MinorSixth = 8, MajorSixth = 9, MinorSeventh = 10, MajorSeventh = 11,
  Octave = 12 }
- enum {
  Aeolian = 1, Dorian = 2, Phrygian = 3, Lydian = 4,
  Mixolydian = 5, Ionian = 6, Locrian = 7 }
- enum { DirectMotion = 1, ContraryMotion = 2, ObliqueMotion = 3, NoMotion = 4 }
- enum {
  WholeNote = 8, HalfNote = 4, DottedHalfNote = 6, QuarterNote = 2,
  DottedQuarterNote = 3, EighthNote = 1 }
- enum {
  One = 0, Two = 2, Three = 3, Four = 4,
  Five = 5, Six = 6, Eight = 8 }
- enum { infinity = 1000000, Bad = 100, RealBad = 200 }
- enum { INTERVALS_WITH_BASS_SIZE = 8 }
- enum { NumFields = 16, Field = (MostVoices_+1), EndF = (Field*NumFields) }

Public Member Functions

- CounterpointNode ()
- virtual ~CounterpointNode ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &globalCoordinates)
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)

- virtual void addChild (Node *node)
- void message (const char *format,...)
- void message (const char *format, va_list valist)
- virtual void initialize (int mostnotes, int mostvoices)
- virtual void clear ()
- int ABS (int i)
- int MIN (int a, int b)
- int MAX (int a, int b)
- void ARRBLT (int *dest, int *source, int num)
- int InMode (int Pitch, int Mode)
- int BadMelody (int Intv)
- int ASkip (int Interval)
- int AStep (int Interval)
- int AThird (int Interval)
- int ASeventh (int Interval)
- int AnOctave (int Interval)
- int ATenth (int Interval)
- int MotionType (int Pitch1, int Pitch2, int Pitch3, int Pitch4)
- int DirectMotionToPerfectConsonance (int Pitch1, int Pitch2, int Pitch3, int Pitch4)
- int ConsecutiveSkipsInSameDirection (int Pitch1, int Pitch2, int Pitch3)
- int OutOfRange (int Pitch)
- int ExtremeRange (int Pitch)
- int Us (int n, int v)
- int LastNote (int n, int v)
- int FirstNote (int n, int v)
- int NextToLastNote (int n, int v)
- void SetUs (int n, int p, int v)
- int TotalRange (int Cn, int Cp, int v)
- int Cantus (int n, int v)
- int VIndex (int Time, int VNum)
- int Other (int Cn, int v, int v1)
- int Bass (int Cn, int v)
- int Beat8 (int n)
- int DownBeat (int n, int v)
- int UpBeat (int n, int v)
- int PitchRepeats (int Cn, int Cp, int v)
- int Size (int MelInt)
- int TooMuchOfInterval (int Cn, int Cp, int v)
- int ADissonance (int Interval, int Cn, int Cp, int v, int Species)
- int Doubled (int Pitch, int Cn, int v)
- int SpecialSpeciesCheck (int Cn, int Cp, int v, int Other0, int Other1, int Other2, int NumParts, int Species, int MelInt, int Interval, int ActInt, int LastIntClass, int Pitch, int LastMelInt, int CurLim)
- void AddInterval (int n)
- int OtherVoiceCheck (int Cn, int Cp, int v, int NumParts, int Species, int CurLim)
- int Check (int Cn, int Cp, int v, int NumParts, int Species, int CurLim)
- int SaveIndx (int indx, int *Sp)
- void SaveResults (int CurrentPenalty, int Penalty, int v1, int Species)
- int Look (int CurPen, int CurVoice, int NumParts, int Species, int Lim, int *Pens, int *Is, int *CurNotes)
- void BestFitFirst (int CurTime, int CurrentPenalty, int NumParts, int Species, int BrLim)
- void FillRhyPat ()

- float RANDOM (float amp)
- void UsedRhy (int n)
- int CurRhy (int n)
- void CleanRhy ()
- int GoodRhy ()
- void counterpoint (int OurMode, int *StartPitches, int CurV, int CantusFirmusLength, int Species, int *cantus)
- void AnySpecies (int OurMode, int *StartPitches, int CurV, int CantusFirmusLength, int Species)
- void fillCantus (int c0, int c1, int c2, int c3, int c4, int c5, int c6, int c7, int c8, int c9, int c10, int c11, int c12, int c13, int c14)
- void toCsoundScore (std::string filename, double secondsPerPulse)
- void winners (int v1, int *data, int *best, int *best1, int *best2, int *durs)

Public Attributes

- int generationMode
- int musicMode
- int species
- size_t voices
- double secondsPerPulse
- std::vector< int > voiceBeginnings
- std::vector< Node * > children
- boost::variate_generator< boost::mt19937, boost::uniform_real<> > * uniform_real_generator
- void(* messageCallback)(void *userdata, int attribute, const char *format, va_list valist)
- int MostNotes
- int MostVoices
- long randx
- boost::numeric::ublas::matrix< int > Ctrpt
- boost::numeric::ublas::matrix< int > Onset
- boost::numeric::ublas::matrix< int > Dur
- boost::numeric::ublas::vector< int > TotalNotes
- boost::numeric::ublas::matrix< int > BestFit
- boost::numeric::ublas::matrix< int > BestFit1
- boost::numeric::ublas::matrix< int > BestFit2
- boost::numeric::ublas::vector< int > vbs
- boost::numeric::ublas::matrix< int > RhyPat
- boost::numeric::ublas::vector< int > RhyNotes
- int Fits [3]
- int LowestSemitone
- int HighestSemitone
- int BasePitch
- int Mode
- int TotalTime
- int UnisonPenalty
- int DirectToFifthPenalty
- int DirectToOctavePenalty
- int ParallelFifthPenalty
- int ParallelUnisonPenalty
- int EndOnPerfectPenalty
- int NoLeadingTonePenalty

- int DissonancePenalty
- int OutOfRangePenalty
- int OutOfModePenalty
- int TwoSkipsPenalty
- int DirectMotionPenalty
- int PerfectConsonancePenalty
- int CompoundPenalty
- int TenthToOctavePenalty
- int SkipTo8vePenalty
- int SkipFromUnisonPenalty
- int SkipPrecededBySameDirectionPenalty
- int FifthPrecededBySameDirectionPenalty
- int SixthPrecededBySameDirectionPenalty
- int SkipFollowedBySameDirectionPenalty
- int FifthFollowedBySameDirectionPenalty
- int SixthFollowedBySameDirectionPenalty
- int TwoSkipsNotInTriadPenalty
- int BadMelodyPenalty
- int ExtremeRangePenalty
- int LydianCadentialTritonePenalty
- int LowerNeighborPenalty
- int UpperNeighborPenalty
- int OverTwelfthPenalty
- int OverOctavePenalty
- int SixthLeapPenalty
- int OctaveLeapPenalty
- int BadCadencePenalty
- int DirectPerfectOnDownbeatPenalty
- int RepetitionOnUpbeatPenalty
- int DissonanceNotFillingThirdPenalty
- int UnisonDownbeatPenalty
- int TwoRepeatedNotesPenalty
- int ThreeRepeatedNotesPenalty
- int FourRepeatedNotesPenalty
- int LeapAtCadencePenalty
- int NotaCambiataPenalty
- int NotBestCadencePenalty
- int UnisonOnBeat4Penalty
- int NotaLigaturePenalty
- int LesserLigaturePenalty
- int UnresolvedLigaturePenalty
- int NoTimeForaLigaturePenalty
- int EighthJumpPenalty
- int HalfUntiedPenalty
- int UnisonUpbeatPenalty
- int MelodicBoredomPenalty
- int SkipToDownBeatPenalty
- int ThreeSkipsPenalty
- int DownBeatUnisonPenalty
- int VerticalTritonePenalty
- int MelodicTritonePenalty
- int AscendingSixthPenalty

- int RepeatedPitchPenalty
- int NotContraryToOthersPenalty
- int NotTriadPenalty
- int InnerVoicesInDirectToPerfectPenalty
- int InnerVoicesInDirectToTritonePenalty
- int SixFiveChordPenalty
- int UnpreparedSixFivePenalty
- int UnresolvedSixFivePenalty
- int AugmentedIntervalPenalty
- int ThirdDoubledPenalty
- int DoubledLeadingTonePenalty
- int DoubledSixthPenalty
- int DoubledFifthPenalty
- int TripledBassPenalty
- int UpperVoicesTooFarApartPenalty
- int UnresolvedLeadingTonePenalty
- int AllVoicesSkipPenalty
- int DirectToTritonePenalty
- int CrossBelowBassPenalty
- int CrossAboveCantusPenalty
- int NoMotionAgainstOctavePenalty
- int IntervalsWithBass [INTERVALS_WITH_BASS_SIZE]
- int BestFitPenalty
- int MaxPenalty
- int Branches
- int AllDone
- float PenaltyRatio

Static Public Attributes

- static int PerfectConsonance [13]
- static int ImperfectConsonance [13]
- static int Dissonance [13]
- static int _Ionian [12]
- static int _Dorian [12]
- static int _Phrygian [12]
- static int _Lydian [12]
- static int _Mixolydian [12]
- static int _Aeolian [12]
- static int _Locrian [12]
- static int BadMelodyInterval [13]
- static int Indx [17]
- static boost::mt19937 mersenneTwister

Protected Attributes

- ublas::matrix< double > localCoordinates

6.27.2. Member Enumeration Documentation

6.27.2.1. anonymous enum [inherited]

Enumeration values:

MostNotes_

MostVoices_

Definition at line 100 of file Counterpoint.hpp.

6.27.2.2. anonymous enum [inherited]

Enumeration values:

Unison

MinorSecond

MajorSecond

MinorThird

MajorThird

Fourth

Tritone

Fifth

MinorSixth

MajorSixth

MinorSeventh

MajorSeventh

Octave

Definition at line 215 of file Counterpoint.hpp.

6.27.2.3. anonymous enum [inherited]

Enumeration values:

Aeolian

Dorian

Phrygian

Lydian

Mixolydian

Ionian

Locrian

Definition at line 235 of file Counterpoint.hpp.

6.27.2.4. anonymous enum [inherited]

Enumeration values:

DirectMotion

ContraryMotion

ObliqueMotion

NoMotion

Definition at line 283 of file Counterpoint.hpp.

6.27.2.5. anonymous enum [inherited]

Enumeration values:

WholeNote

HalfNote

DottedHalfNote

QuarterNote

DottedQuarterNote

EighthNote

Definition at line 366 of file Counterpoint.hpp.

6.27.2.6. anonymous enum [inherited]

Enumeration values:

One

Two

Three

Four

Five

Six

Eight

Definition at line 388 of file Counterpoint.hpp.

6.27.2.7. anonymous enum [inherited]

Enumeration values:

infinity

Bad

RealBad

Definition at line 503 of file Counterpoint.hpp.

6.27.2.8. anonymous enum [inherited]

Enumeration values:

INTERVALS_WITH_BASS_SIZE

Definition at line 754 of file Counterpoint.hpp.

6.27.2.9. anonymous enum [inherited]

Enumeration values:

NumFields

Field

EndF

Definition at line 1207 of file Counterpoint.hpp.

6.27.2.10. anonymous enum

Enumeration values:

GenerateCounterpoint

CorrectCounterpoint

Definition at line 58 of file CounterpointNode.hpp.

6.27.3. Constructor & Destructor Documentation

6.27.3.1. csound::CounterpointNode::CounterpointNode ()

6.27.3.2. virtual csound::CounterpointNode::~CounterpointNode () [virtual]

6.27.4. Member Function Documentation

6.27.4.1. int Counterpoint::ABS (int *i*) [inline, inherited]

Definition at line 211 of file Counterpoint.hpp.

Referenced by Counterpoint::AnOctave(), Counterpoint::ASkip(), Counterpoint::AStep(), Counterpoint::ATenth(), Counterpoint::BadMelody(), Counterpoint::DirectMotionToPerfectConsonance(), and Counterpoint::Size().

6.27.4.2. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.27.4.3. void Counterpoint::AddInterval (int *n*) [inline, inherited]

Definition at line 762 of file Counterpoint.hpp.

6.27.4.4. int Counterpoint::ADissonance (int *Interval*, int *Cn*, int *Cp*, int *v*, int *Species*) [inline, inherited]

Definition at line 433 of file Counterpoint.hpp.

6.27.4.5. int Counterpoint::AnOctave (int *Interval*) [inline, inherited]

Definition at line 280 of file Counterpoint.hpp.

References Counterpoint::ABS(), and Counterpoint::Unison.

6.27.4.6. void Counterpoint::AnySpecies (int *OurMode*, int * *StartPitches*, int *CurV*, int *CantusFirmusLength*, int *Species*) [inline, inherited]

Definition at line 1471 of file Counterpoint.hpp.

6.27.4.7. void Counterpoint::ARRBLT (int * *dest*, int * *source*, int *num*) [inline, inherited]

Definition at line 214 of file Counterpoint.hpp.

6.27.4.8. int Counterpoint::ASeventh (int *Interval*) [inline, inherited]

Definition at line 279 of file Counterpoint.hpp.

References Counterpoint::MajorSeventh, and Counterpoint::MinorSeventh.

6.27.4.9. int Counterpoint::ASkip (int *Interval*) [inline, inherited]

Definition at line 275 of file Counterpoint.hpp.

References Counterpoint::ABS(), and Counterpoint::MajorSecond.

Referenced by Counterpoint::ConsecutiveSkipsInSameDirection().

6.27.4.10. int Counterpoint::AStep (int *Interval*) [inline, inherited]

Definition at line 276 of file Counterpoint.hpp.

References Counterpoint::ABS(), Counterpoint::MajorSecond, and Counterpoint::MinorSecond.

6.27.4.11. int Counterpoint::ATenth (int *Interval*) [inline, inherited]

Definition at line 281 of file Counterpoint.hpp.

References Counterpoint::ABS(), and Counterpoint::AThird().

6.27.4.12. int Counterpoint::AThird (int *Interval*) [inline, inherited]

Definition at line 278 of file Counterpoint.hpp.

References Counterpoint::MajorThird, and Counterpoint::MinorThird.

Referenced by Counterpoint::ATenth().

6.27.4.13. int Counterpoint::BadMelody (int *Intv*) [inline, inherited]

Definition at line 273 of file Counterpoint.hpp.

References Counterpoint::ABS(), Counterpoint::BadMelodyInterval, Counterpoint::MinorSixth, and Counterpoint::Octave.

6.27.4.14. int Counterpoint::Bass (int *Cn*, int *v*) [inline, inherited]

Definition at line 358 of file Counterpoint.hpp.

References Counterpoint::Cantus(), Counterpoint::MIN(), and Counterpoint::Other().

6.27.4.15. int Counterpoint::Beat8 (int *n*) [inline, inherited]

Definition at line 376 of file Counterpoint.hpp.

Referenced by Counterpoint::DownBeat().

6.27.4.16. void Counterpoint::BestFitFirst (int *CurTime*, int *CurrentPenalty*, int *NumParts*, int *Species*, int *BrLim*) [inline, inherited]

Definition at line 1344 of file Counterpoint.hpp.

6.27.4.17. int Counterpoint::Cantus (int *n*, int *v*) [inline, inherited]

Definition at line 346 of file Counterpoint.hpp.

References Counterpoint::Ctrpt, and Counterpoint::Onset.

Referenced by Counterpoint::Bass().

6.27.4.18. int Counterpoint::Check (int *Cn*, int *Cp*, int *v*, int *NumParts*, int *Species*, int *CurLim*) [inline, inherited]

Definition at line 902 of file Counterpoint.hpp.

6.27.4.19. void Counterpoint::CleanRhy () [inline, inherited]

Definition at line 1459 of file Counterpoint.hpp.

6.27.4.20. virtual void Counterpoint::clear () [virtual, inherited]

6.27.4.21. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.27.4.22. int Counterpoint::ConsecutiveSkipsInSameDirection (int *Pitch1*, int *Pitch2*, int *Pitch3*) [inline, inherited]

Definition at line 312 of file Counterpoint.hpp.

References Counterpoint::ASkip().

6.27.4.23. void Counterpoint::counterpoint (int *OurMode*, int * *StartPitches*, int *CurV*, int *CantusFirmusLength*, int *Species*, int * *cantus*) [inherited]

6.27.4.24. int Counterpoint::CurRhy (int *n*) [inline, inherited]

Definition at line 1458 of file Counterpoint.hpp.

6.27.4.25. int Counterpoint::DirectMotionToPerfectConsonance (int *Pitch1*, int *Pitch2*, int *Pitch3*, int *Pitch4*) [inline, inherited]

Definition at line 307 of file Counterpoint.hpp.

References Counterpoint::ABS(), Counterpoint::DirectMotion, Counterpoint::MotionType(), and Counterpoint::PerfectConsonance.

6.27.4.26. int Counterpoint::Doubled (int *Pitch*, int *Cn*, int *v*) [inline, inherited]

Definition at line 493 of file Counterpoint.hpp.

6.27.4.27. int Counterpoint::DownBeat (int *n*, int *v*) [inline, inherited]

Definition at line 377 of file Counterpoint.hpp.

References Counterpoint::Beat8(), and Counterpoint::Onset.

Referenced by Counterpoint::UpBeat().

6.27.4.28. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.27.4.29. int Counterpoint::ExtremeRange (int *Pitch*) [inline, inherited]

Definition at line 322 of file Counterpoint.hpp.

References Counterpoint::HighestSemitone, and Counterpoint::LowestSemitone.

6.27.4.30. void Counterpoint::fillCantus (int *c0*, int *c1*, int *c2*, int *c3*, int *c4*, int *c5*, int *c6*, int *c7*, int *c8*, int *c9*, int *c10*, int *c11*, int *c12*, int *c13*, int *c14*) [inline, inherited]

Definition at line 1548 of file Counterpoint.hpp.

6.27.4.31. void Counterpoint::FillRhyPat () [inline, inherited]

Definition at line 1421 of file Counterpoint.hpp.

6.27.4.32. int Counterpoint::FirstNote (int *n*, int *v*) [inline, inherited]

Definition at line 328 of file Counterpoint.hpp.

6.27.4.33. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.27.4.34. int Counterpoint::GoodRhy () [inline, inherited]

Definition at line 1460 of file Counterpoint.hpp.

6.27.4.35. virtual void Counterpoint::initialize (int *mostnotes*, int *mostvoices*) [virtual, inherited]

Referenced by Counterpoint::Counterpoint().

6.27.4.36. int Counterpoint::InMode (int *Pitch*, int *Mode*) [inline, inherited]

Definition at line 253 of file Counterpoint.hpp.

References Counterpoint::_Aeolian, Counterpoint::_Dorian, Counterpoint::_Ionian, Counterpoint::_Locrian, Counterpoint::_Lydian, Counterpoint::_Mixolydian, Counterpoint::_Phrygian, Counterpoint::Aeolian, Counterpoint::Dorian, Counterpoint::Ionian, Counterpoint::Locrian, Counterpoint::Lydian, Counterpoint::Mixolydian, and Counterpoint::Phrygian.

6.27.4.37. int Counterpoint::LastNote (int *n*, int *v*) [inline, inherited]

Definition at line 327 of file Counterpoint.hpp.

References Counterpoint::TotalNotes.

6.27.4.38. int Counterpoint::Look (int *CurPen*, int *CurVoice*, int *NumParts*, int *Species*, int *Lim*, int * *Pens*, int * *ls*, int * *CurNotes*) [inline, inherited]

Definition at line 1305 of file Counterpoint.hpp.

6.27.4.39. int Counterpoint::MAX (int *a*, int *b*) [inline, inherited]

Definition at line 213 of file Counterpoint.hpp.

Referenced by Counterpoint::TotalRange().

6.27.4.40. void Counterpoint::message (const char * *format*, va_list *valist*) [inline, inherited]

Definition at line 87 of file Counterpoint.hpp.

References Counterpoint::messageCallback.

6.27.4.41. void Counterpoint::message (const char * *format*, ...) [inline, inherited]

Definition at line 80 of file Counterpoint.hpp.

6.27.4.42. int Counterpoint::MIN (int *a*, int *b*) [inline, inherited]

Definition at line 212 of file Counterpoint.hpp.

Referenced by Counterpoint::Bass(), and Counterpoint::TotalRange().

6.27.4.43. int Counterpoint::MotionType (int *Pitch1*, int *Pitch2*, int *Pitch3*, int *Pitch4*) [inline, inherited]

Definition at line 291 of file Counterpoint.hpp.

References Counterpoint::ContraryMotion, Counterpoint::DirectMotion, Counterpoint::NoMotion, and Counterpoint::ObliqueMotion.

Referenced by Counterpoint::DirectMotionToPerfectConsonance().

6.27.4.44. int Counterpoint::NextToLastNote (int *n*, int *v*) [inline, inherited]

Definition at line 329 of file Counterpoint.hpp.

References Counterpoint::TotalNotes.

6.27.4.45. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.27.4.46. int Counterpoint::Other (int *Cn*, int *v*, int *v1*) [inline, inherited]

Definition at line 356 of file Counterpoint.hpp.

References Counterpoint::Ctrpt, Counterpoint::Onset, and Counterpoint::VIndex().

Referenced by Counterpoint::Bass().

6.27.4.47. int Counterpoint::OtherVoiceCheck (int *Cn*, int *Cp*, int *v*, int *NumParts*, int *Species*, int *CurLim*) [inline, inherited]

Definition at line 778 of file Counterpoint.hpp.

6.27.4.48. int Counterpoint::OutOfRange (int *Pitch*) [inline, inherited]

Definition at line 321 of file Counterpoint.hpp.

References Counterpoint::HighestSemitone, and Counterpoint::LowestSemitone.

6.27.4.49. int Counterpoint::PitchRepeats (int *Cn*, int *Cp*, int *v*) [inline, inherited]

Definition at line 380 of file Counterpoint.hpp.

References Counterpoint::Us().

6.27.4.50. virtual void csound::CounterpointNode::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *globalCoordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::Node.

6.27.4.51. float Counterpoint::RANDOM (float *amp*) [inline, inherited]

Definition at line 1452 of file Counterpoint.hpp.

6.27.4.52. int Counterpoint::SaveIndx (int *indx*, int * *Sp*) [inline, inherited]

Definition at line 1213 of file Counterpoint.hpp.

6.27.4.53. void Counterpoint::SaveResults (int *CurrentPenalty*, int *Penalty*, int *v1*, int *Species*) [inline, inherited]

Definition at line 1237 of file Counterpoint.hpp.

6.27.4.54. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.27.4.55. void Counterpoint::SetUs (int *n*, int *p*, int *v*) [inline, inherited]

Definition at line 330 of file Counterpoint.hpp.

References Counterpoint::Ctrpt.

6.27.4.56. int Counterpoint::Size (int *MelInt*) [inline, inherited]

Definition at line 398 of file Counterpoint.hpp.

References Counterpoint::ABS(), Counterpoint::Eight, Counterpoint::Fifth, Counterpoint::Five, Counterpoint::Four, Counterpoint::Fourth, Counterpoint::MajorSecond, Counterpoint::MajorThird, Counterpoint::MinorSecond, Counterpoint::MinorSixth, Counterpoint::MinorThird, Counterpoint::Octave, Counterpoint::One, Counterpoint::Six, Counterpoint::Three, Counterpoint::Two, and Counterpoint::Unison.

Referenced by Counterpoint::TooMuchOfInterval().

6.27.4.57. int Counterpoint::SpecialSpeciesCheck (int *Cn*, int *Cp*, int *v*, int *Other0*, int *Other1*, int *Other2*, int *NumParts*, int *Species*, int *MelInt*, int *Interval*, int *ActInt*, int *LastIntClass*, int *Pitch*, int *LastMelInt*, int *CurLim*) [inline, inherited]

Definition at line 593 of file Counterpoint.hpp.

6.27.4.58. void Counterpoint::toCsoundScore (std::string *filename*, double *secondsPerPulse*) [inherited]

6.27.4.59. int Counterpoint::TooMuchOfInterval (int *Cn*, int *Cp*, int *v*) [inline, inherited]

Definition at line 417 of file Counterpoint.hpp.

References Counterpoint::Ctrpt, and Counterpoint::Size().

6.27.4.60. int Counterpoint::TotalRange (int *Cn*, int *Cp*, int *v*) [inline, inherited]

Definition at line 332 of file Counterpoint.hpp.

References Counterpoint::MAX(), Counterpoint::MIN(), and Counterpoint::Us().

6.27.4.61. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.27.4.62. int Counterpoint::UpBeat (int *n*, int *v*) [inline, inherited]

Definition at line 378 of file Counterpoint.hpp.

References Counterpoint::DownBeat().

6.27.4.63. int Counterpoint::Us (int *n*, int *v*) [inline, inherited]

Definition at line 326 of file Counterpoint.hpp.

References Counterpoint::Ctrpt.

Referenced by Counterpoint::PitchRepeats(), and Counterpoint::TotalRange().

6.27.4.64. void Counterpoint::UsedRhy (int *n*) [inline, inherited]

Definition at line 1457 of file Counterpoint.hpp.

6.27.4.65. int Counterpoint::VIndex (int *Time*, int *VNum*) [inline, inherited]

Definition at line 348 of file Counterpoint.hpp.

References Counterpoint::Dur, Counterpoint::Onset, and Counterpoint::TotalNotes.

Referenced by Counterpoint::Other().

6.27.4.66. void Counterpoint::winners (int *v1*, int * *data*, int * *best*, int * *best1*, int * *best2*, int * *durs*) [inline, inherited]

Definition at line 1557 of file Counterpoint.hpp.

6.27.5. Member Data Documentation

6.27.5.1. int Counterpoint::_Aeolian[12] [static, inherited]

Definition at line 250 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.2. int Counterpoint::_Dorian[12] [static, inherited]

Definition at line 246 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.3. int Counterpoint::_Ionian[12] [static, inherited]

Definition at line 245 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.4. int Counterpoint::_Locrian[12] [static, inherited]

Definition at line 251 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.5. int Counterpoint::_Lydian[12] [static, inherited]

Definition at line 248 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.6. int Counterpoint::_Mixolydian[12] [static, inherited]

Definition at line 249 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.7. int Counterpoint::_Phrygian[12] [static, inherited]

Definition at line 247 of file Counterpoint.hpp.

Referenced by Counterpoint::InMode().

6.27.5.8. int Counterpoint::AllDone [inherited]

Definition at line 1205 of file Counterpoint.hpp.

6.27.5.9. int Counterpoint::AllVoicesSkipPenalty [inherited]

Definition at line 585 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.10. int Counterpoint::AscendingSixthPenalty [inherited]

Definition at line 568 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.11. int Counterpoint::AugmentedIntervalPenalty [inherited]

Definition at line 577 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.12. int Counterpoint::BadCadencePenalty [inherited]

Definition at line 543 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.13. int Counterpoint::BadMelodyInterval[13] [static, inherited]

Definition at line 272 of file Counterpoint.hpp.

Referenced by Counterpoint::BadMelody().

6.27.5.14. int Counterpoint::BadMelodyPenalty [inherited]

Definition at line 534 of file Counterpoint.hpp.
Referenced by Counterpoint::Counterpoint().

6.27.5.15. int Counterpoint::BasePitch [inherited]

Definition at line 324 of file Counterpoint.hpp.

6.27.5.16. boost::numeric::ublas::matrix<int> Counterpoint::BestFit [inherited]

Definition at line 110 of file Counterpoint.hpp.

6.27.5.17. boost::numeric::ublas::matrix<int> Counterpoint::BestFit1 [inherited]

Definition at line 111 of file Counterpoint.hpp.

6.27.5.18. boost::numeric::ublas::matrix<int> Counterpoint::BestFit2 [inherited]

Definition at line 112 of file Counterpoint.hpp.

6.27.5.19. int Counterpoint::BestFitPenalty [inherited]

Definition at line 1205 of file Counterpoint.hpp.

6.27.5.20. int Counterpoint::Branches [inherited]

Definition at line 1205 of file Counterpoint.hpp.

6.27.5.21. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.
Definition at line 57 of file Node.hpp.

6.27.5.22. int Counterpoint::CompoundPenalty [inherited]

Definition at line 523 of file Counterpoint.hpp.
Referenced by Counterpoint::Counterpoint().

6.27.5.23. int Counterpoint::CrossAboveCantusPenalty [inherited]

Definition at line 590 of file Counterpoint.hpp.
Referenced by Counterpoint::Counterpoint().

6.27.5.24. int Counterpoint::CrossBelowBassPenalty [inherited]

Definition at line 587 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.25. boost::numeric::ublas::matrix<int> Counterpoint::Ctrpt [inherited]

Definition at line 106 of file Counterpoint.hpp.

Referenced by Counterpoint::Cantus(), Counterpoint::Other(), Counterpoint::SetUs(), Counterpoint::TooMuchOfInterval(), and Counterpoint::Us().

6.27.5.26. int Counterpoint::DirectMotionPenalty [inherited]

Definition at line 521 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.27. int Counterpoint::DirectPerfectOnDownbeatPenalty [inherited]

Definition at line 544 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.28. int Counterpoint::DirectToFifthPenalty [inherited]

Definition at line 511 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.29. int Counterpoint::DirectToOctavePenalty [inherited]

Definition at line 512 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.30. int Counterpoint::DirectToTritonePenalty [inherited]

Definition at line 586 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.31. int Counterpoint::Dissonance[13] [static, inherited]

Definition at line 233 of file Counterpoint.hpp.

6.27.5.32. int Counterpoint::DissonanceNotFillingThirdPenalty [inherited]

Definition at line 546 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.33. int Counterpoint::DissonancePenalty [inherited]

Definition at line 517 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.34. int Counterpoint::DoubledFifthPenalty [inherited]

Definition at line 581 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.35. int Counterpoint::DoubledLeadingTonePenalty [inherited]

Definition at line 579 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.36. int Counterpoint::DoubledSixthPenalty [inherited]

Definition at line 580 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.37. int Counterpoint::DownBeatUnisonPenalty [inherited]

Definition at line 565 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.38. boost::numeric::ublas::matrix<int> Counterpoint::Dur [inherited]

Definition at line 108 of file Counterpoint.hpp.

Referenced by Counterpoint::VIndex().

6.27.5.39. int Counterpoint::EighthJumpPenalty [inherited]

Definition at line 559 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.40. int Counterpoint::EndOnPerfectPenalty [inherited]

Definition at line 515 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.41. int Counterpoint::ExtremeRangePenalty [inherited]

Definition at line 535 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.42. int Counterpoint::FifthFollowedBySameDirectionPenalty [inherited]

Definition at line 531 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.43. int Counterpoint::FifthPrecededBySameDirectionPenalty [inherited]

Definition at line 528 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.44. int Counterpoint::Fits[3] [inherited]

Definition at line 116 of file Counterpoint.hpp.

6.27.5.45. int Counterpoint::FourRepeatedNotesPenalty [inherited]

Definition at line 550 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.46. int csound::CounterpointNode::generationMode

Definition at line 63 of file CounterpointNode.hpp.

6.27.5.47. int Counterpoint::HalfUntiedPenalty [inherited]

Definition at line 560 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.48. int Counterpoint::HighestSemitone [inherited]

Definition at line 319 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint(), Counterpoint::ExtremeRange(), and Counterpoint::OutOfRange().

6.27.5.49. int Counterpoint::ImperfectConsonance[13] [static, inherited]

Definition at line 232 of file Counterpoint.hpp.

6.27.5.50. int Counterpoint::Indx[17] [static, inherited]

Definition at line 1303 of file Counterpoint.hpp.

6.27.5.51. int Counterpoint::InnerVoicesInDirectToPerfectPenalty [inherited]

Definition at line 572 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.52. int Counterpoint::InnerVoicesInDirectToTritonePenalty [inherited]

Definition at line 573 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.53. int Counterpoint::IntervalsWithBass[INTERVALS_WITH_BASS_SIZE] [inherited]

Definition at line 759 of file Counterpoint.hpp.

6.27.5.54. int Counterpoint::LeapAtCadencePenalty [inherited]

Definition at line 551 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.55. int Counterpoint::LesserLigaturePenalty [inherited]

Definition at line 556 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.56. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.27.5.57. int Counterpoint::LowerNeighborPenalty [inherited]

Definition at line 537 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.58. int Counterpoint::LowestSemitone [inherited]

Definition at line 318 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint(), Counterpoint::ExtremeRange(), and Counterpoint::OutOfRange().

6.27.5.59. int Counterpoint::LydianCadentialTritonePenalty [inherited]

Definition at line 536 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.60. int Counterpoint::MaxPenalty [inherited]

Definition at line 1205 of file Counterpoint.hpp.

6.27.5.61. int Counterpoint::MelodicBoredomPenalty [inherited]

Definition at line 562 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.62. int Counterpoint::MelodicTritonePenalty [inherited]

Definition at line 567 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.63. boost::mt19937 Counterpoint::mersenneTwister [static, inherited]

Definition at line 1450 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.64. void(* Counterpoint::messageCallback)(void *userdata, int attribute, const char *format, va_list valist) [inherited]

Referenced by Counterpoint::Counterpoint(), and Counterpoint::message().

6.27.5.65. int Counterpoint::Mode [inherited]

Definition at line 324 of file Counterpoint.hpp.

6.27.5.66. int Counterpoint::MostNotes [inherited]

Definition at line 98 of file Counterpoint.hpp.

6.27.5.67. int Counterpoint::MostVoices [inherited]

Definition at line 99 of file Counterpoint.hpp.

6.27.5.68. int csound::CounterpointNode::musicMode

Definition at line 64 of file CounterpointNode.hpp.

6.27.5.69. int Counterpoint::NoLeadingTonePenalty [inherited]

Definition at line 516 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.70. int Counterpoint::NoMotionAgainstOctavePenalty [inherited]

Definition at line 591 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.71. int Counterpoint::NotaCambiataPenalty [inherited]

Definition at line 552 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.72. int Counterpoint::NotaLigaturePenalty [inherited]

Definition at line 555 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.73. int Counterpoint::NotBestCadencePenalty [inherited]

Definition at line 553 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.74. int Counterpoint::NotContraryToOthersPenalty [inherited]

Definition at line 570 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.75. int Counterpoint::NoTimeForaLigaturePenalty [inherited]

Definition at line 558 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.76. int Counterpoint::NotTriadPenalty [inherited]

Definition at line 571 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.77. int Counterpoint::OctaveLeapPenalty [inherited]

Definition at line 542 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.78. boost::numeric::ublas::matrix<int> Counterpoint::Onset [inherited]

Definition at line 107 of file Counterpoint.hpp.

Referenced by Counterpoint::Cantus(), Counterpoint::DownBeat(), Counterpoint::Other(), and Counterpoint::VIndex().

6.27.5.79. int Counterpoint::OutOfModePenalty [inherited]

Definition at line 519 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.80. int Counterpoint::OutOfRangePenalty [inherited]

Definition at line 518 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.81. int Counterpoint::OverOctavePenalty [inherited]

Definition at line 540 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.82. int Counterpoint::OverTwelfthPenalty [inherited]

Definition at line 539 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.83. int Counterpoint::ParallelFifthPenalty [inherited]

Definition at line 513 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.84. int Counterpoint::ParallelUnisonPenalty [inherited]

Definition at line 514 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.85. float Counterpoint::PenaltyRatio [inherited]

Definition at line 1206 of file Counterpoint.hpp.

6.27.5.86. int Counterpoint::PerfectConsonance[13] [static, inherited]

Definition at line 231 of file Counterpoint.hpp.

Referenced by Counterpoint::DirectMotionToPerfectConsonance().

6.27.5.87. int Counterpoint::PerfectConsonancePenalty [inherited]

Definition at line 522 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.88. long Counterpoint::randx [inherited]

Definition at line 105 of file Counterpoint.hpp.

6.27.5.89. int Counterpoint::RepeatedPitchPenalty [inherited]

Definition at line 569 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.90. int Counterpoint::RepetitionOnUpbeatPenalty [inherited]

Definition at line 545 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.91. boost::numeric::ublas::vector<int> Counterpoint::RhyNotes [inherited]

Definition at line 115 of file Counterpoint.hpp.

6.27.5.92. boost::numeric::ublas::matrix<int> Counterpoint::RhyPat [inherited]

Definition at line 114 of file Counterpoint.hpp.

6.27.5.93. double csound::CounterpointNode::secondsPerPulse

Definition at line 67 of file CounterpointNode.hpp.

6.27.5.94. int Counterpoint::SixFiveChordPenalty [inherited]

Definition at line 574 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.95. int Counterpoint::SixthFollowedBySameDirectionPenalty [inherited]

Definition at line 532 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.96. int Counterpoint::SixthLeapPenalty [inherited]

Definition at line 541 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.97. int Counterpoint::SixthPrecededBySameDirectionPenalty [inherited]

Definition at line 529 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.98. int Counterpoint::SkipFollowedBySameDirectionPenalty [inherited]

Definition at line 530 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.99. int Counterpoint::SkipFromUnisonPenalty [inherited]

Definition at line 526 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.100. int Counterpoint::SkipPrecededBySameDirectionPenalty [inherited]

Definition at line 527 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.101. int Counterpoint::SkipTo8vePenalty [inherited]

Definition at line 525 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.102. int Counterpoint::SkipToDownBeatPenalty [inherited]

Definition at line 563 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.103. int csound::CounterpointNode::species

Definition at line 65 of file CounterpointNode.hpp.

6.27.5.104. int Counterpoint::TenthToOctavePenalty [inherited]

Definition at line 524 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.105. int Counterpoint::ThirdDoubledPenalty [inherited]

Definition at line 578 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.106. int Counterpoint::ThreeRepeatedNotesPenalty [inherited]

Definition at line 549 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.107. int Counterpoint::ThreeSkipsPenalty [inherited]

Definition at line 564 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.108. boost::numeric::ublas::vector<int> Counterpoint::TotalNotes [inherited]

Definition at line 109 of file Counterpoint.hpp.

Referenced by Counterpoint::LastNote(), Counterpoint::NextToLastNote(), and Counterpoint::VIndex().

6.27.5.109. int Counterpoint::TotalTime [inherited]

Definition at line 324 of file Counterpoint.hpp.

6.27.5.110. int Counterpoint::TripledBassPenalty [inherited]

Definition at line 582 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.111. int Counterpoint::TwoRepeatedNotesPenalty [inherited]

Definition at line 548 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.112. int Counterpoint::TwoSkipsNotInTriadPenalty [inherited]

Definition at line 533 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.113. int Counterpoint::TwoSkipsPenalty [inherited]

Definition at line 520 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.114. boost::variate_generator<boost::mt19937, boost::uniform_real<> >* Counterpoint::uniform_real_generator [inherited]

Definition at line 78 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint(), and Counterpoint::~Counterpoint().

6.27.5.115. int Counterpoint::UnisonDownbeatPenalty [inherited]

Definition at line 547 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.116. int Counterpoint::UnisonOnBeat4Penalty [inherited]

Definition at line 554 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.117. int Counterpoint::UnisonPenalty [inherited]

Definition at line 510 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.118. int Counterpoint::UnisonUpbeatPenalty [inherited]

Definition at line 561 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.119. int Counterpoint::UnpreparedSixFivePenalty [inherited]

Definition at line 575 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.120. int Counterpoint::UnresolvedLeadingTonePenalty [inherited]

Definition at line 584 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.121. int Counterpoint::UnresolvedLigaturePenalty [inherited]

Definition at line 557 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.122. int Counterpoint::UnresolvedSixFivePenalty [inherited]

Definition at line 576 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.123. int Counterpoint::UpperNeighborPenalty [inherited]

Definition at line 538 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.124. int Counterpoint::UpperVoicesTooFarApartPenalty [inherited]

Definition at line 583 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.125. boost::numeric::ublas::vector<int> Counterpoint::vbs [inherited]

Definition at line 113 of file Counterpoint.hpp.

6.27.5.126. int Counterpoint::VerticalTritonePenalty [inherited]

Definition at line 566 of file Counterpoint.hpp.

Referenced by Counterpoint::Counterpoint().

6.27.5.127. std::vector<int> csound::CounterpointNode::voiceBeginnings

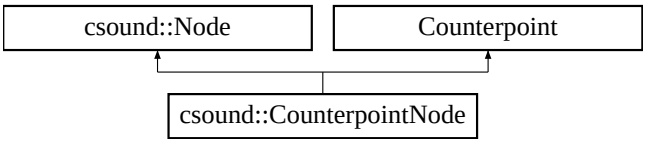
Definition at line 68 of file CounterpointNode.hpp.

6.27.5.128. size_t csound::CounterpointNode::voices

Definition at line 66 of file CounterpointNode.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CounterpointNode.hpp


csound::Node
Counterpoint
csound::CounterpointNode

6.28. CppSound Class Reference

`#include <CppSound.hpp>`

Inheritance diagram for CppSound::

6.28.1. Detailed Description

Class interface to the Csound API.

Definition at line 45 of file CppSound.hpp.

Public Member Functions

- CppSound ()
- virtual ~CppSound ()
- virtual int perform (int argc, char **argv)
- virtual int perform ()
- virtual void stop ()
- virtual int preCompile ()
- virtual int compile (int argc, char **argv)
- virtual int compile ()
- virtual int performKsmps (bool absolute=true)
- virtual void cleanup ()
- virtual void reset ()
- virtual MYFLT * getSpin ()
- virtual MYFLT * getSpout ()
- virtual size_t getSpoutSize () const
- virtual void setMessageCallback (void(*messageCallback)(void *hostData, int attr, const char *format, va_list args))
- virtual void message (const char *format,...)
- virtual void messageV (const char *format, va_list args)
- virtual void throwMessage (const char *format,...)
- virtual void throwMessageV (const char *format, va_list args)
- virtual void setThrowMessageCallback (void(*throwMessageCallback)(void *csound, const char *format, va_list args))
- virtual void setExternalMidiInOpenCallback (int(*ExternalMidiInOpen)(void *csound, void **userData, const char *devName))
- virtual void setExternalMidiReadCallback (int(*ExternalMidiRead)(void *csound, void *userData, unsigned char *buf, int nbytes))
- virtual void setExternalMidiInCloseCallback (int(*ExternalMidiInClose)(void *csound, void *userData))
- virtual int getKsmps ()
- virtual int getNchnls ()
- virtual void setMessageLevel (int messageLevel)
- virtual int getMessageLevel ()
- int appendOpcode (char *opname, int dsblksiz, int thread, char *outypes, char *intypes, SUBR iopadr, SUBR kopadr, SUBR aopadr)
- virtual int isScorePending ()
- virtual void setScorePending (int pending)
- virtual void setScoreOffsetSeconds (MYFLT offset)
- virtual MYFLT getScoreOffsetSeconds ()
- virtual void rewindScore ()

- virtual MYFLT getSr ()
- virtual MYFLT getKr ()
- virtual int loadExternals ()
- virtual void inputMessage (std::string istatement)
- virtual void * getCsound ()
- virtual void write (const char *text)
- virtual long getThis ()
- virtual bool getIsCompiled () const
- virtual void setIsPerforming (bool isPerforming)
- virtual bool getIsPerforming () const
- virtual bool getIsGo () const
- virtual void setPythonMessageCallback ()
- virtual int tableLength (int table)
- virtual MYFLT tableGet (int table, int index)
- virtual void tableSet (int table, int index, MYFLT value)
- virtual void scoreEvent (char opcode, std::vector< MYFLT > &pfields)
- virtual CsoundFile * getThisCsoundFile ()
- virtual void setFLTKThreadLocking (bool isLocking)
- virtual bool getFLTKThreadLocking ()
- virtual std::string getOutputSoundfileName () const
- virtual void setInputValueCallback (void(*inputValueCallback)(void *csound, char *channelName, MYFLT *value))
- virtual void setOutputValueCallback (void(*outputValueCallback)(void *csound, char *channelName, MYFLT value))
- virtual std::string generateFilename ()
- virtual std::string getFilename (void)
- virtual void setFilename (std::string name)
- virtual int load (std::string filename)
- virtual int load (std::istream &stream)
- virtual int save (std::string filename) const
- virtual int save (std::ostream &stream) const
- virtual int importFile (std::string filename)
- virtual int importFile (std::istream &stream)
- virtual int importCommand (std::istream &stream)
- virtual int exportCommand (std::ostream &stream) const
- virtual int importOrchestra (std::istream &stream)
- virtual int exportOrchestra (std::ostream &stream) const
- virtual int importScore (std::istream &stream)
- virtual int exportScore (std::ostream &stream) const
- virtual int importArrangement (std::istream &stream)
- virtual int exportArrangement (std::ostream &stream) const
- virtual int exportArrangementForPerformance (std::string filename) const
- virtual int exportArrangementForPerformance (std::ostream &stream) const
- virtual int importMidifile (std::istream &stream)
- virtual int exportMidifile (std::ostream &stream) const
- virtual std::string getCommand (void) const
- virtual void setCommand (std::string commandLine)
- virtual std::string getOrcFilename (void) const
- virtual std::string getScoFilename (void) const
- virtual std::string getMidiFilename (void) const
- virtual std::string getOrchestra (void) const
- virtual void setOrchestra (std::string orchestra)

- virtual int getInstrumentCount (void) const
- virtual std::string getOrchestraHeader (void) const
- virtual bool getInstrument (int number, std::string &definition) const
- virtual bool getInstrument (std::string name, std::string &definition) const
- virtual std::string getScore () const
- virtual void setScore (std::string score)
- virtual int getArrangementCount () const
- virtual std::string getArrangement (int index) const
- virtual void addArrangement (std::string instrument)
- virtual void setArrangement (int index, std::string instrument)
- virtual void insertArrangement (int index, std::string instrument)
- virtual void removeArrangement (int index)
- virtual void removeArrangement (void)
- virtual void setCSD (std::string xml)
- virtual std::string getCSD (void) const
- virtual void addScoreLine (const std::string line)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9, double p10, double p11)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9, double p10)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5)
- virtual void addNote (double p1, double p2, double p3, double p4)
- virtual void addNote (double p1, double p2, double p3)
- virtual bool exportForPerformance (void)
- virtual void removeAll (void)
- virtual void removeCommand (void)
- virtual void removeOrchestra (void)
- virtual void removeScore (void)
- virtual void removeMidifile (void)
- virtual bool loadOrcLibrary (const char *filename=0)

Public Attributes

- FUNC *(* ftfind)(MYFLT *index)
- std::string libraryFilename
- std::vector< std::string > arrangement

Protected Attributes

- ENVIRON * csound
- bool isCompiled
- bool isPerforming
- bool go
- size_t spoutSize
- std::string filename

- std::string command

- std::string orchestra

- std::string score

- std::vector< unsigned char > midifile

6.28.2. Constructor & Destructor Documentation

6.28.2.1. CppSound::CppSound ()

Default creator.

6.28.2.2. virtual CppSound::~CppSound () [virtual]

Virtual destructor.

6.28.3. Member Function Documentation

6.28.3.1. virtual void CsoundFile::addArrangement (std::string *instrument*) `[virtual, inherited]`

6.28.3.2. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*) `[virtual, inherited]`

6.28.3.3. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*) `[virtual, inherited]`

6.28.3.4. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*) `[virtual, inherited]`

6.28.3.5. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*) `[virtual, inherited]`

6.28.3.6. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*) `[virtual, inherited]`

6.28.3.7. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*) `[virtual, inherited]`

6.28.3.8. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*) `[virtual, inherited]`

6.28.3.9. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*, double *p10*) `[virtual, inherited]`

6.28.3.10. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*, double *p10*, double *p11*) `[virtual, inherited]`

6.28.3.11. virtual void CsoundFile::addScoreLine (const std::string *line*) `[virtual, inherited]`

6.28.3.12. int CppSound::appendOpcode (char * *opname*, int *dsblksiz*, int *thread*, char * *outypes*, char * *intypes*, SUBR *iopadr*, SUBR *kopadr*, SUBR *aopadr*)

Appends an opcode to the opcode list.

6.28.3.13. virtual void CppSound::cleanup () `[virtual]`

Must be called after the final call to performKsmps.

6.28.3.14. virtual int CppSound::compile () `[virtual]`

Using stored arguments, compiles the score and orchestra without performing them, in preparation for calling performKsmps.

6.28.3.15. virtual int CppSound::compile (int *argc*, char ** *argv*) [virtual]

Compiles the score and orchestra without performing them, in preparation for calling performKsmps.

6.28.3.16. virtual int CsoundFile::exportArrangement (std::ostream & *stream*) const [virtual, inherited]

6.28.3.17. virtual int CsoundFile::exportArrangementForPerformance (std::ostream & *stream*) const [virtual, inherited]

6.28.3.18. virtual int CsoundFile::exportArrangementForPerformance (std::string *filename*) const [virtual, inherited]

6.28.3.19. virtual int CsoundFile::exportCommand (std::ostream & *stream*) const [virtual, inherited]

6.28.3.20. virtual bool CsoundFile::exportForPerformance (void) [virtual, inherited]

6.28.3.21. virtual int CsoundFile::exportMidifile (std::ostream & *stream*) const [virtual, inherited]

6.28.3.22. virtual int CsoundFile::exportOrchestra (std::ostream & *stream*) const [virtual, inherited]

6.28.3.23. virtual int CsoundFile::exportScore (std::ostream & *stream*) const [virtual, inherited]

6.28.3.24. virtual std::string CsoundFile::generateFilename () [virtual, inherited]

6.28.3.25. virtual std::string CsoundFile::getArrangement (int *index*) const [virtual, inherited]

6.28.3.26. virtual int CsoundFile::getArrangementCount () const [virtual, inherited]

6.28.3.27. virtual std::string CsoundFile::getCommand (void) const [virtual, inherited]

6.28.3.28. virtual std::string CsoundFile::getCSD (void) const [virtual, inherited]

6.28.3.29. virtual void* CppSound::getCsound () [virtual]

Returns the actual instance of Csound.

6.28.3.30. virtual std::string CsoundFile::getFilename (void) [virtual, inherited]

6.28.3.31. virtual bool CppSound::getFLTKThreadLocking () [virtual]

Return whether the FLTK widgets thread calls Fl::lock().

6.28.3.32. virtual bool CsoundFile::getInstrument (std::string *name*, std::string & *definition*) const [virtual, inherited]

6.28.3.33. virtual bool CsoundFile::getInstrument (int *number*, std::string & *definition*) const [virtual, inherited]

6.28.3.34. virtual int CsoundFile::getInstrumentCount (void) const [virtual, inherited]

6.28.3.35. virtual bool CppSound::getIsCompiled () const [virtual]

Indicates whether orc and sco have been compiled.

6.28.3.36. virtual bool CppSound::getIsGo () const [virtual]

Indicates whether orc and sco have been compiled, and performance should continue.

6.28.3.37. virtual bool CppSound::getIsPerforming () const [virtual]

Indicates whether orc and sco have been compiled, and performance has now begun.

6.28.3.38. virtual MYFLT CppSound::getKr () [virtual]

Returns the number of control samples per second.

6.28.3.39. virtual int CppSound::getKsmps () [virtual]

Returns the number of audio sample frames per control sample.

6.28.3.40. virtual int CppSound::getMessageLevel () [virtual]

Returns the Csound message level (0 to 7).

6.28.3.41. virtual std::string CsoundFile::getMidiFilename (void) const [virtual, inherited]

6.28.3.42. virtual int CppSound::getNchnls () [virtual]

Returns the number of audio output channels.

6.28.3.43. virtual std::string CsoundFile::getOrcFilename (void) const [virtual, inherited]

6.28.3.44. virtual std::string CsoundFile::getOrchestra (void) const [virtual, inherited]

6.28.3.45. virtual std::string CsoundFile::getOrchestraHeader (void) const [virtual, inherited]

6.28.3.46. virtual std::string CppSound::getOutputSoundfileName () const [virtual]

Return the name of the output soundfile from the Csound parameters structure.

Reimplemented from CsoundFile.

6.28.3.47. virtual std::string CsoundFile::getScoFilename (void) const [virtual, inherited]

6.28.3.48. virtual std::string CsoundFile::getScore () const [virtual, inherited]

6.28.3.49. virtual MYFLT CppSound::getScoreOffsetSeconds () [virtual]

Csound events prior to the offset are consumed and discarded prior to beginning performance. Can be used by external software to begin performance midway through a Csound score.

6.28.3.50. virtual MYFLT* CppSound::getSpin () [virtual]

Returns the address of the Csound input buffer; external software can write to it before calling performKsmps.

6.28.3.51. virtual MYFLT* CppSound::getSpout () [virtual]

Returns the address of the Csound output buffer; external software can read from it after calling performKsmps.

6.28.3.52. virtual size_t CppSound::getSpoutSize () const [virtual]

Returns the size of the sample frame output buffer in bytes.

6.28.3.53. virtual MYFLT CppSound::getSr () [virtual]

Returns the number of audio sample frames per second.

6.28.3.54. virtual long CppSound::getThis () [virtual]

Shortcut for CsoundVST to get an instance pointer to a CppSound instance created in Python.

6.28.3.55. virtual CsoundFile* CppSound::getThisCsoundFile () [virtual]

Return a pointer to the CppSound base (for Java wrappers).

6.28.3.56. virtual int CsoundFile::importArrangement (std::istream & *stream*) [virtual, inherited]

6.28.3.57. virtual int CsoundFile::importCommand (std::istream & *stream*) [virtual, inherited]

6.28.3.58. virtual int CsoundFile::importFile (std::istream & *stream*) [virtual, inherited]

6.28.3.59. virtual int CsoundFile::importFile (std::string *filename*) [virtual, inherited]

6.28.3.60. virtual int CsoundFile::importMidifile (std::istream & *stream*) [virtual, inherited]

6.28.3.61. virtual int CsoundFile::importOrchestra (std::istream & *stream*) [virtual, inherited]

6.28.3.62. virtual int CsoundFile::importScore (std::istream & *stream*) [virtual, inherited]

6.28.3.63. virtual void CppSound::inputMessage (std::string *istatement*) [virtual]

Sends a line event.

6.28.3.64. virtual void CsoundFile::insertArrangement (int *index*, std::string *instrument*) [virtual, inherited]

6.28.3.65. virtual int CppSound::isScorePending () [virtual]

Returns whether Csound's score is synchronized with external software.

6.28.3.66. virtual int CsoundFile::load (std::istream & *stream*) [virtual, inherited]

6.28.3.67. virtual int CsoundFile::load (std::string *filename*) [virtual, inherited]

6.28.3.68. virtual int CppSound::loadExternals () [virtual]

Loads plugin opcodes.

6.28.3.69. virtual bool CsoundFile::loadOrcLibrary (const char * *filename* = 0) [virtual, inherited]

6.28.3.70. virtual void CppSound::message (const char * *format*, ...) [virtual]

Print an informational message.

6.28.3.71. virtual void CppSound::messageV (const char * *format*, va_list *args*) [virtual]

Print an informational message.

6.28.3.72. virtual int CppSound::perform () [virtual]

Using stored arguments, compiles and performs the orchestra and score, in one pass, just as Csound would do.

6.28.3.73. virtual int CppSound::perform (int *argc*, char ** *argv*) [virtual]

Using the specified arguments, compiles and performs the orchestra and score, in one pass, just as Csound would do.

6.28.3.74. virtual int CppSound::performKsmps (bool *absolute* = true) [virtual]

Causes Csound to read ksmps of audio sample frames from its input buffer, compute the performance, and write the performed sample frames to its output buffer. If absolute is true, performs a block of audio whether or not the Csound score is finished.

6.28.3.75. virtual int CppSound::preCompile () [virtual]

Reset and prepare an instance of Csound for compilation. Returns CSOUND_SUCCESS on success, and CSOUND_ERROR or CSOUND_MEMORY if an error occured.

6.28.3.76. virtual void CsoundFile::removeAll (void) [virtual, inherited]

6.28.3.77. virtual void CsoundFile::removeArrangement (void) [virtual, inherited]

6.28.3.78. virtual void CsoundFile::removeArrangement (int *index*) [virtual, inherited]

6.28.3.79. virtual void CsoundFile::removeCommand (void) [virtual, inherited]

6.28.3.80. virtual void CsoundFile::removeMidifile (void) [virtual, inherited]

6.28.3.81. virtual void CsoundFile::removeOrchestra (void) [virtual, inherited]

6.28.3.82. virtual void CsoundFile::removeScore (void) [virtual, inherited]

6.28.3.83. virtual void CppSound::reset () [virtual]

Resets all internal state.

6.28.3.84. virtual void CppSound::rewindScore () [virtual]

Rewind a compiled Csound score to its beginning.

6.28.3.85. virtual int CsoundFile::save (std::ostream & *stream*) const [virtual, inherited]

6.28.3.86. virtual int CsoundFile::save (std::string *filename*) const [virtual, inherited]

6.28.3.87. virtual void CppSound::scoreEvent (char *opcode*, std::vector< MYFLT > & *pfields*) [virtual]

Send a score event to Csound in real time.

6.28.3.88. virtual void CsoundFile::setArrangement (int *index*, std::string *instrument*) [virtual, inherited]

6.28.3.89. virtual void CsoundFile::setCommand (std::string *commandLine*) [virtual, inherited]

6.28.3.90. virtual void CsoundFile::setCSD (std::string *xml*) [virtual, inherited]

6.28.3.91. virtual void CppSound::setExternalMidiInCloseCallback (int(*)(void *csound, void *userData) *ExternalMidiInClose*) [virtual]

Called by external software to set a function for Csound to call to close MIDI input.

6.28.3.92. virtual void CppSound::setExternalMidiInOpenCallback (int(*)(void *csound, void **userData, const char *devName) *ExternalMidiInOpen*) [virtual]

Called by external software to set a function for Csound to call to open MIDI input.

6.28.3.93. virtual void CppSound::setExternalMidiReadCallback (int(*)(void *csound, void *userData, unsigned char *buf, int nbytes) *ExternalMidiRead*) [virtual]

Called by external software to set a function for Csound to call to read MIDI messages.

6.28.3.94. virtual void CsoundFile::setFilename (std::string *name*) [virtual, inherited]

6.28.3.95. virtual void CppSound::setFLTKThreadLocking (bool *isLocking*) [virtual]

Set whether the FLTK widgets thread calls Fl::lock().

6.28.3.96. virtual void CppSound::setInputValueCallback (void(*)(void *csound, char *channelName, MYFLT *value) *inputValueCallback*) [virtual]

Called by external software to set a function for Csound to fetch input control values. The 'invalue' opcodes will directly call this function.

6.28.3.97. virtual void CppSound::setIsPerforming (bool *isPerforming*) [virtual]

Sets whether orc and sco have been compiled, and performance has now begun.

6.28.3.98. virtual void CppSound::setMessageCallback (void(*)(void *hostData, int attr, const char *format, va_list args) *messageCallback*) [virtual]

Sets a function for Csound to call to print informational messages through external software.

6.28.3.99. virtual void CppSound::setMessageLevel (int *messageLevel*) [virtual]

Sets the message level (0 to 7).

6.28.3.100. virtual void CsoundFile::setOrchestra (std::string *orchestra*) [virtual, inherited]

6.28.3.101. virtual void CppSound::setOutputValueCallback (void(∗)(void ∗csound, char ∗channelName, MYFLT value) *outputValueCallback*) [virtual]

Called by external software to set a function for Csound to send output control values. The 'outvalue' opcodes will directly call this function.

6.28.3.102. virtual void CppSound::setPythonMessageCallback () [virtual]

Set up Python to print Csound messages.

6.28.3.103. virtual void CsoundFile::setScore (std::string *score*) [virtual, inherited]

6.28.3.104. virtual void CppSound::setScoreOffsetSeconds (MYFLT *offset*) [virtual]

Csound events prior to the offset are consumed and discarded prior to beginning performance. Can be used by external software to begin performance midway through a Csound score.

6.28.3.105. virtual void CppSound::setScorePending (int *pending*) [virtual]

Sets whether Csound's score is synchronized with external software.

6.28.3.106. virtual void CppSound::setThrowMessageCallback (void(∗)(void ∗csound, const char ∗format, va_list args) *throwMessageCallback*) [virtual]

Called by external software to set a funtion for Csound to stop execution with an error message or exception.

6.28.3.107. virtual void CppSound::stop () [virtual]

Stops the performance.

6.28.3.108. virtual MYFLT CppSound::tableGet (int *table*, int *index*) [virtual]

Returns the value of a slot in a functon table.

6.28.3.109. virtual int CppSound::tableLength (int *table*) [virtual]

Returns the length of a function table.

6.28.3.110. virtual void CppSound::tableSet (int *table*, int *index*, MYFLT *value*) [virtual]

Sets the value of a slot in a function table.

6.28.3.111. virtual void CppSound::throwMessage (const char ∗ *format*, ...) [virtual]

Stops execution with an error message or exception.

6.28.3.112. virtual void CppSound::throwMessageV (const char * *format*, va_list *args*) [virtual]

Stops execution with an error message or exception.

6.28.3.113. virtual void CppSound::write (const char * *text*) [virtual]

For Python.

6.28.4. Member Data Documentation

6.28.4.1. std::vector<std::string> CsoundFile::arrangement [inherited]

Definition at line 175 of file CsoundFile.hpp.

6.28.4.2. std::string CsoundFile::command [protected, inherited]

CsOptions

Definition at line 157 of file CsoundFile.hpp.

6.28.4.3. ENVIRON* CppSound::csound [protected]

Definition at line 49 of file CppSound.hpp.

6.28.4.4. std::string CsoundFile::filename [protected, inherited]

What are we storing, anyway?

Definition at line 153 of file CsoundFile.hpp.

6.28.4.5. FUNC*(* CppSound::ftfind)(MYFLT *index)

Returns a function table.

6.28.4.6. bool CppSound::go [protected]

The user controls this one.

Definition at line 58 of file CppSound.hpp.

6.28.4.7. bool CppSound::isCompiled [protected]

Definition at line 50 of file CppSound.hpp.

6.28.4.8. bool CppSound::isPerforming [protected]

Csound controls this one.

Definition at line 54 of file CppSound.hpp.

6.28.4.9. std::string CsoundFile::libraryFilename [inherited]

Patch library and arrangement.

Definition at line 174 of file CsoundFile.hpp.

6.28.4.10. std::vector<unsigned char> CsoundFile::midifile [protected, inherited]

CsMidi

Definition at line 169 of file CsoundFile.hpp.

6.28.4.11. std::string CsoundFile::orchestra [protected, inherited]

CsInstruments

Definition at line 161 of file CsoundFile.hpp.

6.28.4.12. std::string CsoundFile::score [protected, inherited]

CsScore

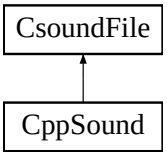
Definition at line 165 of file CsoundFile.hpp.

6.28.4.13. size_t CppSound::spoutSize [protected]

Definition at line 59 of file CppSound.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CppSound.hpp


CsoundFile
CppSound

6.29. Loris::PartialUtils::Cropper Class Reference

`#include <PartialUtils.h>`

6.29.1. Detailed Description

Trim a Partial by removing Breakpoints outside a specified time span. Insert a Breakpoint at the boundary when cropping occurs.

Definition at line 320 of file PartialUtils.h.

Public Member Functions

- Cropper (double t1, double t2)
- void operator() (Partial &p) const

Private Attributes

- double minTime
- double maxTime

6.29.2. Constructor & Destructor Documentation

6.29.2.1. Loris::PartialUtils::Cropper::Cropper (double *t1*, double *t2*) [inline]

Definition at line 325 of file PartialUtils.h.

References maxTime, and minTime.

6.29.3. Member Function Documentation

6.29.3.1. void Loris::PartialUtils::Cropper::operator() (Partial & *p*) const

6.29.4. Member Data Documentation

6.29.4.1. double Loris::PartialUtils::Cropper::maxTime [private]

Definition at line 335 of file PartialUtils.h.

Referenced by Cropper().

6.29.4.2. double Loris::PartialUtils::Cropper::minTime [private]

Definition at line 335 of file PartialUtils.h.

Referenced by Cropper().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.30. CsoundFile Class Reference

`#include <CsoundFile.hpp>`

Inheritance diagram for CsoundFile::

6.30.1. Detailed Description

Manages a Csound Structured Data (CSD) file with facilities for creating an arrangement of selected instruments in the orchestra, and for programmatically building score files.

Definition at line 147 of file CsoundFile.hpp.

Public Member Functions

- CsoundFile ()
- virtual ~CsoundFile (void)
- virtual std::string generateFilename ()
- virtual std::string getFilename (void)
- virtual void setFilename (std::string name)
- virtual int load (std::string filename)
- virtual int load (std::istream &stream)
- virtual int save (std::string filename) const
- virtual int save (std::ostream &stream) const
- virtual int importFile (std::string filename)
- virtual int importFile (std::istream &stream)
- virtual int importCommand (std::istream &stream)
- virtual int exportCommand (std::ostream &stream) const
- virtual int importOrchestra (std::istream &stream)
- virtual int exportOrchestra (std::ostream &stream) const
- virtual int importScore (std::istream &stream)
- virtual int exportScore (std::ostream &stream) const
- virtual int importArrangement (std::istream &stream)
- virtual int exportArrangement (std::ostream &stream) const
- virtual int exportArrangementForPerformance (std::string filename) const
- virtual int exportArrangementForPerformance (std::ostream &stream) const
- virtual int importMidifile (std::istream &stream)
- virtual int exportMidifile (std::ostream &stream) const
- virtual std::string getCommand (void) const
- virtual void setCommand (std::string commandLine)
- virtual std::string getOrcFilename (void) const
- virtual std::string getScoFilename (void) const
- virtual std::string getMidiFilename (void) const
- virtual std::string getOutputSoundfileName (void) const
- virtual std::string getOrchestra (void) const
- virtual void setOrchestra (std::string orchestra)
- virtual int getInstrumentCount (void) const
- virtual std::string getOrchestraHeader (void) const
- virtual bool getInstrument (int number, std::string &definition) const
- virtual bool getInstrument (std::string name, std::string &definition) const
- virtual std::string getScore () const
- virtual void setScore (std::string score)

- virtual int getArrangementCount () const
- virtual std::string getArrangement (int index) const
- virtual void addArrangement (std::string instrument)
- virtual void setArrangement (int index, std::string instrument)
- virtual void insertArrangement (int index, std::string instrument)
- virtual void removeArrangement (int index)
- virtual void setCSD (std::string xml)
- virtual std::string getCSD (void) const
- virtual void addScoreLine (const std::string line)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9, double p10, double p11)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9, double p10)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8, double p9)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7, double p8)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6, double p7)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5, double p6)
- virtual void addNote (double p1, double p2, double p3, double p4, double p5)
- virtual void addNote (double p1, double p2, double p3, double p4)
- virtual void addNote (double p1, double p2, double p3)
- virtual bool exportForPerformance (void)
- virtual void removeAll (void)
- virtual void removeCommand (void)
- virtual void removeOrchestra (void)
- virtual void removeScore (void)
- virtual void removeArrangement (void)
- virtual void removeMidifile (void)
- virtual bool loadOrcLibrary (const char *filename=0)

Public Attributes

- std::string libraryFilename
- std::vector< std::string > arrangement

Protected Attributes

- std::string filename
- std::string command
- std::string orchestra
- std::string score
- std::vector< unsigned char > midifile

6.30.2. Constructor & Destructor Documentation

6.30.2.1. CsoundFile::CsoundFile ()

6.30.2.2. virtual CsoundFile::~CsoundFile (void) [inline, virtual]

Definition at line 177 of file CsoundFile.hpp.

6.30.3. Member Function Documentation

6.30.3.1. virtual void CsoundFile::addArrangement (std::string *instrument*) [virtual]

6.30.3.2. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*) [virtual]

6.30.3.3. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*) [virtual]

6.30.3.4. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*) [virtual]

6.30.3.5. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*) [virtual]

6.30.3.6. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*) [virtual]

6.30.3.7. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*) [virtual]

6.30.3.8. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*) [virtual]

6.30.3.9. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*, double *p10*) [virtual]

6.30.3.10. virtual void CsoundFile::addNote (double *p1*, double *p2*, double *p3*, double *p4*, double *p5*, double *p6*, double *p7*, double *p8*, double *p9*, double *p10*, double *p11*) [virtual]

6.30.3.11. virtual void CsoundFile::addScoreLine (const std::string *line*) [virtual]

6.30.3.12. virtual int CsoundFile::exportArrangement (std::ostream & *stream*) const [virtual]

6.30.3.13. virtual int CsoundFile::exportArrangementForPerformance (std::ostream & *stream*) const [virtual]

6.30.3.14. virtual int CsoundFile::exportArrangementForPerformance (std::string *filename*) const [virtual]

6.30.3.15. virtual int CsoundFile::exportCommand (std::ostream & *stream*) const [virtual]

6.30.3.16. virtual bool CsoundFile::exportForPerformance (void) [virtual]

6.30.3.17. virtual int CsoundFile::exportMidifile (std::ostream & *stream*) const [virtual]

6.30.3.18. virtual int CsoundFile::exportOrchestra (std::ostream & *stream*) const [virtual]

6.30.3.19. virtual int CsoundFile::exportScore (std::ostream & *stream*) const [virtual]

6.30.3.20. virtual std::string CsoundFile::generateFilename () [virtual]

6.30.3.21. virtual std::string CsoundFile::getArrangement (int *index*) const [virtual]

6.30.3.22. virtual int CsoundFile::getArrangementCount () const [virtual]

6.30.3.23. virtual std::string CsoundFile::getCommand (void) const [virtual]

6.30.3.24. virtual std::string CsoundFile::getCSD (void) const [virtual]

6.30.3.34. virtual std::string CsoundFile::getScoFilename (void) const [virtual]

6.30.3.35. virtual std::string CsoundFile::getScore () const [virtual]

6.30.3.36. virtual int CsoundFile::importArrangement (std::istream & *stream*) [virtual]

6.30.3.37. virtual int CsoundFile::importCommand (std::istream & *stream*) [virtual]

6.30.3.38. virtual int CsoundFile::importFile (std::istream & *stream*) [virtual]

6.30.3.39. virtual int CsoundFile::importFile (std::string *filename*) [virtual]

6.30.3.40. virtual int CsoundFile::importMidifile (std::istream & *stream*) [virtual]

6.30.3.41. virtual int CsoundFile::importOrchestra (std::istream & *stream*) [virtual]

6.30.3.42. virtual int CsoundFile::importScore (std::istream & *stream*) [virtual]

6.30.3.43. virtual void CsoundFile::insertArrangement (int *index*, std::string *instrument*) [virtual]

6.30.3.44. virtual int CsoundFile::load (std::istream & *stream*) [virtual]

6.30.3.45. virtual int CsoundFile::load (std::string *filename*) [virtual]

6.30.3.46. virtual bool CsoundFile::loadOrcLibrary (const char * *filename* = 0) [virtual]

6.30.3.47. virtual void CsoundFile::removeAll (void) [virtual]

6.30.3.48. virtual void CsoundFile::removeArrangement (void) [virtual]

6.30.3.49. virtual void CsoundFile::removeArrangement (int *index*) [virtual]

6.30.3.50. virtual void CsoundFile::removeCommand (void) [virtual]

6.30.3.51. virtual void CsoundFile::removeMidifile (void) [virtual]

6.30.3.52. virtual void CsoundFile::removeOrchestra (void) [virtual]

6.30.3.53. virtual void CsoundFile::removeScore (void) [virtual]

6.30.3.54. virtual int CsoundFile::save (std::ostream & *stream*) const [virtual]

6.30.3.55. virtual int CsoundFile::save (std::string *filename*) const [virtual]

6.30.3.56. virtual void CsoundFile::setArrangement (int *index*, std::string *instrument*) [virtual]

6.30.3.57. virtual void CsoundFile::setCommand (std::string *commandLine*) [virtual]

6.30.3.58. virtual void CsoundFile::setCSD (std::string *xml*) [virtual]

6.30.3.59. virtual void CsoundFile::setFilename (std::string *name*) [virtual]

6.30.3.60. virtual void CsoundFile::setOrchestra (std::string *orchestra*) [virtual]

6.30.3.61. virtual void CsoundFile::setScore (std::string *score*) [virtual]

6.30.4. Member Data Documentation

6.30.4.1. std::vector<std::string> CsoundFile::arrangement

6.30.4.2. std::string CsoundFile::command [protected]

CsOptions

Definition at line 157 of file CsoundFile.hpp.

6.30.4.3. std::string CsoundFile::filename [protected]

What are we storing, anyway?

Definition at line 153 of file CsoundFile.hpp.

6.30.4.4. std::string CsoundFile::libraryFilename

Patch library and arrangement.

Definition at line 174 of file CsoundFile.hpp.

6.30.4.5. std::vector<unsigned char> CsoundFile::midifile [protected]

CsMidi

Definition at line 169 of file CsoundFile.hpp.

6.30.4.6. std::string CsoundFile::orchestra [protected]

CsInstruments

Definition at line 161 of file CsoundFile.hpp.

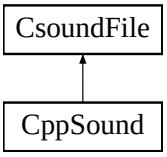
6.30.4.7. std::string CsoundFile::score [protected]

CsScore

Definition at line 165 of file CsoundFile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CsoundFile.hpp


CsoundFile
CppSound

6.31. CsoundVST Class Reference

`#include <CsoundVST.hpp>`

Inheritance diagram for CsoundVST::

Public Member Functions

- CsoundVST (audioMasterCallback audioMaster)
- virtual ~CsoundVST ()
- virtual AEffEditor * getEditor ()
- virtual bool getEffectName (char *name)
- virtual bool getVendorString (char *name)
- virtual bool getProductString (char *name)
- virtual long canDo (char *text)
- virtual bool getInputProperties (long index, VstPinProperties *properties)
- virtual bool getOutputProperties (long index, VstPinProperties *properties)
- virtual bool keysRequired ()
- virtual long getProgram ()
- virtual void setProgram (long program)
- virtual void setProgramName (char *name)
- virtual void getProgramName (char *name)
- virtual bool copyProgram (long destination)
- virtual bool getProgramNameIndexed (long category, long index, char *text)
- virtual long getChunk (void **data, bool isPreset)
- virtual long setChunk (void *data, long byteSize, bool isPreset)
- virtual void suspend ()
- virtual void resume ()
- virtual long processEvents (VstEvents *vstEvents)
- virtual void process (float **inputs, float **outputs, long sampleFrames)
- virtual void processReplacing (float **inputs, float **outputs, long sampleFrames)
- virtual void open ()
- CsoundVST ()
- virtual CppSound * getCppSound ()
- virtual bool getIsSynth () const
- virtual void setIsSynth (bool isSynth)
- virtual bool getIsVst () const
- virtual void setIsVst (bool isSynth)
- virtual bool getIsPython () const
- virtual void setIsPython (bool isPython)
- virtual void performanceThreadRoutine ()
- virtual int perform ()
- virtual std::string getText ()
- virtual void setText (const std::string text)
- virtual void synchronizeScore ()
- virtual void reset ()
- virtual void openFile (std::string filename)
- virtual int run ()
- virtual void openView (bool doRun=true)
- virtual void closeView ()
- virtual bool getIsMultiThreaded () const
- virtual void setIsMultiThreaded (bool isMultiThreaded)

- virtual bool getIsAutoPlayback () const
- virtual void setIsAutoPlayback (bool autoPlay)
- virtual void close ()
- virtual void main (int argc, char **argv)
- virtual void initialize ()
- virtual void clear ()
- virtual void setFilename (std::string filename)
- virtual std::string getFilename () const
- virtual std::string getOutputSoundfileName () const
- virtual std::string getMidiFilename () const
- virtual std::string getScript () const
- virtual void setScript (std::string text)
- virtual void load (std::string filename)
- virtual void loadAppend (std::string filename)
- virtual void save (std::string filename) const
- virtual void save () const
- virtual int runScript ()
- virtual int runScript (std::string script)
- virtual void stop ()

Static Public Member Functions

- static int midiDeviceOpen (void *csound, void **userData, const char *devName)
- static int midiRead (void *csound, void *userData, unsigned char *buf, int nbytes)
- static std::string generateFilename ()

Public Attributes

- std::vector< Preset > bank

Protected Types

- enum { kNumInputs = 2 }
- enum { kNumOutputs = 2 }
- enum { kNumPrograms = 10 }

Protected Attributes

- CppSound cppSound_
- CppSound * cppSound
- bool isSynth
- bool isVst
- bool isPython
- bool isMultiThreaded
- bool isAutoPlayback
- size_t csoundFrameI
- size_t csoundLastFrame
- size_t channelI
- size_t channelN
- size_t hostFrameI
- float vstSr

- float vstCurrentSampleBlockStart
- float vstCurrentSampleBlockEnd
- float vstCurrentSamplePosition
- float vstPriorSamplePosition
- CsoundVstFltk * csoundVstFltk
- std::list< VstMidiEvent > midiEventQueue
- std::string filename
- std::string script

Static Protected Attributes

- static double inputScale
- static double outputScale

6.31.1. Member Enumeration Documentation

6.31.1.1. anonymous enum [protected]

Enumeration values:
kNumInputs

Definition at line 62 of file CsoundVST.hpp.

6.31.1.2. anonymous enum [protected]

Enumeration values:
kNumOutputs

Definition at line 66 of file CsoundVST.hpp.

6.31.1.3. anonymous enum [protected]

Enumeration values:
kNumPrograms

Definition at line 70 of file CsoundVST.hpp.

6.31.2. Constructor & Destructor Documentation

6.31.2.1. CsoundVST::CsoundVST (audioMasterCallback *audioMaster*)

6.31.2.2. virtual CsoundVST::~CsoundVST () [virtual]

6.31.2.3. CsoundVST::CsoundVST ()

6.31.3. Member Function Documentation

6.31.3.1. virtual long CsoundVST::canDo (char * *text*) [virtual]

6.31.3.2. virtual void csound::Shell::clear () [virtual, inherited]

6.31.3.3. virtual void csound::Shell::close () [virtual, inherited]

6.31.3.4. virtual void CsoundVST::closeView () [virtual]

6.31.3.5. virtual bool CsoundVST::copyProgram (long *destination*) [virtual]

6.31.3.6. static std::string csound::Shell::generateFilename () [static, inherited]

6.31.3.7. virtual long CsoundVST::getChunk (void ** *data*, bool *isPreset*) [virtual]

6.31.3.8. virtual CppSound* CsoundVST::getCppSound () [virtual]

6.31.3.9. virtual AEffEditor* CsoundVST::getEditor () [virtual]

6.31.3.10. virtual bool CsoundVST::getEffectName (char * *name*) [virtual]

6.31.3.11. virtual std::string csound::Shell::getFilename () const [virtual, inherited]

6.31.3.12. virtual bool CsoundVST::getInputProperties (long *index*, VstPinProperties * *properties*) [virtual]

6.31.3.13. virtual bool CsoundVST::getIsAutoPlayback () const [virtual]

6.31.3.14. virtual bool CsoundVST::getIsMultiThreaded () const [virtual]

6.31.3.15. virtual bool CsoundVST::getIsPython () const [virtual]

6.31.3.16. virtual bool CsoundVST::getIsSynth () const [virtual]

6.31.3.17. virtual bool CsoundVST::getIsVst () const [virtual]

6.31.3.18. virtual std::string csound::Shell::getMidiFilename () const [virtual, inherited]

6.31.3.19. virtual bool CsoundVST::getOutputProperties (long *index*, VstPinProperties * *properties*) [virtual]

6.31.3.20. virtual std::string csound::Shell::getOutputSoundfileName () const [virtual, inherited]

6.31.3.21. virtual bool CsoundVST::getProductString (char * *name*) [virtual]

6.31.3.22. virtual long CsoundVST::getProgram () [virtual]

6.31.3.23. virtual void CsoundVST::getProgramName (char * *name*) [virtual]

6.31.3.24. virtual bool CsoundVST::getProgramNameIndexed (long *category*, long *index*, char * *text*) [virtual]

6.31.3.36. virtual void CsoundVST::openFile (std::string *filename*) [virtual]

6.31.3.37. virtual void CsoundVST::openView (bool *doRun* = true) [virtual]

6.31.3.38. virtual int CsoundVST::perform () [virtual]

6.31.3.39. virtual void CsoundVST::performanceThreadRoutine () [virtual]

6.31.3.40. virtual void CsoundVST::process (float ** *inputs*, float ** *outputs*, long *sampleFrames*) [virtual]

6.31.3.41. virtual long CsoundVST::processEvents (VstEvents * *vstEvents*) [virtual]

6.31.3.42. virtual void CsoundVST::processReplacing (float ** *inputs*, float ** *outputs*, long *sampleFrames*) [virtual]

6.31.3.43. virtual void CsoundVST::reset () [virtual]

6.31.3.44. virtual void CsoundVST::resume () [virtual]

6.31.3.45. virtual int CsoundVST::run () [virtual]

6.31.3.46. virtual int csound::Shell::runScript (std::string *script*) [virtual, inherited]

6.31.3.47. virtual int csound::Shell::runScript () [virtual, inherited]

6.31.3.48. virtual void csound::Shell::save () const [virtual, inherited]

6.31.3.49. virtual void csound::Shell::save (std::string *filename*) const [virtual, inherited]

6.31.3.50. virtual long CsoundVST::setChunk (void * *data*, long *byteSize*, bool *isPreset*) [virtual]

6.31.3.51. virtual void csound::Shell::setFilename (std::string *filename*) [virtual, inherited]

6.31.3.52. virtual void CsoundVST::setIsAutoPlayback (bool *autoPlay*) [virtual]

6.31.3.53. virtual void CsoundVST::setIsMultiThreaded (bool *isMultiThreaded*) [virtual]

6.31.3.54. virtual void CsoundVST::setIsPython (bool *isPython*) [virtual]

6.31.3.55. virtual void CsoundVST::setIsSynth (bool *isSynth*) [virtual]

6.31.3.56. virtual void CsoundVST::setIsVst (bool *isSynth*) [virtual]

6.31.3.57. virtual void CsoundVST::setProgram (long *program*) [virtual]

6.31.3.58. virtual void CsoundVST::setProgramName (char * *name*) [virtual]

6.31.3.59. virtual void csound::Shell::setScript (std::string *text*) [virtual, inherited]

6.31.3.60. virtual void CsoundVST::setText (const std::string *text*) [virtual]

6.31.3.61. virtual void csound::Shell::stop () [virtual, inherited]

6.31.3.62. virtual void CsoundVST::suspend () [virtual]

6.31.3.63. virtual void CsoundVST::synchronizeScore () [virtual]

6.31.4 Member Data Documentation

6.31.4.2. size_t CsoundVST::channelI [protected]

Definition at line 85 of file CsoundVST.hpp.

6.31.4.3. size_t CsoundVST::channelN [protected]

Definition at line 86 of file CsoundVST.hpp.

6.31.4.4. CppSound* CsoundVST::cppSound [protected]

Definition at line 77 of file CsoundVST.hpp.

6.31.4.5. CppSound CsoundVST::cppSound_ [protected]

Definition at line 76 of file CsoundVST.hpp.

6.31.4.6. size_t CsoundVST::csoundFrameI [protected]

Definition at line 83 of file CsoundVST.hpp.

6.31.4.7. size_t CsoundVST::csoundLastFrame [protected]

Definition at line 84 of file CsoundVST.hpp.

6.31.4.8. CsoundVstFltk* CsoundVST::csoundVstFltk [protected]

Definition at line 93 of file CsoundVST.hpp.

6.31.4.9. std::string csound::Shell::filename [protected, inherited]

Definition at line 44 of file Shell.hpp.

6.31.4.10. size_t CsoundVST::hostFrameI [protected]

Definition at line 87 of file CsoundVST.hpp.

6.31.4.11. double CsoundVST::inputScale [static, protected]

Definition at line 74 of file CsoundVST.hpp.

6.31.4.12. bool CsoundVST::isAutoPlayback [protected]

Definition at line 82 of file CsoundVST.hpp.

6.31.4.13. bool CsoundVST::isMultiThreaded [protected]

Definition at line 81 of file CsoundVST.hpp.

6.31.4.14. bool CsoundVST::isPython [protected]

Definition at line 80 of file CsoundVST.hpp.

6.31.4.15. bool CsoundVST::isSynth [protected]

Definition at line 78 of file CsoundVST.hpp.

6.31.4.16. bool CsoundVST::isVst [protected]

Definition at line 79 of file CsoundVST.hpp.

6.31.4.17. std::list<VstMidiEvent> CsoundVST::midiEventQueue [protected]

Definition at line 94 of file CsoundVST.hpp.

6.31.4.18. double CsoundVST::outputScale [static, protected]

Definition at line 75 of file CsoundVST.hpp.

6.31.4.19. std::string csound::Shell::script [protected, inherited]

Definition at line 45 of file Shell.hpp.

6.31.4.20. float CsoundVST::vstCurrentSampleBlockEnd [protected]

Definition at line 90 of file CsoundVST.hpp.

6.31.4.21. float CsoundVST::vstCurrentSampleBlockStart [protected]

Definition at line 89 of file CsoundVST.hpp.

6.31.4.22. float CsoundVST::vstCurrentSamplePosition [protected]

Definition at line 91 of file CsoundVST.hpp.

6.31.4.23. float CsoundVST::vstPriorSamplePosition [protected]

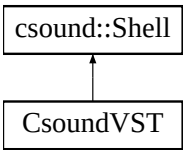
Definition at line 92 of file CsoundVST.hpp.

6.31.4.24. float CsoundVST::vstSr [protected]

Definition at line 88 of file CsoundVST.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CsoundVST.hpp


csound::Shell
CsoundVST

6.32. CsoundVstFltk Class Reference

`#include <CsoundVstFltk.hpp>`

Public Types

- enum AEffEditorSize { kEditorWidth = 610, kEditorHeight = 430, xPad = 4, yPad = 4 }

Public Member Functions

- CsoundVstFltk (AudioEffect *audioEffect)
- virtual ~CsoundVstFltk (void)
- virtual void updateCaption ()
- virtual void updateModel ()
- virtual long getRect (ERect **rect)
- virtual long open (void *windowHandle)
- virtual void close ()
- virtual void idle ()
- virtual void update ()
- virtual void postUpdate ()
- void onPerformScriptButtonThreadRoutine ()
- void onNew (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onNewVersion (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onOpen (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onImport (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onSave (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onSaveAs (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onPerform (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onStop (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onEdit (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onSettingsVstPluginMode (Fl_Check_Button *, CsoundVstFltk *csoundVstFltk)
- void onSettingsVstInstrumentMode (Fl_Check_Button *, CsoundVstFltk *csoundVstFltk)
- void onSettingsCsoundPerformanceModeClassic (Fl_Check_Button *, CsoundVstFltk *csoundVstFltk)
- void onSettingsCsoundPerformanceModePython (Fl_Check_Button *, CsoundVstFltk *csoundVstFltk)
- void onSettingsApply (Fl_Button *, CsoundVstFltk *csoundVstFltk)
- void onAutoPlayCheckButton (Fl_Check_Button *, CsoundVstFltk *csoundVstFltk)

Static Public Member Functions

- static void messageCallback (void *userdata, int attribute, const char *format, va_list valist)

Public Attributes

- void * windowHandle
- Fl_Window * csoundVstUi
- CsoundVST * csoundVST
- int useCount
- Fl_Pack * mainPack
- Fl_Tabs * mainTabs

- Fl_Input * commandInput
- Fl_Group * runtimeMessagesGroup
- Fl_Browser * runtimeMessagesBrowser
- Fl_Text_Editor * orchestraTextEdit
- Fl_Text_Buffer * orchestraTextBuffer
- Fl_Text_Editor * scoreTextEdit
- Fl_Text_Buffer * scoreTextBuffer
- Fl_Text_Editor * scriptTextEdit
- Fl_Text_Buffer * scriptTextBuffer
- Fl_Input * settingsEditSoundfileInput
- Fl_Check_Button * settingsVstPluginModeEffect
- Fl_Check_Button * settingsVstPluginModeInstrument
- Fl_Check_Button * settingsCsoundPerformanceModeClassic
- Fl_Check_Button * settingsCsoundPerformanceModePython
- Fl_Check_Button * autoPlayCheckButton
- Fl_Text_Buffer * aboutTextBuffer
- Fl_Text_Display * aboutTextDisplay
- Fl_Group * orchestraGroup
- Fl_Group * scoreGroup
- Fl_Group * scriptGroup
- std::list< std::string > messages
- std::string helpFilename
- std::string messagebuffer

Static Public Attributes

- static std::string aboutText
- static Fl_Preferences preferences

6.32.1. Member Enumeration Documentation

6.32.1.1. enum CsoundVstFltk::AEffEditorSize

Enumeration values:

kEditorWidth

kEditorHeight

xPad

yPad

Definition at line 69 of file CsoundVstFltk.hpp.

6.32.2. Constructor & Destructor Documentation

6.32.2.1. CsoundVstFltk::CsoundVstFltk (AudioEffect * *audioEffect*)

6.32.2.2. virtual CsoundVstFltk::~CsoundVstFltk (void) [virtual]

6.32.3. Member Function Documentation

6.32.3.1. virtual void CsoundVstFltk::close () [virtual]

6.32.3.2. virtual long CsoundVstFltk::getRect (ERect ** *rect*) [virtual]

6.32.3.3. virtual void CsoundVstFltk::idle () [virtual]

6.32.3.4. static void CsoundVstFltk::messageCallback (void * *userdata*, int *attribute*, const char * *format*, va_list *valist*) [static]

6.32.3.5. void CsoundVstFltk::onAutoPlayCheckButton (Fl_Check_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.6. void CsoundVstFltk::onEdit (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.7. void CsoundVstFltk::onImport (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.8. void CsoundVstFltk::onNew (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.9. void CsoundVstFltk::onNewVersion (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.10. void CsoundVstFltk::onOpen (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.11. void CsoundVstFltk::onPerform (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.12. void CsoundVstFltk::onPerformScriptButtonThreadRoutine ()

6.32.3.13. void CsoundVstFltk::onSave (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.14. void CsoundVstFltk::onSaveAs (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.15. void CsoundVstFltk::onSettingsApply (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.16. void CsoundVstFltk::onSettingsCsoundPerformanceModeClassic (Fl_Check_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.17. void CsoundVstFltk::onSettingsCsoundPerformanceModePython (Fl_Check_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.18. void CsoundVstFltk::onSettingsVstInstrumentMode (Fl_Check_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.19. void CsoundVstFltk::onSettingsVstPluginMode (Fl_Check_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.20. void CsoundVstFltk::onStop (Fl_Button *, CsoundVstFltk * *csoundVstFltk*)

6.32.3.21. virtual long CsoundVstFltk::open (void * *windowHandle*) [virtual]

6.32.3.22. virtual void CsoundVstFltk::postUpdate () [virtual]

6.32.3.23. virtual void CsoundVstFltk::update () [virtual]

6.32.3.24. virtual void CsoundVstFltk::updateCaption () [virtual]

6.32.4.2. Fl_Text_Buffer* CsoundVstFltk::aboutTextBuffer

Definition at line 92 of file CsoundVstFltk.hpp.

6.32.4.3. Fl_Text_Display* CsoundVstFltk::aboutTextDisplay

Definition at line 93 of file CsoundVstFltk.hpp.

6.32.4.4. Fl_Check_Button* CsoundVstFltk::autoPlayCheckButton

Definition at line 91 of file CsoundVstFltk.hpp.

6.32.4.5. Fl_Input* CsoundVstFltk::commandInput

Definition at line 77 of file CsoundVstFltk.hpp.

6.32.4.6. CsoundVST* CsoundVstFltk::csoundVST

Definition at line 65 of file CsoundVstFltk.hpp.

6.32.4.7. Fl_Window* CsoundVstFltk::csoundVstUi

Definition at line 64 of file CsoundVstFltk.hpp.

6.32.4.8. std::string CsoundVstFltk::helpFilename

Definition at line 98 of file CsoundVstFltk.hpp.

6.32.4.9. Fl_Pack* CsoundVstFltk::mainPack

Definition at line 75 of file CsoundVstFltk.hpp.

6.32.4.10. Fl_Tabs* CsoundVstFltk::mainTabs

Definition at line 76 of file CsoundVstFltk.hpp.

6.32.4.11. std::string CsoundVstFltk::messagebuffer

Definition at line 99 of file CsoundVstFltk.hpp.

6.32.4.12. std::list<std::string> CsoundVstFltk::messages

Definition at line 97 of file CsoundVstFltk.hpp.

6.32.4.13. Fl_Group* CsoundVstFltk::orchestraGroup

Definition at line 94 of file CsoundVstFltk.hpp.

6.32.4.14. Fl_Text_Buffer* CsoundVstFltk::orchestraTextBuffer

Definition at line 81 of file CsoundVstFltk.hpp.

6.32.4.15. Fl_Text_Editor* CsoundVstFltk::orchestraTextEdit

Definition at line 80 of file CsoundVstFltk.hpp.

6.32.4.16. Fl_Preferences CsoundVstFltk::preferences [static]

Definition at line 68 of file CsoundVstFltk.hpp.

6.32.4.17. Fl_Browser* CsoundVstFltk::runtimeMessagesBrowser

Definition at line 79 of file CsoundVstFltk.hpp.

6.32.4.18. Fl_Group* CsoundVstFltk::runtimeMessagesGroup

Definition at line 78 of file CsoundVstFltk.hpp.

6.32.4.19. Fl_Group* CsoundVstFltk::scoreGroup

Definition at line 95 of file CsoundVstFltk.hpp.

6.32.4.20. Fl_Text_Buffer* CsoundVstFltk::scoreTextBuffer

Definition at line 83 of file CsoundVstFltk.hpp.

6.32.4.21. Fl_Text_Editor* CsoundVstFltk::scoreTextEdit

Definition at line 82 of file CsoundVstFltk.hpp.

6.32.4.22. Fl_Group* CsoundVstFltk::scriptGroup

Definition at line 96 of file CsoundVstFltk.hpp.

6.32.4.23. Fl_Text_Buffer* CsoundVstFltk::scriptTextBuffer

Definition at line 85 of file CsoundVstFltk.hpp.

6.32.4.24. Fl_Text_Editor* CsoundVstFltk::scriptTextEdit

Definition at line 84 of file CsoundVstFltk.hpp.

6.32.4.25. Fl_Check_Button* CsoundVstFltk::settingsCsoundPerformanceModeClassic

Definition at line 89 of file CsoundVstFltk.hpp.

6.32.4.26. Fl_Check_Button* CsoundVstFltk::settingsCsoundPerformanceModePython

Definition at line 90 of file CsoundVstFltk.hpp.

6.32.4.27. Fl_Input* CsoundVstFltk::settingsEditSoundfileInput

Definition at line 86 of file CsoundVstFltk.hpp.

6.32.4.28. Fl_Check_Button* CsoundVstFltk::settingsVstPluginModeEffect

Definition at line 87 of file CsoundVstFltk.hpp.

6.32.4.29. Fl_Check_Button* CsoundVstFltk::settingsVstPluginModeInstrument

Definition at line 88 of file CsoundVstFltk.hpp.

6.32.4.30. int CsoundVstFltk::useCount

Definition at line 66 of file CsoundVstFltk.hpp.

6.32.4.31. void* CsoundVstFltk::windowHandle

Definition at line 63 of file CsoundVstFltk.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CsoundVstFltk.hpp

6.33. csRtAudioParams Struct Reference

`#include <csound.h>`

6.33.1. Detailed Description

Real-time audio parameters structure

Definition at line 776 of file csound.h.

Public Attributes

- char * devName
- int devNum
- int bufSamp_SW
- int bufSamp_HW
- int nChannels
- int sampleFormat
- float sampleRate

6.33.2. Member Data Documentation

6.33.2.1. int csRtAudioParams::bufSamp_HW

Definition at line 781 of file csound.h.

6.33.2.2. int csRtAudioParams::bufSamp_SW

Definition at line 779 of file csound.h.

6.33.2.3. char* csRtAudioParams::devName

Definition at line 777 of file csound.h.

6.33.2.4. int csRtAudioParams::devNum

Definition at line 778 of file csound.h.

6.33.2.5. int csRtAudioParams::nChannels

Definition at line 782 of file csound.h.

6.33.2.6. int csRtAudioParams::sampleFormat

Definition at line 783 of file csound.h.

6.33.2.7. float csRtAudioParams::sampleRate

Definition at line 784 of file csound.h.

The documentation for this struct was generated from the following file:

- H/csound.h

6.34. Curve Class Reference

`#include <curve.hpp>`

Public Member Functions

- Curve (float *, size_t, const std::string &, Polarity, float, float, float, float, bool)
- Curve (double *, size_t, const std::string &, Polarity, double, double, double, double, bool)
- Curve (const Curve &)
- Curve & operator= (const Curve &)
- ~Curve ()
- float * get_data () const
- size_t get_size () const
- std::string get_caption () const
- Polarity get_polarity () const
- float get_max () const
- float get_min () const
- float get_absmax () const
- float get_y_scale () const
- bool is_divider_dotted () const
- bool has_same_caption (Curve *) const

Private Member Functions

- void destroy ()

Static Private Member Functions

- static float * copy (size_t, float *)
- static float * copy (size_t, double *)

Private Attributes

- float * m_data
- size_t m_size
- std::string m_caption
- Polarity m_polarity
- float m_max
- float m_min
- float m_absmax
- float m_y_scale
- bool m_dotted_divider

6.34.1. Constructor & Destructor Documentation

6.34.1.1. Curve::Curve (float *, size_t, const std::string &, Polarity, float, float, float, float, bool)

6.34.1.2. Curve::Curve (double *, size_t, const std::string &, Polarity, double, double, double, double, bool)

6.34.1.3. Curve::Curve (const Curve &)

6.34.1.4. Curve::~Curve ()

6.34.2. Member Function Documentation

6.34.2.1. static float* Curve::copy (size_t, double *) [static, private]

6.34.2.2. static float* Curve::copy (size_t, float *) [static, private]

6.34.2.3. void Curve::destroy () [private]

6.34.2.4. float Curve::get_absmax () const

6.34.2.5. std::string Curve::get_caption () const

6.34.2.6. float* Curve::get_data () const

6.34.2.7. float Curve::get_max () const

6.34.2.8. float Curve::get_min () const

6.34.2.9. Polarity Curve::get_polarity () const

6.34.2.10. size_t Curve::get_size () const

6.34.2.11. float Curve::get_y_scale () const

6.34.2.12. bool Curve::has_same_caption (Curve *) const

6.34.2.13. bool Curve::is_divider_dotted () const

6.34.2.14. Curve& Curve::operator= (const Curve &)

6.34.3. Member Data Documentation

6.34.3.1. float Curve::m_absmax [private]

Definition at line 56 of file curve.hpp.

6.34.3.2. std::string Curve::m_caption [private]

Definition at line 54 of file curve.hpp.

6.34.3.3. float* Curve::m_data [private]

Definition at line 52 of file curve.hpp.

6.34.3.4. bool Curve::m_dotted_divider [private]

Definition at line 57 of file curve.hpp.

6.34.3.5. float Curve::m_max [private]

Definition at line 56 of file curve.hpp.

6.34.3.6. float Curve::m_min [private]

Definition at line 56 of file curve.hpp.

6.34.3.7. Polarity Curve::m_polarity [private]

Definition at line 55 of file curve.hpp.

6.34.3.8. size_t Curve::m_size [private]

Definition at line 53 of file curve.hpp.

6.34.3.9. float Curve::m_y_scale [private]

Definition at line 56 of file curve.hpp.

The documentation for this class was generated from the following file:

- frontends/flcsound/curve.hpp

6.35. Loris::Dilator Class Reference

`#include <Dilator.h>`

6.35.1. Detailed Description

Class Dilator represents an algorithm for non-uniformly expanding and contracting the Partial parameter envelopes according to the initial and target (desired) times of temporal features.

It is frequently necessary to redistribute temporal events in this way in preparation for a sound morph. For example, when morphing instrument tones, it is common to align the attack, sustain, and release portions of the source sounds by dilating or contracting those temporal regions.

This same procedure can be applied to the Markers stored in AiffFile, SdifFile, and SpcFile (see Marker.h).

Definition at line 63 of file Dilator.h.

Public Member Functions

- Dilator (void)
- template<typename Iter1, typename Iter2> Dilator (Iter1 ibegin, Iter1 iend, Iter2 tbegin)
- void insert (double i, double t)
- void dilate (Partial &p) const
- void operator() (Partial &p) const
- void dilate (Marker &m) const
- void operator() (Marker &m) const
- template<typename Iter> void dilate (Iter dilate_begin, Iter dilate_end) const
- template<typename Iter> void operator() (Iter dilate_begin, Iter dilate_end) const
- double warpTime (double currentTime) const

Static Public Member Functions

- template<typename PartialsIter, typename TimeIter1, typename TimeIter2> static void dilate (PartialsIter dilate_begin, PartialsIter dilate_end, TimeIter1 ibegin, TimeIter1 iend, TimeIter2 tbegin)

Private Attributes

- std::vector< double > _initial
- std::vector< double > _target

6.35.2. Constructor & Destructor Documentation

6.35.2.1. Loris::Dilator::Dilator (void)

Construct a new Dilator with no time points.

6.35.2.2. template<typename Iter1, typename Iter2> Loris::Dilator::Dilator (Iter1 *ibegin*, Iter1 *iend*, Iter2 *tbegin*)

Construct a new Dilator using a range of initial time points and a range of target (desired) time points. The client must ensure that the target range has at least as many elements as the initial range.

Parameters:

ibegin is the beginning of a sequence of initial, or source, time points.

iend is (one-past) the end of a sequence of initial, or source, time points.

tbegin is the beginning of a sequence of target time points; this sequence must be as long as the sequence of initial time point described by ibegin and iend.

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only const double * arguments.

Definition at line 269 of file Dilator.h.

6.35.3. Member Function Documentation

6.35.3.1. template<typename PartialsIter, typename TimeIter1, typename TimeIter2> void Loris::Dilator::dilate (PartialsIter *dilate_begin*, PartialsIter *dilate_end*, TimeIter1 *ibegin*, TimeIter1 *iend*, TimeIter2 *tbegin*) [static]

Parameters:

dilate_begin is the beginning of a sequence of Partials to dilate.

dilate_end is (one-past) the end of a sequence of Partials to dilate.

ibegin is the beginning of a sequence of initial, or source, time points.

iend is (one-past) the end of a sequence of initial, or source, time points.

tbegin is the beginning of a sequence of target time points; this sequence must be as long as the sequence of initial time point described by ibegin and iend.

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only PartialList::const_iterator arguments. Otherwise, this member also works for sequences of Markers. If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only const double * arguments for the times, otherwise, any iterator will do..

See also:

Dilator::dilate(Partial & p) const
Dilator::dilate(Marker & m) const

Definition at line 394 of file Dilator.h.

References dilate().

6.35.3.2. template<typename Iter> void Loris::Dilator::dilate (Iter *dilate_begin*, Iter *dilate_end*) const

Non-uniformly expand and contract the parameter envelopes of the each Partial in the specified half-open range according to this Dilator's stored initial and target (desired) times.

Parameters:

dilate_begin is the beginning of a sequence of Partials to dilate.

dilate_end is (one-past) the end of a sequence of Partials to dilate.

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only PartialList::const_iterator arguments. Otherwise, this member also works for sequences of Markers.

See also:
Dilator::dilate(Partial & p) const
Dilator::dilate(Marker & m) const

Definition at line 300 of file Dilator.h.

6.35.3.3. void Loris::Dilator::dilate (Marker & *m*) const

Compute a new time for the specified Marker using warpTime(), exactly as Partial Breakpoint times are recomputed. This can be used to dilate the Markers corresponding to a collection of Partials.

Parameters:
m is the Marker whose time should be recomputed.

6.35.3.4. void Loris::Dilator::dilate (Partial & *p*) const

Replace the Partial envelope with a new envelope having the same Breakpoints at times computed to align temporal features in the sorted sequence of initial time points with their counterparts the sorted sequence of target time points.

Depending on the specification of initial and target time points, the dilated Partial may have Breakpoints at times less than 0, even if the original Partial did not.

It is possible to have duplicate time points in either sequence. Duplicate initial time points result in very localized stretching. Duplicate target time points result in very localized compression.

If all initial time points are greater than 0, then an implicit time point at 0 is assumed in both initial and target sequences, so the onset of a sound can be stretched without explcitly specifying a zero point in each vector. (This seems most intuitive, and only looks like an inconsistency if clients are using negative time points in their Dilator, or Partials having Breakpoints before time 0, both of which are probably unusual circumstances.)

Parameters:
p is the Partial to dilate.

Referenced by dilate(), and operator()().

6.35.3.5. void Loris::Dilator::insert (double *i*, double *t*)

Insert a pair of initial and target time points.

Specify a pair of initial and target time points to be used by this Dilator, corresponding, for example, to the initial and desired time of a particular temporal feature in an analyzed sound.

Parameters:
i is an initial, or source, time point
t is a target time point

The time points will be sorted before they are used. If, in the sequences of initial and target time points, there are exactly the same number of initial time points preceding i as target time points preceding t, then time i will be warped to time t in the dilation process.

6.35.3.6. template<typename Iter> void Loris::Dilator::operator() (Iter *dilate_begin*, Iter *dilate_end*) const

Function call operator: same as dilate(Iter dilate_begin, Iter dilate_end)

If compiled with NO_TEMPLATE_MEMBERS defined, this member accepts only PartialList::const_iterator arguments. Otherwise, this member also works for sequences of Markers.

See also:
Dilator::dilate(Partial & p) const
Dilator::dilate(Marker & m) const

Definition at line 328 of file Dilator.h.

6.35.3.7. void Loris::Dilator::operator() (Marker & *m*) const [inline]

Function call operator: same as dilate(Marker & p).

Function call operator: same as dilate(Marker & m).

See also:
Dilator::dilate(Marker & m) const

Definition at line 359 of file Dilator.h.

References dilate().

6.35.3.8. void Loris::Dilator::operator() (Partial & *p*) const [inline]

Function call operator: same as dilate(Partial & p).

Function call operator: same as dilate(Partial & p).

See also:
Dilator::dilate(Partial & p) const

Definition at line 346 of file Dilator.h.

References dilate().

6.35.3.9. double Loris::Dilator::warpTime (double *currentTime*) const

Return the dilated time value corresponding to the specified initial time.

Parameters:
currentTime is a pre-dilated time.

Returns:
the dilated time corresponding to the initial time currentTime

6.35.4. Member Data Documentation

6.35.4.1. std::vector< double > Loris::Dilator::_initial [private]

Definition at line 67 of file Dilator.h.

6.35.4.2. std::vector< double > Loris::Dilator::_target [private]

Definition at line 67 of file Dilator.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Dilator.h

6.36. Loris::Distiller Class Reference

`#include <Distiller.h>`

6.36.1. Detailed Description

Class Distiller represents an algorithm for "distilling" a group of Partials that logically represent a single component into a single Partial.

The sound morphing algorithm in Loris requires that Partials in a given source be labeled uniquely, that is, no two Partials can have the same label. The Distiller enforces this condition. All Partials identified with a particular frequency channel (see Channelizer), and, therefore, having a common label, are distilled into a single Partial, leaving at most a single Partial per frequency channel and label. Channels that contain no Partials are not represented in the distilled data. Partials that are not labeled, that is, Partials having label 0, are are "collated " into groups of non-overlapping (in time) Partials, assigned an unused label (greater than the label associated with any frequency channel), and fused into a single Partial per group. "Collating" is a bit like "sifting" but non-overlapping Partials are grouped without regard to frequency proximity. This algorithm produces the smallest-possible number of collated Partials. Thanks to Ulrike Axen for providing this optimal algorithm.

Distillation modifies the Partial container (a PartialList). All Partials in the distilled range having a common label are replaced by a single Partial in the distillation process.

Definition at line 72 of file Distiller.h.

Public Member Functions

- Distiller (double partialFadeTime=0.001, double partialSilentTime=0.0001)
- template<typename Container> Container::iterator distill (Container &partials)
- template<typename Container> Container::iterator operator() (Container &partials)

Static Public Member Functions

- template<typename Container> static Container::iterator distill (Container &partials, double partialFadeTime, double partialSilentTime=0.0001)

Private Member Functions

- void distillOne (PartialList &partials, Partial::label_type label, PartialList &distilled)
- void collateUnlabeled (PartialList &unlabled, Partial::label_type startLabel)

Private Attributes

- double _fadeTime
- double _gapTime

6.36.2. Constructor & Destructor Documentation

6.36.2.1. Loris::Distiller::Distiller (double *partialFadeTime* = 0.001, double *partialSilentTime* = 0.0001) [explicit]

Construct a new Distiller using the specified fade time for gaps between Partials. When two non-overlapping Partials are distilled into a single Partial, the distilled Partial fades out at the end of the earlier Partial and back in again at the onset of the later one. The fade time is the time over which these fades occur. By default, use a 1 ms fade time. The gap time is the additional time over which a Partial faded out must remain at zero amplitude before it can fade back in. By default, use a gap time of one tenth of a millisecond, to prevent a pair of arbitrarily close null Breakpoints being inserted.

Parameters:

partialFadeTime is the time (in seconds) over which Partials joined by distillation fade to and from zero amplitude. Default is 0.001 (one millisecond).

partialSilentTime is the minimum duration (in seconds) of the silent (zero-amplitude) gap between two Partials joined by distillation. (Default is 0.0001 (one tenth of a millisecond).

6.36.3. Member Function Documentation

6.36.3.1. void Loris::Distiller::collateUnlabeled (PartialList & *unlabled*, Partial::label_type *startLabel*) [private]

Collate unlabeled (zero labeled) Partials into the smallest possible number of Partials that does not combine any temporally overlapping Partials. Give each collated Partial a label, starting with startlabel, and incrementing. The unlabeled Partials are collated in-place.

6.36.3.2. template<typename Container> Container::iterator Loris::Distiller::distill (Container & *partials*, double *partialFadeTime*, double *partialSilentTime* = 0.0001) [static]

Static member that constructs an instance and applies it to a sequence of Partials. Construct a Distiller using default parameters, and use it to distill a sequence of Partials.

Postcondition:

All Partials in the collection are uniquely-labeled

Parameters:

partials is the collection of Partials to distill in-place

partialFadeTime is the time (in seconds) over which Partials joined by distillation fade to and from zero amplitude.

partialSilentTime is the minimum duration (in seconds) of the silent (zero-amplitude) gap between two Partials joined by distillation. (Default is 0.0001 (one tenth of a millisecond).

Returns:

the position of the end of the range of distilled Partials, which is either the end of the collection, oor the position of the first collated Partial, composed of unlabeled Partials in the original collection.

If compiled with NO_TEMPLATE_MEMBERS defined, then partials must be a PartialList, otherwise it can be any container type storing Partials that supports at least bidirectional iterators.

Definition at line 394 of file Distiller.h.

References distill().

6.36.3.3. template<typename Container> Container::iterator Loris::Distiller::distill (Container & *partials*)

See also:
Distiller::distill(Container & partials)

Definition at line 239 of file Distiller.h.

Referenced by distill().

6.36.3.4. void Loris::Distiller::distillOne (PartialList & *partials*, Partial::label_type *label*, PartialList & *distilled*) [private]

Distill a list of Partials having a common label into a single Partial with that label, and append it to the distilled collection. If an empty list of Partials is passed, then an empty Partial having the specified label is appended.

6.36.3.5. template<typename Container> Container::iterator Loris::Distiller::operator() (Container & *partials*)

Function call operator: same as distill(PartialList & partials).

Function call operator: same as distill(PartialList & partials).

See also:
Distiller::distill(Container & partials)

Definition at line 356 of file Distiller.h.

6.36.4. Member Data Documentation

6.36.4.1. double Loris::Distiller::_fadeTime [private]

Definition at line 76 of file Distiller.h.

6.36.4.2. double Loris::Distiller::_gapTime [private]

Definition at line 76 of file Distiller.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Distiller.h

6.37. dklst Struct Reference

`#include <csoundCore.h>`

Public Attributes

- dklst * nxtlst
- long pgmno
- MYFLT keylst [1]

6.37.1. Member Data Documentation

6.37.1.1. MYFLT dklst::keylst[1]

Definition at line 246 of file csoundCore.h.

6.37.1.2. struct dklst* dklst::nxtlst

Definition at line 244 of file csoundCore.h.

6.37.1.3. long dklst::pgmno

Definition at line 245 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.38. DOWNDAT Struct Reference

`#include <csoundCore.h>`

Public Attributes

- long npts
- long nocts
- long nsamps
- MYFLT lofrq
- MYFLT hifrq
- MYFLT looct
- MYFLT srate
- OCTDAT octdata [MAXOCTS]
- AUXCH auxch

6.38.1. Member Data Documentation

6.38.1.1. AUXCH DOWNDAT::auxch

Definition at line 349 of file csoundCore.h.

6.38.1.2. MYFLT DOWNDAT::hifrq

Definition at line 347 of file csoundCore.h.

6.38.1.3. MYFLT DOWNDAT::lofrq

Definition at line 347 of file csoundCore.h.

6.38.1.4. MYFLT DOWNDAT::looct

Definition at line 347 of file csoundCore.h.

6.38.1.5. long DOWNDAT::nocts

Definition at line 346 of file csoundCore.h.

6.38.1.6. long DOWNDAT::npts

Definition at line 346 of file csoundCore.h.

6.38.1.7. long DOWNDAT::nsamps

Definition at line 346 of file csoundCore.h.

6.38.1.8. OCTDAT DOWNDAT::octdata[MAXOCTS]

Definition at line 348 of file csoundCore.h.

6.38.1.9. MYFLT DOWNDAT::srate

Definition at line 347 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.39. DPARM Struct Reference

`#include <csoundCore.h>`

Public Attributes

- DPEXCL dpexcl [8]
- int exclset [75]

6.39.1. Member Data Documentation

6.39.1.1. DPEXCL DPARM::dpexcl[8]

Definition at line 239 of file csoundCore.h.

6.39.1.2. int DPARM::exclset[75]

Definition at line 240 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.40. DPEXCL Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int notnum [4]

6.40.1. Member Data Documentation

6.40.1.1. int DPEXCL::notnum[4]

Definition at line 235 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.41. Envelope Class Reference

`#include <Envelope.h>`

6.41.1. Detailed Description

This class implements a simple envelope generator which is capable of ramping to a target value by a specified *rate*. It also responds to simple *keyOn* and *keyOff* messages, ramping to 1.0 on keyOn and to 0.0 on keyOff.

by Perry R. Cook and Gary P. Scavone, 1995 - 2004.

Definition at line 21 of file Envelope.h.

Public Member Functions

- Envelope (void)
- virtual ~Envelope (void)
- virtual void keyOn (void)
- virtual void keyOff (void)
- void setRate (StkFloat rate)
- void setTime (StkFloat time)
- virtual void setTarget (StkFloat target)
- virtual void setValue (StkFloat value)
- virtual int getState (void) const
- virtual StkFloat tick (void)
- virtual StkFloat * tick (StkFloat *vector, unsigned int vectorSize)
- virtual StkFrames & tick (StkFrames &frames, unsigned int channel=1)

Protected Attributes

- StkFloat value_
- StkFloat target_
- StkFloat rate_
- int state_

6.41.2. Constructor & Destructor Documentation

6.41.2.1. Envelope::Envelope (void)

Default constructor.

6.41.2.2. virtual Envelope::~Envelope (void) [virtual]

Class destructor.

6.41.3. Member Function Documentation

6.41.3.1. virtual int Envelope::getState (void) const [virtual]

Return the current envelope *state* (0 = at target, 1 otherwise).

6.41.3.2. virtual void Envelope::keyOff (void) [virtual]

Set target = 0.

6.41.3.3. virtual void Envelope::keyOn (void) [virtual]

Set target = 1.

6.41.3.4. void Envelope::setRate (StkFloat *rate*)

Set the *rate*.

6.41.3.5. virtual void Envelope::setTarget (StkFloat *target*) [virtual]

Set the target value.

6.41.3.6. void Envelope::setTime (StkFloat *time*)

Set the *rate* based on a time duration.

6.41.3.7. virtual void Envelope::setValue (StkFloat *value*) [virtual]

Set current and target values to *aValue*.

6.41.3.8. virtual StkFrames& Envelope::tick (StkFrames & *frames*, unsigned int *channel* = 1) [virtual]

Fill a channel of the StkFrames object with computed outputs.

The `channel` argument should be one or greater (the first channel is specified by 1). An StkError will be thrown if the `channel` argument is zero or it is greater than the number of channels in the StkFrames object.

6.41.3.9. virtual StkFloat* Envelope::tick (StkFloat * *vector*, unsigned int *vectorSize*) [virtual]

Compute *vectorSize* outputs and return them in *vector*.

6.41.3.10. virtual StkFloat Envelope::tick (void) [virtual]

Return one envelope output value.

6.41.4. Member Data Documentation

6.41.4.1. StkFloat Envelope::rate_ [protected]

Definition at line 70 of file Envelope.h.

6.41.4.2. int Envelope::state_ [protected]

Definition at line 71 of file Envelope.h.

6.41.4.3. StkFloat Envelope::target_ [protected]

Definition at line 69 of file Envelope.h.

6.41.4.4. StkFloat Envelope::value_ [protected]

Definition at line 68 of file Envelope.h.

The documentation for this class was generated from the following file:

- Opcodes/stk/include/Envelope.h

6.42. Loris::Envelope Class Reference

`#include <Envelope.h>`

Inheritance diagram for Loris::Envelope::

Public Member Functions

- virtual ~Envelope (void)
- virtual Envelope * clone (void) const =0
- virtual double valueAt (double x) const =0

Protected Member Functions

- Envelope (void)
- Envelope (const Envelope &)

6.42.1. Constructor & Destructor Documentation

6.42.1.1. virtual Loris::Envelope::~Envelope (void) [virtual]

Destroy this Envelope (virtual to allow subclassing).

6.42.1.2. Loris::Envelope::Envelope (void) [protected]

6.42.1.3. Loris::Envelope::Envelope (const Envelope &) [protected]

6.42.2. Member Function Documentation

6.42.2.1. virtual Envelope* Loris::Envelope::clone (void) const [pure virtual]

Return an exact copy of this Envelope (following the Prototype pattern).

Implemented in Loris::BreakpointEnvelope, and Loris::FrequencyReference.

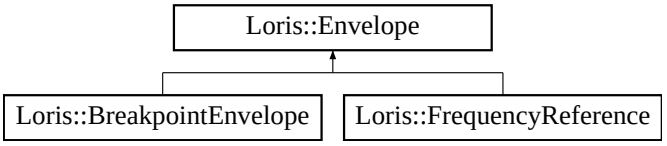
6.42.2.2. virtual double Loris::Envelope::valueAt (double *x*) const [pure virtual]

Return the value of this Envelope at the specified time.

Implemented in Loris::BreakpointEnvelope, and Loris::FrequencyReference.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Envelope.h


Loris::Envelope
Loris::BreakpointEnvelope
Loris::FrequencyReference

6.43. ENVIRON_ Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int(* GetVersion)(void)
- int(* GetAPIVersion)(void)
- void *(* GetHostData)(void *csound)
- void(* SetHostData)(void *csound, void *hostData)
- int(* Perform)(void *csound, int argc, char **argv)
- int(* Compile)(void *csound, int argc, char **argv)
- int(* PerformKsmps)(void *csound)
- int(* PerformBuffer)(void *csound)
- int(* Cleanup)(void *csound)
- void(* Reset)(void *csound)
- MYFLT(* GetSr)(void *csound)
- MYFLT(* GetKr)(void *csound)
- int(* GetKsmps)(void *csound)
- int(* GetNchnls)(void *csound)
- int(* GetSampleFormat)(void *csound)
- int(* GetSampleSize)(void *csound)
- long(* GetInputBufferSize)(void *csound)
- long(* GetOutputBufferSize)(void *csound)
- void *(* GetInputBuffer)(void *csound)
- void *(* GetOutputBuffer)(void *csound)
- MYFLT *(* GetSpin)(void *csound)
- MYFLT *(* GetSpout)(void *csound)
- MYFLT(* GetScoreTime)(void *csound)
- MYFLT(* GetProgress)(void *csound)
- MYFLT(* GetProfile)(void *csound)
- MYFLT(* GetCpuUsage)(void *csound)
- int(* IsScorePending)(void *csound)
- void(* SetScorePending)(void *csound, int pending)
- MYFLT(* GetScoreOffsetSeconds)(void *csound)
- void(* SetScoreOffsetSeconds)(void *csound, MYFLT offset)
- void(* RewindScore)(void *csound)
- void(* Message)(void *csound, const char *format,...)
- void(* MessageS)(void *csound, int attr, const char *format,...)
- void(* MessageV)(void *csound, int attr, const char *format, va_list args)
- void(* ThrowMessage)(void *csound, const char *format,...)
- void(* ThrowMessageV)(void *csound, const char *format, va_list args)
- void(* SetMessageCallback)(void *csound, void(*csoundMessageCallback)(void *hostData, int attr, const char *format, va_list valist))
- void(* SetThrowMessageCallback)(void *csound, void(*throwMessageCallback)(void *hostData, const char *format, va_list valist))
- int(* GetMessageLevel)(void *csound)
- void(* SetMessageLevel)(void *csound, int messageLevel)
- void(* InputMessage)(void *csound, const char *message__)
- void(* KeyPress)(void *csound, char c__)
- void(* SetInputValueCallback)(void *csound, void(*inputValueCalback)(void *hostData, char *channelName, MYFLT *value))

- void(* SetOutputValueCallback)(void *csound, void(*outputValueCalback)(void *hostData, char *channelName, MYFLT value))
- int(* ScoreEvent)(void *csound, char type, MYFLT *pFields, long numFields)
- void(* SetExternalMidiInOpenCallback)(void *csound, int(*func)(void *, void **, const char *))
- void(* SetExternalMidiReadCallback)(void *csound, int(*func)(void *, void *, unsigned char *, int))
- void(* SetExternalMidiInCloseCallback)(void *csound, int(*func)(void *, void *))
- void(* SetExternalMidiOutOpenCallback)(void *csound, int(*func)(void *, void **, const char *))
- void(* SetExternalMidiWriteCallback)(void *csound, int(*func)(void *, void *, unsigned char *, int))
- void(* SetExternalMidiOutCloseCallback)(void *csound, int(*func)(void *, void *))
- void(* SetExternalMidiErrorStringCallback)(void *csound, char *(*func)(int))
- void(* SetIsGraphable)(void *csound, int isGraphable)
- void(* SetMakeGraphCallback)(void *csound, void(*makeGraphCallback)(void *hostData, WINDAT *p, char *name))
- void(* SetDrawGraphCallback)(void *csound, void(*drawGraphCallback)(void *hostData, WINDAT *p))
- void(* SetKillGraphCallback)(void *csound, void(*killGraphCallback)(void *hostData, WINDAT *p))
- void(* SetExitGraphCallback)(void *csound, int(*exitGraphCallback)(void *hostData))
- opcodelist *(* NewOpcodeList)(void)
- void(* DisposeOpcodeList)(opcodelist *opcodelist_)
- int(* AppendOpcode)(void *csound, char *opname, int dsblksiz, int thread, char *outypes, char *intypes, int(*iopadr)(void *, void *), int(*kopadr)(void *, void *), int(*aopadr)(void *, void *))
- int(* AppendOpcodes)(void *csound, const OENTRY *opcodeList, int n)
- int(* LoadExternal)(void *csound, const char *libraryPath)
- int(* LoadExternals)(void *csound)
- void *(* OpenLibrary)(const char *libraryPath)
- int(* CloseLibrary)(void *library)
- void *(* GetLibrarySymbol)(void *library, const char *procedureName)
- int(* CheckEvents)(void *csound)
- void(* SetYieldCallback)(void *csound, int(*yieldCallback)(void *hostData))
- char *(* GetEnv)(void *csound, const char *name)
- char *(* FindInputFile)(void *csound, const char *filename, const char *envList)
- char *(* FindOutputFile)(void *csound, const char *filename, const char *envList)
- void(* SetPlayopenCallback)(void *csound, int(*playopen__)(void *csound, csRtAudioParams *parm))
- void(* SetRtplayCallback)(void *csound, void(*rtplay__)(void *csound, void *outBuf, int nbytes))
- void(* SetRecopenCallback)(void *csound, int(*recopen__)(void *csound, csRtAudioParams *parm))
- void(* SetRtrecordCallback)(void *csound, int(*rtrecord__)(void *csound, void *inBuf, int nbytes))
- void(* SetRtcloseCallback)(void *csound, void(*rtclose__)(void *csound))
- void(* AuxAlloc)(void *csound, long nbytes, AUXCH *auxchp)
- FUNC *(* FTFind)(void *csound, MYFLT *argp)
- FUNC *(* FTFindP)(void *csound, MYFLT *argp)
- FUNC *(* FTnp2Find)(void *csound, MYFLT *argp)
- MYFLT *(* GetTable)(void *csound, int tableNum, int *tableLength)
- void *(* Malloc)(void *csound, size_t nbytes)

- void *(* Calloc)(void *csound, size_t nbytes)
- void *(* ReAlloc)(void *csound, void *oldp, size_t nbytes)
- void(* Free)(void *csound, void *ptr)
- void(* Die)(void *csound, const char *msg,...)
- int(* InitError)(void *csound, const char *msg,...)
- int(* PerfError)(void *csound, const char *msg,...)
- void(* Warning)(void *csound, const char *msg,...)
- void(* DebugMsg)(void *csound, const char *msg,...)
- void(* dispset)(void *, WINDAT *, MYFLT *, long, char *, int, char *)
- void(* display)(void *, WINDAT *)
- int(* dispexit)(void *)
- MYFLT(* intpow)(MYFLT, long)
- MEMFIL *(* ldmemfile)(void *, const char *)
- SNDMEMFILE *(* LoadSoundFile)(void *, const char *, SF_INFO *)
- FUNC *(* hfgens)(struct ENVIRON_ *, EVTBLK *)
- int(* getopnum)(struct ENVIRON_ *, char *s)
- long(* strarg2insno)(struct ENVIRON_ *csound, void *p, int is_string)
- long(* strarg2opcno)(struct ENVIRON_ *csound, void *p, int is_string, int force_opcode)
- char *(* strarg2name)(struct ENVIRON_ *, char *, void *, const char *, int)
- int(* insert_score_event)(struct ENVIRON_ *, EVTBLK *, double, int)
- void(* rewriteheader)(SNDFILE *ofd, int verbose)
- void(* writeheader)(struct ENVIRON_ *csound, int ofd, char *ofname)
- void *(* SAsndgetset)(void *, char *, void *, MYFLT *, MYFLT *, MYFLT *, int)
- void *(* sndgetset)(void *, void *)
- int(* getsndin)(void *, void *, MYFLT *, int, void *)
- int(* PerformKsmpsAbsolute)(void *csound)
- int(* GetDebug)(void *csound)
- void(* SetDebug)(void *csound, int d)
- int(* TableLength)(void *csound, int table)
- MYFLT(* TableGet)(void *csound, int table, int index)
- void(* TableSet)(void *csound, int table, int index, MYFLT value)
- void *(* CreateThread)(void *csound, int(*threadRoutine)(void *userdata), void *userdata)
- int(* JoinThread)(void *csound, void *thread)
- void *(* CreateThreadLock)(void *csound)
- void(* WaitThreadLock)(void *csound, void *lock, size_t milliseconds)
- void(* NotifyThreadLock)(void *csound, void *lock)
- void(* DestroyThreadLock)(void *csound, void *lock)
- void(* SetFLTKThreadLocking)(void *csound, int isLocking)
- int(* GetFLTKThreadLocking)(void *csound)
- void(* timers_struct_init)(RTCLOCK *)
- double(* timers_get_real_time)(RTCLOCK *)
- double(* timers_get_CPU_time)(RTCLOCK *)
- unsigned long(* timers_random_seed)(void)
- char *(* LocalizeString)(const char *)
- int(* CreateGlobalVariable)(void *csound, const char *name, size_t nbytes)
- void *(* QueryGlobalVariable)(void *csound, const char *name)
- void *(* QueryGlobalVariableNoCheck)(void *csound, const char *name)
- int(* DestroyGlobalVariable)(void *csound, const char *name)
- int(* CreateConfigurationVariable)(void *csound, const char *name, void *p, int type, int flags, void *min, void *max, const char *shortDesc, const char *longDesc)
- int(* SetConfigurationVariable)(void *csound, const char *name, void *value)
- int(* ParseConfigurationVariable)(void *csound, const char *name, const char *value)

- csCfgVariable_t *(* QueryConfigurationVariable)(void *csound, const char *name)
- csCfgVariable_t **(* ListConfigurationVariables)(void *csound)
- int(* DeleteConfigurationVariable)(void *csound, const char *name)
- char *(* CfgErrorCodeToString)(int errcode)
- int(* GetSizeOfMYFLT)(void)
- void **(* GetRtRecordUserData)(void *csound)
- void **(* GetRtPlayUserData)(void *csound)
- MYFLT(* GetInverseComplexFFTScale)(void *csound, int FFTsize)
- MYFLT(* GetInverseRealFFTScale)(void *csound, int FFTsize)
- void(* ComplexFFT)(void *csound, MYFLT *buf, int FFTsize)
- void(* InverseComplexFFT)(void *csound, MYFLT *buf, int FFTsize)
- void(* RealFFT)(void *csound, MYFLT *buf, int FFTsize)
- void(* InverseRealFFT)(void *csound, MYFLT *buf, int FFTsize)
- void(* RealFFTMult)(void *csound, MYFLT *outbuf, MYFLT *buf1, MYFLT *buf2, int FFTsize, MYFLT scaleFac)
- void(* RealFFTnp2)(void *csound, MYFLT *buf, int FFTsize)
- void(* InverseRealFFTnp2)(void *csound, MYFLT *buf, int FFTsize)
- int(* AddUtility)(void *csound, const char *name, int(*UtilFunc)(void *, int, char **))
- int(* Utility)(void *csound, const char *name, int argc, char **argv)
- char **(* ListUtilities)(void *csound)
- int(* SetUtilityDescription)(void *csound, const char *utilName, const char *utilDesc)
- char *(* GetUtilityDescription)(void *csound, const char *utilName)
- int(* RegisterSenseEventCallback)(void *csound, void(*func)(void *, void *), void *userData)
- int(* RegisterDeinitCallback)(void *csound, void *p, int(*func)(void *, void *))
- int(* RegisterResetCallback)(void *csound, void *userData, int(*func)(void *, void *))
- void *(* CreateFileHandle)(void *, void *, int, const char *)
- void *(* FileOpen)(void *, void *, int, const char *, void *, const char *)
- char *(* GetFileName)(void *)
- int(* FileClose)(void *, void *)
- int(* PVOC_CreateFile)(struct ENVIRON_ *, const char *, unsigned long, unsigned long, unsigned long, unsigned long, long, int, int, float, float *, unsigned long)
- int(* PVOC_OpenFile)(struct ENVIRON_ *, const char *, void *, void *)
- int(* PVOC_CloseFile)(struct ENVIRON_ *, int)
- int(* PVOC_PutFrames)(struct ENVIRON_ *, int, const float *, long)
- int(* PVOC_GetFrames)(struct ENVIRON_ *, int, float *, unsigned long)
- int(* PVOC_FrameCount)(struct ENVIRON_ *, int)
- int(* PVOC_Rewind)(struct ENVIRON_ *, int, int)
- const char *(* PVOC_ErrorString)(struct ENVIRON_ *)
- int(* PVOCEX_LoadFile)(struct ENVIRON_ *, const char *, PVOCEX_MEMFILE *)
- void(* LongJmp)(struct ENVIRON_ *, int)
- SUBR dummyfn_1
- SUBR dummyfn_2 [116]
- OPDS * ids
- OPDS * pds
- int ksmps
- int global_ksmps
- int nchnls
- int spoutactive
- long kcounter
- long global_kcounter
- int reinitflag

- int tieflag
- MYFLT esr
- MYFLT onedsr
- MYFLT sicvt
- MYFLT tpidsr
- MYFLT pidsr
- MYFLT mpidsr
- MYFLT mtpdsr
- MYFLT onedksmps
- MYFLT ekr
- MYFLT global_ekr
- MYFLT onedkr
- MYFLT kicvt
- MYFLT e0dbfs
- MYFLT dbfs_to_float
- double timeOffs
- double beatOffs
- double curTime
- double curTime_inc
- double curBeat
- double curBeat_inc
- double beatTime
- unsigned int rtin_dev
- unsigned int rtout_dev
- char * rtin_devs
- char * rtout_devs
- int nchanik
- int nchania
- int nchanok
- int nchanoa
- MYFLT * chanik
- MYFLT * chania
- MYFLT * chanok
- MYFLT * chanoa
- MYFLT * zkstart
- MYFLT * zastart
- long zklast
- long zalast
- MYFLT * spin
- MYFLT * spout
- int nspin
- int nspout
- OPARMS * oparms
- EVTBLK * currevent
- INSDS * curip
- void * hostdata
- void * rtRecord_userdata
- void * rtPlay_userdata
- char * orchname
- char * scorename
- int holdrand
- int strVarMaxLen

- int maxinsno
- int strsmax
- char ** strsets
- INSTRTXT ** instrtxtp
- MCHNBLK * m_chnbp [64]

6.43.1. Member Data Documentation

6.43.1.1. int(* ENVIRON_::AddUtility)(void *csound, const char *name, int(*UtilFunc)(void *, int, char **))

6.43.1.2. int(* ENVIRON_::AppendOpcode)(void *csound, char *opname, int dsblksiz, int thread, char *outypes, char *intypes, int(*iopadr)(void *, void *), int(*kopadr)(void *, void *), int(*aopadr)(void *, void *))

6.43.1.3. int(* ENVIRON_::AppendOpcodes)(void *csound, const OENTRY *opcodeList, int n)

6.43.1.4. void(* ENVIRON_::AuxAlloc)(void *csound, long nbytes, AUXCH *auxchp)

6.43.1.5. double ENVIRON_::beatOffs

Definition at line 818 of file csoundCore.h.

6.43.1.6. double ENVIRON_::beatTime

Definition at line 821 of file csoundCore.h.

6.43.1.7. void*(* ENVIRON_::Calloc)(void *csound, size_t nbytes)

6.43.1.8. char*(* ENVIRON_::CfgErrorCodeToString)(int errcode)

6.43.1.9. MYFLT * ENVIRON_::chania

Definition at line 825 of file csoundCore.h.

6.43.1.10. MYFLT* ENVIRON_::chanik

Definition at line 825 of file csoundCore.h.

6.43.1.11. MYFLT * ENVIRON_::chanoa

Definition at line 825 of file csoundCore.h.

6.43.1.12. MYFLT * ENVIRON_::chanok

Definition at line 825 of file csoundCore.h.

6.43.1.13. int(* ENVIRON_::CheckEvents)(void *csound)

6.43.1.14. int(* ENVIRON_::Cleanup)(void *csound)

6.43.1.15. int(* ENVIRON_::CloseLibrary)(void *library)

6.43.1.16. int(* ENVIRON_::Compile)(void *csound, int argc, char **argv)

6.43.1.17. void(* ENVIRON_::ComplexFFT)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.18. int(* ENVIRON_::CreateConfigurationVariable)(void *csound, const char *name, void *p, int type, int flags, void *min, void *max, const char *shortDesc, const char *longDesc)

6.43.1.19. void*(* ENVIRON_::CreateFileHandle)(void *, void *, int, const char *)

6.43.1.20. int(* ENVIRON_::CreateGlobalVariable)(void *csound, const char *name, size_t nbytes)

6.43.1.21. void*(* ENVIRON_::CreateThread)(void *csound, int(*threadRoutine)(void *userdata), void *userdata)

6.43.1.22. void*(* ENVIRON_::CreateThreadLock)(void *csound)

6.43.1.23. double ENVIRON_::curBeat

Definition at line 820 of file csoundCore.h.

6.43.1.24. double ENVIRON_::curBeat_inc

Definition at line 820 of file csoundCore.h.

6.43.1.25. INSDS* ENVIRON_::curip

Definition at line 836 of file csoundCore.h.

6.43.1.26. EVTBLK* ENVIRON_::currevent

Definition at line 835 of file csoundCore.h.

6.43.1.27. double ENVIRON_::curTime

Definition at line 819 of file csoundCore.h.

6.43.1.28. double ENVIRON_::curTime_inc

Definition at line 819 of file csoundCore.h.

6.43.1.29. MYFLT ENVIRON_::dbfs_to_float

Definition at line 817 of file csoundCore.h.

6.43.1.30. void(* ENVIRON_::DebugMsg)(void *csound, const char *msg,...)

6.43.1.31. int(* ENVIRON_::DeleteConfigurationVariable)(void *csound, const char *name)

6.43.1.32. int(* ENVIRON_::DestroyGlobalVariable)(void *csound, const char *name)

6.43.1.33. void(* ENVIRON_::DestroyThreadLock)(void *csound, void *lock)

6.43.1.34. void(* ENVIRON_::Die)(void *csound, const char *msg,...)

6.43.1.35. int(* ENVIRON_::dispexit)(void *)

6.43.1.36. void(* ENVIRON_::display)(void *, WINDAT *)

6.43.1.37. void(* ENVIRON_::DisposeOpcodeList)(opcodelist *opcodelist_)

6.43.1.38. void(* ENVIRON_::dispset)(void *, WINDAT *, MYFLT *, long, char *, int, char *)

6.43.1.39. SUBR ENVIRON_::dummyfn_1

Definition at line 803 of file csoundCore.h.

6.43.1.40. SUBR ENVIRON_::dummyfn_2[116]

Definition at line 804 of file csoundCore.h.

6.43.1.41. MYFLT ENVIRON_::e0dbfs

Definition at line 817 of file csoundCore.h.

6.43.1.42. MYFLT ENVIRON_::ekr

Definition at line 814 of file csoundCore.h.

6.43.1.43. MYFLT ENVIRON_::esr

Definition at line 811 of file csoundCore.h.

6.43.1.44. int(* ENVIRON_::FileClose)(void *, void *)

6.43.1.45. void*(* ENVIRON_::FileOpen)(void *, void *, int, const char *, void *, const char *)

6.43.1.46. char*(* ENVIRON_::FindInputFile)(void *csound, const char *filename, const char *envList)

6.43.1.47. char*(* ENVIRON_::FindOutputFile)(void *csound, const char *filename, const char *envList)

6.43.1.48. void(* ENVIRON_::Free)(void *csound, void *ptr)

6.43.1.49. FUNC*(* ENVIRON_::FTFind)(void *csound, MYFLT *argp)

6.43.1.50. FUNC*(* ENVIRON_::FTFindP)(void *csound, MYFLT *argp)

6.43.1.51. FUNC*(* ENVIRON_::FTnp2Find)(void *csound, MYFLT *argp)

6.43.1.52. int(* ENVIRON_::GetAPIVersion)(void)

6.43.1.53. MYFLT(* ENVIRON_::GetCpuUsage)(void *csound)

6.43.1.54. int(* ENVIRON_::GetDebug)(void *csound)

6.43.1.55. char*(* ENVIRON_::GetEnv)(void *csound, const char *name)

6.43.1.56. char*(* ENVIRON_::GetFileName)(void *)

6.43.1.57. int(* ENVIRON_::GetFLTKThreadLocking)(void *csound)

6.43.1.58. void*(* ENVIRON_::GetHostData)(void *csound)

6.43.1.59. void*(* ENVIRON_::GetInputBuffer)(void *csound)

6.43.1.60. long(* ENVIRON_::GetInputBufferSize)(void *csound)

6.43.1.61. MYFLT(* ENVIRON_::GetInverseComplexFFTScale)(void *csound, int FFTsize)

6.43.1.62. MYFLT(* ENVIRON_::GetInverseRealFFTScale)(void *csound, int FFTsize)

6.43.1.63. MYFLT(* ENVIRON_::GetKr)(void *csound)

6.43.1.64. int(* ENVIRON_::GetKsmps)(void *csound)

6.43.1.65. void*(* ENVIRON_::GetLibrarySymbol)(void *library, const char *procedureName)

6.43.1.66. int(* ENVIRON_::GetMessageLevel)(void *csound)

6.43.1.67. int(* ENVIRON_::GetNchnls)(void *csound)

6.43.1.68. int(* ENVIRON_::getopnum)(struct ENVIRON_ *, char *s)

6.43.1.69. void*(* ENVIRON_::GetOutputBuffer)(void *csound)

6.43.1.70. long(* ENVIRON_::GetOutputBufferSize)(void *csound)

6.43.1.71. MYFLT(* ENVIRON_::GetProfile)(void *csound)

6.43.1.72. MYFLT(* ENVIRON_::GetProgress)(void *csound)

6.43.1.88. long ENVIRON_::global_kcounter

Definition at line 808 of file csoundCore.h.

6.43.1.89. int ENVIRON_::global_ksmps

Definition at line 807 of file csoundCore.h.

6.43.1.90. FUNC*(* ENVIRON_::hfgens)(struct ENVIRON_ *, EVTBLK *)

6.43.1.91. int ENVIRON_::holdrand

Definition at line 841 of file csoundCore.h.

6.43.1.92. void* ENVIRON_::hostdata

Definition at line 837 of file csoundCore.h.

6.43.1.93. OPDS* ENVIRON_::ids

Definition at line 806 of file csoundCore.h.

6.43.1.94. int(* ENVIRON_::InitError)(void *csound, const char *msg,...)

6.43.1.95. void(* ENVIRON_::InputMessage)(void *csound, const char *message__)

6.43.1.96. int(* ENVIRON_::insert_score_event)(struct ENVIRON_ *, EVTBLK *, double, int)

6.43.1.97. INSTRTXT** ENVIRON_::instrtxtp

Definition at line 846 of file csoundCore.h.

6.43.1.98. MYFLT(* ENVIRON_::intpow)(MYFLT, long)

6.43.1.99. void(* ENVIRON_::InverseComplexFFT)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.100. void(* ENVIRON_::InverseRealFFT)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.101. void(* ENVIRON_::InverseRealFFTnp2)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.102. int(* ENVIRON_::IsScorePending)(void *csound)

6.43.1.103. int(* ENVIRON_::JoinThread)(void *csound, void *thread)

6.43.1.104. long ENVIRON_::kcounter

Definition at line 808 of file csoundCore.h.

6.43.1.105. void(* ENVIRON_::KeyPress)(void *csound, char c__)

6.43.1.106. MYFLT ENVIRON_::kicvt

Definition at line 816 of file csoundCore.h.

6.43.1.107. int ENVIRON_::ksmps

Definition at line 807 of file csoundCore.h.

6.43.1.108. MEMFIL*(* ENVIRON_::ldmemfile)(void *, const char *)

6.43.1.109. csCfgVariable_t**(* ENVIRON_::ListConfigurationVariables)(void *csound)

6.43.1.110. char**(* ENVIRON_::ListUtilities)(void *csound)

6.43.1.111. int(* ENVIRON_::LoadExternal)(void *csound, const char *libraryPath)

6.43.1.112. int(* ENVIRON_::LoadExternals)(void *csound)

6.43.1.113. SNDMEMFILE*(* ENVIRON_::LoadSoundFile)(void *, const char *, SF_INFO *)

6.43.1.114. char*(* ENVIRON_::LocalizeString)(const char *)

6.43.1.115. void(* ENVIRON_::LongJmp)(struct ENVIRON_ *, int)

6.43.1.116. MCHNBLK* ENVIRON_::m_chnbp[64]

Definition at line 847 of file csoundCore.h.

6.43.1.117. void*(* ENVIRON_::Malloc)(void *csound, size_t nbytes)

6.43.1.118. int ENVIRON_::maxinsno

Definition at line 843 of file csoundCore.h.

6.43.1.119. void(* ENVIRON_::Message)(void *csound, const char *format,...)

6.43.1.120. void(* ENVIRON_::MessageS)(void *csound, int attr, const char *format,...)

6.43.1.121. void(* ENVIRON_::MessageV)(void *csound, int attr, const char *format, va_list args)

Referenced by OpcodeBase< T >::log(), and OpcodeBase< T >::warn().

6.43.1.122. MYFLT ENVIRON_::mpidsr

Definition at line 812 of file csoundCore.h.

6.43.1.123. MYFLT ENVIRON_::mtpdsr

Definition at line 812 of file csoundCore.h.

6.43.1.124. int ENVIRON_::nchania

Definition at line 824 of file csoundCore.h.

6.43.1.125. int ENVIRON_::nchanik

Definition at line 824 of file csoundCore.h.

6.43.1.126. int ENVIRON_::nchanoa

Definition at line 824 of file csoundCore.h.

6.43.1.127. int ENVIRON_::nchanok

Definition at line 824 of file csoundCore.h.

6.43.1.128. int ENVIRON_::nchnls

Definition at line 807 of file csoundCore.h.

6.43.1.129. opcodelist*(* ENVIRON_::NewOpcodeList)(void)

6.43.1.130. void(* ENVIRON_::NotifyThreadLock)(void *csound, void *lock)

6.43.1.131. int ENVIRON_::nspin

Definition at line 832 of file csoundCore.h.

6.43.1.132. int ENVIRON_::nspout

Definition at line 833 of file csoundCore.h.

6.43.1.133. MYFLT ENVIRON_::onedkr

Definition at line 815 of file csoundCore.h.

6.43.1.134. MYFLT ENVIRON_::onedksmps

Definition at line 813 of file csoundCore.h.

6.43.1.135. MYFLT ENVIRON_::onedsr

Definition at line 811 of file csoundCore.h.

6.43.1.136. OPARMS* ENVIRON_::oparms

Definition at line 834 of file csoundCore.h.

6.43.1.137. void*(* ENVIRON_::OpenLibrary)(const char *libraryPath)

6.43.1.138. char* ENVIRON_::orchname

Definition at line 840 of file csoundCore.h.

6.43.1.139. int(* ENVIRON_::ParseConfigurationVariable)(void *csound, const char *name, const char *value)

6.43.1.140. OPDS * ENVIRON_::pds

Definition at line 806 of file csoundCore.h.

6.43.1.141. int(* ENVIRON_::PerfError)(void *csound, const char *msg,...)

6.43.1.142. int(* ENVIRON_::Perform)(void *csound, int argc, char **argv)

6.43.1.143. int(* ENVIRON_::PerformBuffer)(void *csound)

6.43.1.144. int(* ENVIRON_::PerformKsmps)(void *csound)

6.43.1.145. int(* ENVIRON_::PerformKsmpsAbsolute)(void *csound)

6.43.1.146. MYFLT ENVIRON_::pidsr

Definition at line 812 of file csoundCore.h.

6.43.1.147. int(* ENVIRON_::PVOC_CloseFile)(struct ENVIRON_ *, int)

6.43.1.148. int(* ENVIRON_::PVOC_CreateFile)(struct ENVIRON_ *, const char *, unsigned long, unsigned long, unsigned long, unsigned long, long, int, int, float, float *, unsigned long)

6.43.1.149. const char*(* ENVIRON_::PVOC_ErrorString)(struct ENVIRON_ *)

6.43.1.150. int(* ENVIRON_::PVOC_FrameCount)(struct ENVIRON_ *, int)

6.43.1.151. int(* ENVIRON_::PVOC_GetFrames)(struct ENVIRON_ *, int, float *, unsigned long)

6.43.1.152. int(* ENVIRON_::PVOC_OpenFile)(struct ENVIRON_ *, const char *, void *, void *)

6.43.1.153. int(* ENVIRON_::PVOC_PutFrames)(struct ENVIRON_ *, int, const float *, long)

6.43.1.154. int(* ENVIRON_::PVOC_Rewind)(struct ENVIRON_ *, int, int)

6.43.1.155. int(* ENVIRON_::PVOCEX_LoadFile)(struct ENVIRON_ *, const char *, PVOCEX_MEMFILE *)

6.43.1.156. csCfgVariable_t*(* ENVIRON_::QueryConfigurationVariable)(void *csound, const char *name)

6.43.1.157. void*(* ENVIRON_::QueryGlobalVariable)(void *csound, const char *name)

6.43.1.158. void*(* ENVIRON_::QueryGlobalVariableNoCheck)(void *csound, const char *name)

6.43.1.159. void(* ENVIRON_::RealFFT)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.160. void(* ENVIRON_::RealFFTMult)(void *csound, MYFLT *outbuf, MYFLT *buf1, MYFLT *buf2, int FFTsize, MYFLT scaleFac)

6.43.1.161. void(* ENVIRON_::RealFFTnp2)(void *csound, MYFLT *buf, int FFTsize)

6.43.1.162. void*(* ENVIRON_::ReAlloc)(void *csound, void *oldp, size_t nbytes)

6.43.1.163. int(* ENVIRON_::RegisterDeinitCallback)(void *csound, void *p, int(*func)(void *, void *))

Referenced by OpcodeBase< T >::init_().

6.43.1.164. int(* ENVIRON_::RegisterResetCallback)(void *csound, void *userData, int(*func)(void *, void *))

6.43.1.165. int(* ENVIRON_::RegisterSenseEventCallback)(void *csound, void(*func)(void *, void *), void *userData)

6.43.1.166. int ENVIRON_::reinitflag

Definition at line 809 of file csoundCore.h.

Referenced by OpcodeBase< T >::init_().

6.43.1.167. void(* ENVIRON_::Reset)(void *csound)

6.43.1.168. void(* ENVIRON_::RewindScore)(void *csound)

6.43.1.169. void(* ENVIRON_::rewriteheader)(SNDFILE *ofd, int verbose)

6.43.1.170. unsigned int ENVIRON_::rtin_dev

Definition at line 822 of file csoundCore.h.

6.43.1.171. char* ENVIRON_::rtin_devs

Definition at line 823 of file csoundCore.h.

6.43.1.172. unsigned int ENVIRON_::rtout_dev

Definition at line 822 of file csoundCore.h.

6.43.1.173. char * ENVIRON_::rtout_devs

Definition at line 823 of file csoundCore.h.

6.43.1.174. void* ENVIRON_::rtPlay_userdata

Definition at line 839 of file csoundCore.h.

6.43.1.175. void* ENVIRON_::rtRecord_userdata

Definition at line 838 of file csoundCore.h.

6.43.1.176. void*(* ENVIRON_::SAsndgetset)(void *, char *, void *, MYFLT *, MYFLT *, MYFLT *, int)

6.43.1.177. int(* ENVIRON_::ScoreEvent)(void *csound, char type, MYFLT *pFields, long numFields)

6.43.1.178. char * ENVIRON_::scorename

Definition at line 840 of file csoundCore.h.

6.43.1.179. int(* ENVIRON_::SetConfigurationVariable)(void *csound, const char *name, void *value)

6.43.1.180. void(* ENVIRON_::SetDebug)(void *csound, int d)

6.43.1.181. void(* ENVIRON_::SetDrawGraphCallback)(void *csound, void(*drawGraphCallback)(void *hostData,WINDAT *p))

6.43.1.182. void(* ENVIRON_::SetExitGraphCallback)(void *csound, int(*exitGraphCallback)(void *hostData))

6.43.1.183. void(* ENVIRON_::SetExternalMidiErrorStringCallback)(void *csound, char *(*func)(int))

6.43.1.184. void(* ENVIRON_::SetExternalMidiInCloseCallback)(void *csound, int(*func)(void *, void *))

6.43.1.185. void(* ENVIRON_::SetExternalMidiInOpenCallback)(void *csound, int(*func)(void *, void **,const char *))

6.43.1.186. void(* ENVIRON_::SetExternalMidiOutCloseCallback)(void *csound, int(*func)(void *, void *))

6.43.1.187. void(* ENVIRON_::SetExternalMidiOutOpenCallback)(void *csound, int(*func)(void *, void **,const char *))

6.43.1.188. void(* ENVIRON_::SetExternalMidiReadCallback)(void *csound, int(*func)(void *, void *,unsigned char *, int))

6.43.1.189. void(* ENVIRON_::SetExternalMidiWriteCallback)(void *csound, int(*func)(void *, void *,unsigned char *, int))

6.43.1.190. void(* ENVIRON_::SetFLTKThreadLocking)(void *csound, int isLocking)

6.43.1.191. void(* ENVIRON_::SetHostData)(void *csound, void *hostData)

6.43.1.192. void(* ENVIRON_::SetInputValueCallback)(void *csound, void(*inputValueCalback)(void *hostData,char *channelName,MYFLT *value))

6.43.1.193. void(* ENVIRON_::SetIsGraphable)(void *csound, int isGraphable)

6.43.1.194. void(* ENVIRON_::SetKillGraphCallback)(void *csound, void(*killGraphCallback)(void *hostData,WINDAT *p))

6.43.1.195. void(* ENVIRON_::SetMakeGraphCallback)(void *csound, void(*makeGraphCallback)(void *hostData,WINDAT *p,char *name))

6.43.1.196. void(* ENVIRON_::SetMessageCallback)(void *csound, void(*csound-MessageCallback)(void *hostData,int attr,const char *format,va_list valist))

6.43.1.197. void(* ENVIRON_::SetMessageLevel)(void *csound, int messageLevel)

6.43.1.198. void(* ENVIRON_::SetOutputValueCallback)(void *csound, void(*outputValueCalback)(void *hostData,char *channelName,MYFLT value))

6.43.1.199. void(* ENVIRON_::SetPlayopenCallback)(void *csound, int(*playopen__)(void *csound,csRtAudioParams *parm))

6.43.1.200. void(* ENVIRON_::SetRecopenCallback)(void *csound, int(*recopen__)(void *csound,csRtAudioParams *parm))

6.43.1.210. void*(* ENVIRON_::sndgetset)(void *, void *)

6.43.1.211. MYFLT* ENVIRON_::spin

Definition at line 830 of file csoundCore.h.

6.43.1.212. MYFLT* ENVIRON_::spout

Definition at line 831 of file csoundCore.h.

6.43.1.213. int ENVIRON_::spoutactive

Definition at line 807 of file csoundCore.h.

6.43.1.214. long(* ENVIRON_::strarg2insno)(struct ENVIRON_ *csound, void *p, int is_string)

6.43.1.215. char*(* ENVIRON_::strarg2name)(struct ENVIRON_ *, char *, void *, const char *, int)

6.43.1.216. long(* ENVIRON_::strarg2opcno)(struct ENVIRON_ *csound, void *p, int is_string, int force_opcode)

6.43.1.217. char** ENVIRON_::strsets

Definition at line 845 of file csoundCore.h.

6.43.1.218. int ENVIRON_::strsmax

Definition at line 844 of file csoundCore.h.

6.43.1.219. int ENVIRON_::strVarMaxLen

Definition at line 842 of file csoundCore.h.

6.43.1.220. MYFLT(* ENVIRON_::TableGet)(void *csound, int table, int index)

6.43.1.221. int(* ENVIRON_::TableLength)(void *csound, int table)

6.43.1.222. void(* ENVIRON_::TableSet)(void *csound, int table, int index, MYFLT value)

6.43.1.223. void(* ENVIRON_::ThrowMessage)(void *csound, const char *format,...)

6.43.1.224. void(* ENVIRON_::ThrowMessageV)(void *csound, const char *format, va_list args)

6.43.1.225. int ENVIRON_::tieflag

Definition at line 810 of file csoundCore.h.

Referenced by OpcodeBase< T >::init_().

6.43.1.226. double ENVIRON_::timeOffs

Definition at line 818 of file csoundCore.h.

6.43.1.227. double(* ENVIRON_::timers_get_CPU_time)(RTCLOCK *)

6.43.1.228. double(* ENVIRON_::timers_get_real_time)(RTCLOCK *)

6.43.1.229. unsigned long(* ENVIRON_::timers_random_seed)(void)

6.43.1.230. void(* ENVIRON_::timers_struct_init)(RTCLOCK *)

6.43.1.231. MYFLT ENVIRON_::tpidsr

Definition at line 812 of file csoundCore.h.

6.43.1.232. int(* ENVIRON_::Utility)(void *csound, const char *name, int argc, char **argv)

6.43.1.233. void(* ENVIRON_::WaitThreadLock)(void *csound, void *lock, size_t milliseconds)

6.43.1.234. void(* ENVIRON_::Warning)(void *csound, const char *msg,...)

6.43.1.235. void(* ENVIRON_::writeheader)(struct ENVIRON_ *csound, int ofd, char *ofname)

6.43.1.236. long ENVIRON_::zalast

Definition at line 829 of file csoundCore.h.

6.43.1.237. MYFLT* ENVIRON_::zastart

Definition at line 827 of file csoundCore.h.

6.43.1.238. long ENVIRON_::zklast

Definition at line 828 of file csoundCore.h.

6.43.1.239. MYFLT* ENVIRON_::zkstart

Definition at line 826 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.44. event Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char * strarg
- char opcod
- short pcnt
- MYFLT p2orig
- MYFLT p3orig
- MYFLT p [PMAX+1]

6.44.1. Member Data Documentation

6.44.1.1. char event::opcod

Definition at line 410 of file csoundCore.h.

6.44.1.2. MYFLT event::p[PMAX+1]

Definition at line 414 of file csoundCore.h.

6.44.1.3. MYFLT event::p2orig

Definition at line 412 of file csoundCore.h.

6.44.1.4. MYFLT event::p3orig

Definition at line 413 of file csoundCore.h.

6.44.1.5. short event::pcnt

Definition at line 411 of file csoundCore.h.

6.44.1.6. char* event::strarg

Definition at line 409 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.45. csound::Event Class Reference

`#include <Event.hpp>`

6.45.1. Detailed Description

Represents an event in music space, such as a note of definite duration, a MIDI-like "note on" or "note off" event, or a MIDI-like control event. Fields have the same semantics as MIDI with some differences. All fields are floats; status is stored separately from channel; channel can have any positive value; spatial location in X, Y, and Z are stored; phase in radians is stored; and pitch-class set is stored.

Events can be multiplied (matrix dot product) with the local coordinate system of a Node or transform to translate, scale, or rotate them in any or all dimensions of music space.

Events usually are value objects, not references.

Silence Events translate to Csound score statements ("i" statements), but they are always real-time score statements at time 0, suitable for use with Csound's -L or line event option.

Definition at line 70 of file Event.hpp.

Public Types

- enum Dimensions {
  TIME = 0, DURATION, STATUS, INSTRUMENT,
  KEY, VELOCITY, PHASE, PAN,
  DEPTH, HEIGHT, PITCHES, HOMOGENEITY,
  ELEMENT_COUNT }
- enum { INDEFINITE = 16384 }

Public Member Functions

- Event ()
- Event (const Event &a)
- Event (std::string text)
- Event (const ublas::vector< double, ublas::unbounded_array< double > > &a)
- Event (double time, double duration, double status, double instrument, double key, double velocity, double phase, double pan, double depth, double height, double pitches)
- Event (const std::vector< double > &v)
- virtual ~Event ()
- void initialize ()
- bool isMidiEvent () const
- bool isNoteOn () const
- bool isNoteOff () const
- bool isNote () const
- bool isMatchingNoteOff (const Event &event) const
- bool isMatchingEvent (const Event &event) const
- void set (double time, double duration, double status, double instrument, double key, double velocity, double phase=0, double pan=0, double depth=0, double height=0, double pitches=4095)
- void setMidi (double time, char status, char key, char velocity)
- int getMidiStatus () const

- int getStatusNumber () const
- double getStatus () const
- void setStatus (double status)
- int getChannel () const
- double getInstrument () const
- void setInstrument (double instrument)
- double getTime () const
- void setTime (double time)
- double getDuration () const
- void setDuration (double duration)
- double getOffTime () const
- int getKeyNumber () const
- double getKey () const
- double getKey (double tonesPerOctave) const
- void setKey (double key)
- double getFrequency () const
- void setFrequency (double frequency)
- int getVelocityNumber () const
- double getVelocity () const
- void setVelocity (double velocity)
- double getGain () const
- double getPan () const
- void setPan (double pan)
- double getDepth () const
- void setDepth (double depth)
- double getHeight () const
- void setHeight (double height)
- double getPitches () const
- void setPitches (double pitches)
- double getAmplitude () const
- void setAmplitude (double amplitude)
- double getPhase () const
- void setPhase (double phase)
- double getLeftGain () const
- double getRightGain () const
- virtual void dump (std::ostream &stream)
- virtual std::string toString () const
- virtual std::string toCsoundIStatement (double tempering=12.0) const
- virtual std::string toCsoundIStatementHeld (int tag, double tempering=12.0) const
- virtual std::string toCsoundIStatementRelease (int tag, double tempering=12.0) const
- virtual void conformToPitchClassSet ()
- virtual void temper (double divisionsPerOctave)
- virtual std::string getProperty (std::string name)
- virtual void setProperty (std::string name, std::string value)
- virtual void removeProperty (std::string nameO)
- virtual void clearProperties ()
- virtual void createNoteOffEvent (Event &event) const
- Event & operator= (const Event &a)
- Event & operator= (const ublas::vector< double > &a)

Public Attributes

- std::map< std::string, std::string > properties

Static Public Attributes

- static int SORT_ORDER []
- static const char * labels []

6.45.2. Member Enumeration Documentation

6.45.2.1. anonymous enum

Enumeration values:

INDEFINITE

Definition at line 90 of file Event.hpp.

6.45.2.2. enum csound::Event::Dimensions

Enumeration values:

TIME

DURATION

STATUS

INSTRUMENT

KEY

VELOCITY

PHASE

PAN

DEPTH

HEIGHT

PITCHES

HOMOGENEITY

ELEMENT_COUNT

Definition at line 74 of file Event.hpp.

6.45.3. Constructor & Destructor Documentation

6.45.3.1. csound::Event::Event ()

6.45.3.2. csound::Event::Event (const Event & *a*)

6.45.3.3. csound::Event::Event (std::string *text*)

6.45.3.4. csound::Event::Event (const ublas::vector< double, ublas::unbounded_array< double > > & *a*)

6.45.3.5. csound::Event::Event (double *time*, double *duration*, double *status*, double *instrument*, double *key*, double *velocity*, double *phase*, double *pan*, double *depth*, double *height*, double *pitches*)

6.45.3.6. csound::Event::Event (const std::vector< double > & *v*)

6.45.3.7. virtual csound::Event::~Event () [virtual]

6.45.4. Member Function Documentation

6.45.4.1. virtual void csound::Event::clearProperties () [virtual]

6.45.4.2. virtual void csound::Event::conformToPitchClassSet () [virtual]

6.45.4.3. virtual void csound::Event::createNoteOffEvent (Event & *event*) const [virtual]

6.45.4.4. virtual void csound::Event::dump (std::ostream & *stream*) [virtual]

6.45.4.5. double csound::Event::getAmplitude () const

6.45.4.6. int csound::Event::getChannel () const

6.45.4.7. double csound::Event::getDepth () const

6.45.4.8. double csound::Event::getDuration () const

6.45.4.9. double csound::Event::getFrequency () const

6.45.4.10. double csound::Event::getGain () const

6.45.4.11. double csound::Event::getHeight () const

6.45.4.12. double csound::Event::getInstrument () const

6.45.4.13. double csound::Event::getKey (double *tonesPerOctave*) const

6.45.4.14. double csound::Event::getKey () const

6.45.4.15. int csound::Event::getKeyNumber () const

6.45.4.16. double csound::Event::getLeftGain () const

6.45.4.17. int csound::Event::getMidiStatus () const

6.45.4.18. double csound::Event::getOffTime () const

6.45.4.19. double csound::Event::getPan () const

6.45.4.20. double csound::Event::getPhase () const

6.45.4.21. double csound::Event::getPitches () const

6.45.5.2. std::map<std::string,std::string> csound::Event::properties

Definition at line 96 of file Event.hpp.

6.45.5.3. int csound::Event::SORT_ORDER[] [static]

Definition at line 94 of file Event.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Event.hpp

6.46. eventnode Struct Reference

`#include <csoundCore.h>`

Public Attributes

- eventnode * nxt
- unsigned long start_kcnt
- EVTBLK evt

6.46.1. Member Data Documentation

6.46.1.1. EVTBLK eventnode::evt

Definition at line 420 of file csoundCore.h.

6.46.1.2. struct eventnode* eventnode::nxt

Definition at line 418 of file csoundCore.h.

6.46.1.3. unsigned long eventnode::start_kcnt

Definition at line 419 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.47. csound::Exception Class Reference

`#include <Exception.hpp>`

6.47.1. Detailed Description

Base class for C++ exceptions in the Silence system.

Definition at line 38 of file Exception.hpp.

Public Member Functions

- Exception (std::string message)
- virtual ~Exception ()
- std::string getMessage () const

Private Attributes

- std::string message

6.47.2. Constructor & Destructor Documentation

6.47.2.1. csound::Exception::Exception (std::string *message*)

6.47.2.2. virtual csound::Exception::~Exception () [virtual]

6.47.3. Member Function Documentation

6.47.3.1. std::string csound::Exception::getMessage () const

6.47.4. Member Data Documentation

6.47.4.1. std::string csound::Exception::message [private]

Definition at line 40 of file Exception.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Exception.hpp

6.48. Loris::Exception Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::Exception::

6.48.1. Detailed Description

Exception is a generic exception class for reporting exceptional circumstances in Loris. Exception is derived from std:exception, and is the base for a hierarchy of derived exception classes in Loris.

Definition at line 51 of file Exception.h.

Public Member Functions

- Exception (const std::string &str, const std::string &where="")
- virtual ~Exception (void) throw ()
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.48.2. Constructor & Destructor Documentation

6.48.2.1. Loris::Exception::Exception (const std::string & *str*, const std::string & *where* = "")

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

6.48.2.2. virtual Loris::Exception::~Exception (void) throw () [inline, virtual]

Destroy this Exception.

Definition at line 69 of file Exception.h.

6.48.3. Member Function Documentation

6.48.3.1. Exception& Loris::Exception::append (const std::string & *str*)

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.48.3.2. const std::string& Loris::Exception::str (void) const [inline]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References _sbuf.

6.48.3.3. const char* Loris::Exception::what (void) const throw () [inline]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References _sbuf.

6.48.4. Member Data Documentation

6.48.4.1. std::string Loris::Exception::_sbuf [protected]

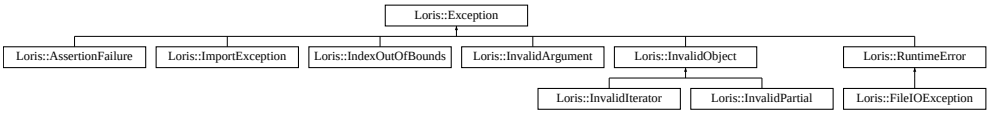
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by str(), and what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::AssertionFailure
Loris::ImportException
Loris::IndexOutOfBounds
Loris::InvalidArgument
Loris::InvalidObject
Loris::RuntimeError
Loris::InvalidIterator
Loris::InvalidPartial
Loris::FileIOException

6.49. fdch Struct Reference

`#include <csoundCore.h>`

Public Attributes

- fdch * nxtchp
- void * fd

6.49.1. Member Data Documentation

6.49.1.1. void* fdch::fd

Definition at line 220 of file csoundCore.h.

6.49.1.2. struct fdch* fdch::nxtchp

Definition at line 219 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.50. FGDATA Struct Reference

`#include <csoundCore.h>`

Public Attributes

- EVTBLK e
- double tpdlen
- int fno
- int guardreq
- int fterrcnt
- long flen
- long flenp1
- long lenmask

6.50.1. Member Data Documentation

6.50.1.1. EVTBLK FGDATA::e

Definition at line 424 of file csoundCore.h.

6.50.1.2. long FGDATA::flen

Definition at line 427 of file csoundCore.h.

6.50.1.3. long FGDATA::flenp1

Definition at line 427 of file csoundCore.h.

6.50.1.4. int FGDATA::fno

Definition at line 426 of file csoundCore.h.

6.50.1.5. int FGDATA::fterrcnt

Definition at line 426 of file csoundCore.h.

6.50.1.6. int FGDATA::guardreq

Definition at line 426 of file csoundCore.h.

6.50.1.7. long FGDATA::lenmask

Definition at line 427 of file csoundCore.h.

6.50.1.8. double FGDATA::tpdlen

Definition at line 425 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.51. Loris::FileIOException Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::FileIOException::

Public Member Functions

- FileIOException (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.51.1. Constructor & Destructor Documentation

6.51.1.1. Loris::FileIOException::FileIOException (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 278 of file Exception.h.

6.51.2. Member Function Documentation

6.51.2.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.51.2.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:

a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.51.2.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.51.3. Member Data Documentation

6.51.3.1. std::string Loris::Exception::_sbuf [protected, inherited]

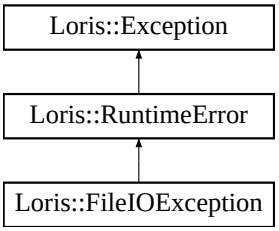
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::RuntimeError
Loris::FileIOException

6.52. Filter Class Reference

`#include <Filter.h>`

6.52.1. Detailed Description

This class implements a generic structure which can be used to create a wide range of filters. It can function independently or be subclassed to provide more specific controls based on a particular filter type.

In particular, this class implements the standard difference equation:

a[0]*y[n] = b[0]*x[n] + ... + b[nb]*x[n-nb] - a[1]*y[n-1] - ... - a[na]*y[n-na]

If a[0] is not equal to 1, the filter coeffcients are normalized by a[0].

The *gain* parameter is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0. This structure results in one extra multiply per computed sample, but allows easy control of the overall filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2004.

Definition at line 37 of file Filter.h.

Public Member Functions

- Filter (void)
- Filter (std::vector< StkFloat > &bCoefficients, std::vector< StkFloat > &aCoefficients)
- virtual ~Filter (void)
- void clear (void)
- void setCoefficients (std::vector< StkFloat > &bCoefficients, std::vector< StkFloat > &a-Coefficients)
- void setNumerator (std::vector< StkFloat > &bCoefficients)
- void setDenominator (std::vector< StkFloat > &aCoefficients)
- virtual void setGain (StkFloat gain)
- virtual StkFloat getGain (void) const
- virtual StkFloat lastOut (void) const
- virtual StkFloat tick (StkFloat sample)
- virtual StkFloat * tick (StkFloat *vector, unsigned int vectorSize)
- virtual StkFrames & tick (StkFrames &frames, unsigned int channel=1)

Protected Attributes

- StkFloat gain_
- std::vector< StkFloat > b_
- std::vector< StkFloat > a_
- std::vector< StkFloat > outputs_
- std::vector< StkFloat > inputs_

6.52.2. Constructor & Destructor Documentation

6.52.2.1. Filter::Filter (void)

Default constructor creates a zero-order pass-through "filter".

6.52.2.2. Filter::Filter (std::vector< StkFloat > & *bCoefficients*, std::vector< StkFloat > & *aCoefficients*)

Overloaded constructor which takes filter coefficients.

An StkError can be thrown if either of the coefficient vector sizes is zero, or if the a[0] coefficient is equal to zero.

6.52.2.3. virtual Filter::~Filter (void) [virtual]

Class destructor.

6.52.3. Member Function Documentation

6.52.3.1. void Filter::clear (void)

Sets all internal states of the filter to zero.

6.52.3.2. virtual StkFloat Filter::getGain (void) const [virtual]

Return the current filter gain.

6.52.3.3. virtual StkFloat Filter::lastOut (void) const [virtual]

Return the last computed output value.

6.52.3.4. void Filter::setCoefficients (std::vector< StkFloat > & *bCoefficients*, std::vector< StkFloat > & *aCoefficients*)

Set filter coefficients.

An StkError can be thrown if either of the coefficient vector sizes is zero, or if the a[0] coefficient is equal to zero. If a[0] is not equal to 1, the filter coeffcients are normalized by a[0]. The internal state of the filter is cleared.

6.52.3.5. void Filter::setDenominator (std::vector< StkFloat > & *aCoefficients*)

Set denominator coefficients.

An StkError can be thrown if the coefficient vector is empty or if the a[0] coefficient is equal to zero. Previously set numerator coefficients are unaffected unless a[0] is not equal to 1, in which case all coeffcients are normalized by a[0]. Note that the default constructor sets the single numerator coefficient b[0] to 1.0. The internal state of the filter is cleared.

6.52.3.6. virtual void Filter::setGain (StkFloat *gain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

6.52.3.7. void Filter::setNumerator (std::vector< StkFloat > & *bCoefficients*)

Set numerator coefficients.

An StkError can be thrown if coefficient vector is empty. Any previously set denominator coefficients are left unaffected. Note that the default constructor sets the single denominator coefficient a[0] to 1.0. The internal state of the filter is cleared.

6.52.3.8. virtual StkFrames& Filter::tick (StkFrames & *frames*, unsigned int *channel* = 1) [virtual]

Take a channel of the StkFrames object as inputs to the filter and replace with corresponding outputs.

The `channel` argument should be one or greater (the first channel is specified by 1). An StkError will be thrown if the `channel` argument is zero or it is greater than the number of channels in the StkFrames object.

6.52.3.9. virtual StkFloat* Filter::tick (StkFloat * *vector*, unsigned int *vectorSize*) [virtual]

Input *vectorSize* samples to the filter and return an equal number of outputs in *vector*.

6.52.3.10. virtual StkFloat Filter::tick (StkFloat *sample*) [virtual]

Input one sample to the filter and return one output.

6.52.4. Member Data Documentation

6.52.4.1. std::vector<StkFloat> Filter::a_ [protected]

Definition at line 117 of file Filter.h.

6.52.4.2. std::vector<StkFloat> Filter::b_ [protected]

Definition at line 116 of file Filter.h.

6.52.4.3. StkFloat Filter::gain_ [protected]

Definition at line 115 of file Filter.h.

6.52.4.4. std::vector<StkFloat> Filter::inputs_ [protected]

Definition at line 119 of file Filter.h.

6.52.4.5. std::vector<StkFloat> Filter::outputs_ [protected]

Definition at line 118 of file Filter.h.

The documentation for this class was generated from the following file:

- Opcodes/stk/include/Filter.h

6.53. FILTER Struct Reference

`#include <filter.h>`

Public Attributes

- OPDS h
- MYFLT * out
- MYFLT * in
- MYFLT * nb
- MYFLT * na
- MYFLT * coeffs [MAXPOLES+MAXZEROS+1]
- MYFLT * d1
- MYFLT * d2
- int numa
- int numb
- double dcoeffs [MAXPOLES+MAXZEROS+1]
- AUXCH delay
- double * currPos
- int ndelay

6.53.1. Member Data Documentation

6.53.1.1. MYFLT* FILTER::coeffs[MAXPOLES+MAXZEROS+1]

Definition at line 46 of file filter.h.

6.53.1.2. double* FILTER::currPos

Definition at line 54 of file filter.h.

6.53.1.3. MYFLT* FILTER::d1

Definition at line 47 of file filter.h.

6.53.1.4. MYFLT * FILTER::d2

Definition at line 47 of file filter.h.

6.53.1.5. double FILTER::dcoeffs[MAXPOLES+MAXZEROS+1]

Definition at line 52 of file filter.h.

6.53.1.6. AUXCH FILTER::delay

Definition at line 53 of file filter.h.

6.53.1.7. OPDS FILTER::h

Definition at line 41 of file filter.h.

6.53.1.8. MYFLT* FILTER::in

Definition at line 44 of file filter.h.

6.53.1.9. MYFLT * FILTER::na

Definition at line 45 of file filter.h.

6.53.1.10. MYFLT* FILTER::nb

Definition at line 45 of file filter.h.

6.53.1.11. int FILTER::ndelay

Definition at line 55 of file filter.h.

6.53.1.12. int FILTER::numa

Definition at line 49 of file filter.h.

6.53.1.13. int FILTER::numb

Definition at line 50 of file filter.h.

6.53.1.14. MYFLT* FILTER::out

Definition at line 43 of file filter.h.

The documentation for this struct was generated from the following file:

- Opcodes/filter.h

6.54. Loris::Filter Class Reference

`#include <Filter.h>`

Public Member Functions

- Filter (void)
- template<typename IterT1, typename IterT2> Filter (IterT1 ma_begin, IterT1 ma_end, IterT2 ar_begin, IterT2 ar_end, double gain=1.)
- Filter (const Filter &other)
- Filter & operator= (const Filter &rhs)
- double sample (double input)
- double operator() (double input)
- void clear (void)

Public Attributes

- std::deque< double > _delayline
- std::vector< double > _maCoefs
- std::vector< double > _arCoefs
- double _gain

6.54.1. Constructor & Destructor Documentation

6.54.1.1. Loris::Filter::Filter (void)

6.54.1.2. template<typename IterT1, typename IterT2> Loris::Filter::Filter (IterT1 *ma_begin*, IterT1 *ma_end*, IterT2 *ar_begin*, IterT2 *ar_end*, double *gain* = 1.)

Definition at line 116 of file Filter.h.

References _arCoefs, _delayline, _maCoefs, and Throw.

6.54.1.3. Loris::Filter::Filter (const Filter & *other*)

6.54.2. Member Function Documentation

6.54.2.1. void Loris::Filter::clear (void)

6.54.2.2. double Loris::Filter::operator() (double *input*) [inline]

Definition at line 90 of file Filter.h.

References sample().

6.54.2.3. Filter& Loris::Filter::operator= (const Filter & *rhs*)

6.54.2.4. double Loris::Filter::sample (double *input*)

Referenced by operator()().

6.54.3. Member Data Documentation

6.54.3.1. std::vector< double > Loris::Filter::_arCoefs

Definition at line 100 of file Filter.h.

Referenced by Filter().

6.54.3.2. std::deque< double > Loris::Filter::_delayline

Definition at line 97 of file Filter.h.

Referenced by Filter().

6.54.3.3. double Loris::Filter::_gain

Definition at line 103 of file Filter.h.

6.54.3.4. std::vector< double > Loris::Filter::_maCoefs

Definition at line 100 of file Filter.h.

Referenced by Filter().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Filter.h

6.55. FLUID_CC Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * iEngineNumber
- MYFLT * iChannelNumber
- MYFLT * iControllerNumber
- MYFLT * kVal
- int priorMidiValue

6.55.1. Member Data Documentation

6.55.1.1. OPDS FLUID_CC::h

Definition at line 30 of file fluidOpcodes.hpp.

6.55.1.2. MYFLT * FLUID_CC::iChannelNumber

Definition at line 32 of file fluidOpcodes.hpp.

6.55.1.3. MYFLT * FLUID_CC::iControllerNumber

Definition at line 32 of file fluidOpcodes.hpp.

6.55.1.4. MYFLT* FLUID_CC::iEngineNumber

Definition at line 32 of file fluidOpcodes.hpp.

6.55.1.5. MYFLT * FLUID_CC::kVal

Definition at line 32 of file fluidOpcodes.hpp.

6.55.1.6. int FLUID_CC::priorMidiValue

Definition at line 33 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.56. FLUID_NOTE Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * iEngineNumber
- MYFLT * iChannelNumber
- MYFLT * iMidiKeyNumber
- MYFLT * iVelocity
- int initDone
- bool released

6.56.1. Member Data Documentation

6.56.1.1. OPDS FLUID_NOTE::h

Definition at line 37 of file fluidOpcodes.hpp.

6.56.1.2. MYFLT * FLUID_NOTE::iChannelNumber

Definition at line 39 of file fluidOpcodes.hpp.

6.56.1.3. MYFLT* FLUID_NOTE::iEngineNumber

Definition at line 39 of file fluidOpcodes.hpp.

6.56.1.4. MYFLT * FLUID_NOTE::iMidiKeyNumber

Definition at line 39 of file fluidOpcodes.hpp.

6.56.1.5. int FLUID_NOTE::initDone

Definition at line 40 of file fluidOpcodes.hpp.

6.56.1.6. MYFLT * FLUID_NOTE::iVelocity

Definition at line 39 of file fluidOpcodes.hpp.

6.56.1.7. bool FLUID_NOTE::released

Definition at line 41 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.57. FLUID_PROGRAM_SELECT Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * iEngineNumber
- MYFLT * iChannelNumber
- MYFLT * iInstrumentNumber
- MYFLT * iBankNumber
- MYFLT * iPresetNumber

6.57.1. Member Data Documentation

6.57.1.1. OPDS FLUID_PROGRAM_SELECT::h

Definition at line 23 of file fluidOpcodes.hpp.

6.57.1.2. MYFLT * FLUID_PROGRAM_SELECT::iBankNumber

Definition at line 25 of file fluidOpcodes.hpp.

6.57.1.3. MYFLT * FLUID_PROGRAM_SELECT::iChannelNumber

Definition at line 25 of file fluidOpcodes.hpp.

6.57.1.4. MYFLT* FLUID_PROGRAM_SELECT::iEngineNumber

Definition at line 25 of file fluidOpcodes.hpp.

6.57.1.5. MYFLT * FLUID_PROGRAM_SELECT::iInstrumentNumber

Definition at line 25 of file fluidOpcodes.hpp.

6.57.1.6. MYFLT* FLUID_PROGRAM_SELECT::iPresetNumber

Definition at line 26 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.58. FLUIDALLOUT Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * aLeftOut
- MYFLT * aRightOut
- int blockSize

6.58.1. Member Data Documentation

6.58.1.1. MYFLT* FLUIDALLOUT::aLeftOut

Definition at line 53 of file fluidOpcodes.hpp.

6.58.1.2. MYFLT * FLUIDALLOUT::aRightOut

Definition at line 53 of file fluidOpcodes.hpp.

6.58.1.3. int FLUIDALLOUT::blockSize

Definition at line 54 of file fluidOpcodes.hpp.

6.58.1.4. OPDS FLUIDALLOUT::h

Definition at line 52 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.59. FLUIDENGINE Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * iEngineNum

6.59.1. Member Data Documentation

6.59.1.1. OPDS FLUIDENGINE::h

Definition at line 8 of file fluidOpcodes.hpp.

6.59.1.2. MYFLT* FLUIDENGINE::iEngineNum

Definition at line 10 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.60. FLUIDLOAD Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * iInstrumentNumber
- MYFLT * filename
- MYFLT * iEngineNum
- MYFLT * iListPresets

6.60.1. Member Data Documentation

6.60.1.1. MYFLT* FLUIDLOAD::filename

Definition at line 19 of file fluidOpcodes.hpp.

6.60.1.2. OPDS FLUIDLOAD::h

Definition at line 15 of file fluidOpcodes.hpp.

6.60.1.3. MYFLT * FLUIDLOAD::iEngineNum

Definition at line 19 of file fluidOpcodes.hpp.

6.60.1.4. MYFLT* FLUIDLOAD::iInstrumentNumber

Definition at line 17 of file fluidOpcodes.hpp.

6.60.1.5. MYFLT * FLUIDLOAD::iListPresets

Definition at line 19 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.61. FLUIDOUT Struct Reference

`#include <fluidOpcodes.hpp>`

Public Attributes

- OPDS h
- MYFLT * aLeftOut
- MYFLT * aRightOut
- MYFLT * iEngineNum
- int blockSize

6.61.1. Member Data Documentation

6.61.1.1. MYFLT* FLUIDOUT::aLeftOut

Definition at line 46 of file fluidOpcodes.hpp.

6.61.1.2. MYFLT * FLUIDOUT::aRightOut

Definition at line 46 of file fluidOpcodes.hpp.

6.61.1.3. int FLUIDOUT::blockSize

Definition at line 48 of file fluidOpcodes.hpp.

6.61.1.4. OPDS FLUIDOUT::h

Definition at line 45 of file fluidOpcodes.hpp.

6.61.1.5. MYFLT* FLUIDOUT::iEngineNum

Definition at line 47 of file fluidOpcodes.hpp.

The documentation for this struct was generated from the following file:

- Opcodes/fluidOpcodes/fluidOpcodes.hpp

6.62. Loris::FourierTransform Class Reference

`#include <FourierTransform.h>`

6.62.1. Detailed Description

FourierTransform provides a simplified interface to the FFTW library (www.fftw.org). Loris uses the FFTW library to perform efficient Fourier transforms of arbitrary length. Clients store and access the in-place transform data as a sequence of std::complex< double >. Samples are stored in the FourierTransform instance using subscript or iterator access, the transform is computed by the transform member, and the transformed samples replace the input samples, and are accessed by subscript or iterator. FourierTransform computes a complex transform, so it can be used to invert a transform of real samples as well. Uses the standard library complex class, which implements arithmetic operations. Does not use FFTW "wisdom" to speed up transform computation.

Definition at line 60 of file FourierTransform.h.

Public Types

- typedef std::vector< std::complex< double > >::size_type size_type
- typedef std::vector< std::complex< double > >::iterator iterator
- typedef std::vector< std::complex< double > >::const_iterator const_iterator

Public Member Functions

- FourierTransform (size_type len)
- FourierTransform (const FourierTransform &rhs)
- ~FourierTransform (void)
- FourierTransform & operator= (const FourierTransform &rhs)
- std::complex< double > & operator[] (size_type index)
- const std::complex< double > & operator[] (size_type index) const
- iterator begin (void)
- iterator end (void)
- const_iterator begin (void) const
- const_iterator end (void) const
- void transform (void)
- size_type size (void) const

Private Attributes

- std::vector< std::complex< double > > _buffer
- FTimpl * _impl

6.62.2. Member Typedef Documentation

6.62.2.1. typedef std::vector< std::complex< double > >::const_iterator Loris::FourierTransform::const_iterator

The type of a const iterator of (complex) transform samples.

Definition at line 73 of file FourierTransform.h.

6.62.2.2. typedef std::vector< std::complex< double > >::iterator Loris::FourierTransform::iterator

The type of a non-const iterator of (complex) transform samples.

Definition at line 70 of file FourierTransform.h.

6.62.2.3. typedef std::vector< std::complex< double > >::size_type Loris::FourierTransform::size_type

An unsigned integral type large enough to represent the length of any transform.

Definition at line 67 of file FourierTransform.h.

6.62.3. Constructor & Destructor Documentation

6.62.3.1. Loris::FourierTransform::FourierTransform (size_type *len*)

Exceptions:

RuntimeError if the necessary buffers cannot be allocated, or there is an error configuring FFTW.

6.62.3.2. Loris::FourierTransform::FourierTransform (const FourierTransform & *rhs*)

Initialize a new FourierTransform that is a copy of another, having the same size and the same buffer contents.

Parameters:

rhs is the instance to copy

Exceptions:

RuntimeError if the necessary buffers cannot be allocated, or there is an error configuring FFTW.

6.62.3.3. Loris::FourierTransform::~FourierTransform (void)

Free the resources associated with this FourierTransform.

6.62.4. Member Function Documentation

6.62.4.1. const_iterator Loris::FourierTransform::begin (void) const [inline]

Return a const iterator refering to the beginning of the sequence of complex samples in the transform buffer.

Returns:

a const iterator refering to the first position in the transform buffer.

Definition at line 165 of file FourierTransform.h.

References _buffer, and begin().

6.62.4.2. iterator Loris::FourierTransform::begin (void) [inline]

Return an iterator refering to the beginning of the sequence of complex samples in the transform buffer.

Returns:
a non-const iterator refering to the first position in the transform buffer.

Definition at line 145 of file FourierTransform.h.

References _buffer.

Referenced by begin().

6.62.4.3. const_iterator Loris::FourierTransform::end (void) const [inline]

complex samples in the transform buffer.

Returns:
a const iterator refering to one past the last position in the transform buffer.

Definition at line 175 of file FourierTransform.h.

References _buffer, and end().

6.62.4.4. iterator Loris::FourierTransform::end (void) [inline]

complex samples in the transform buffer.

Returns:
a non-const iterator refering to one past the last position in the transform buffer.

Definition at line 155 of file FourierTransform.h.

References _buffer.

Referenced by end().

6.62.4.5. FourierTransform& Loris::FourierTransform::operator= (const FourierTransform & *rhs*)

Make this FourierTransform a copy of another, having the same size and buffer contents.

Parameters:
rhs is the instance to copy

Returns:
a refernce to this instance

Exceptions:
RuntimeError if the necessary buffers cannot be allocated, or there is an error configuring FFTW.

6.62.4.6.]

const std::complex< double >& Loris::FourierTransform::operator[] (size_type *index*) const `[inline]`

Access (read-only) a transform sample by index. Use this member to fill the transform buffer before computing the transform, and to access the samples after computing the transform. (inlined for speed)

Parameters:

index is the index or rank of the complex transform sample to access. Zero is the first position in the buffer.

Returns:

const reference to the std::complex< double > at the specified position in the buffer.

Definition at line 135 of file FourierTransform.h.

References _buffer.

6.62.4.7.]

std::complex< double >& Loris::FourierTransform::operator[] (size_type *index*) `[inline]`

Access (read/write) a transform sample by index. Use this member to fill the transform buffer before computing the transform, and to access the samples after computing the transform. (inlined for speed)

Parameters:

index is the index or rank of the complex transform sample to access. Zero is the first position in the buffer.

Returns:

non-const reference to the std::complex< double > at the specified position in the buffer.

Definition at line 120 of file FourierTransform.h.

References _buffer.

6.62.4.8. size_type Loris::FourierTransform::size (void) const

Return the length of the transform (in samples).

Returns:

the length of the transform in samples.

Referenced by Loris::ReassignedSpectrum::size().

6.62.4.9. void Loris::FourierTransform::transform (void)

Compute the Fourier transform of the samples stored in the transform buffer. The samples stored in the transform buffer (accessed by index or by iterator) are replaced by the transformed samples, in-place.

6.62.5. Member Data Documentation

6.62.5.1. std::vector< std::complex< double > > Loris::FourierTransform::_buffer [private]

buffer containing the complex transform input before computing the transform, and the complex transform output after computing the transform

Definition at line 201 of file FourierTransform.h.

Referenced by begin(), end(), and operator[]().

6.62.5.2. FTimpl* Loris::FourierTransform::_impl [private]

Definition at line 205 of file FourierTransform.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/FourierTransform.h

6.63. Loris::BreakpointUtils::frequency_between Struct Reference

`#include <BreakpointUtils.h>`

Public Member Functions

- bool operator() (const Breakpoint &b) const
- frequency_between (double x, double y)

Private Attributes

- double _fmin
- double _fmax

6.63.1. Constructor & Destructor Documentation

6.63.1.1. Loris::BreakpointUtils::frequency_between::frequency_between (double *x*, double *y*) [inline]

Definition at line 118 of file BreakpointUtils.h.

6.63.2. Member Function Documentation

6.63.2.1. bool Loris::BreakpointUtils::frequency_between::operator() (const Breakpoint & *b*) const [inline]

Definition at line 111 of file BreakpointUtils.h.

6.63.3. Member Data Documentation

6.63.3.1. double Loris::BreakpointUtils::frequency_between::_fmax [private]

Definition at line 124 of file BreakpointUtils.h.

6.63.3.2. double Loris::BreakpointUtils::frequency_between::_fmin [private]

Definition at line 124 of file BreakpointUtils.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/BreakpointUtils.h

6.64. Loris::FrequencyReference Class Reference

`#include <FrequencyReference.h>`

Inheritance diagram for Loris::FrequencyReference::

Public Member Functions

- FrequencyReference (PartialList::const_iterator begin, PartialList::const_iterator end, double minFreq, double maxFreq, long numSamps)
- FrequencyReference (PartialList::const_iterator begin, PartialList::const_iterator end, double minFreq, double maxFreq)
- FrequencyReference (const FrequencyReference &other)
- FrequencyReference & operator= (const FrequencyReference &other)
- ~FrequencyReference ()
- BreakpointEnvelope envelope (void) const
- virtual FrequencyReference * clone (void) const
- virtual double valueAt (double x) const

Private Attributes

- std::auto_ptr< BreakpointEnvelope > _env

6.64.1. Constructor & Destructor Documentation

6.64.1.1. Loris::FrequencyReference::FrequencyReference (PartialList::const_iterator *begin*, PartialList::const_iterator *end*, double *minFreq*, double *maxFreq*, long *numSamps*)

6.64.1.2. Loris::FrequencyReference::FrequencyReference (PartialList::const_iterator *begin*, PartialList::const_iterator *end*, double *minFreq*, double *maxFreq*)

6.64.1.3. Loris::FrequencyReference::FrequencyReference (const FrequencyReference & *other*)

6.64.1.4. Loris::FrequencyReference::~FrequencyReference ()

6.64.2. Member Function Documentation

6.64.2.1. virtual FrequencyReference* Loris::FrequencyReference::clone (void) const [virtual]

Return an exact copy of this Envelope (following the Prototype pattern).

Implements Loris::Envelope.

6.64.2.2. BreakpointEnvelope Loris::FrequencyReference::envelope (void) const

6.64.2.3. FrequencyReference& Loris::FrequencyReference::operator= (const FrequencyReference & *other*)

6.64.2.4. virtual double Loris::FrequencyReference::valueAt (double *x*) const [virtual]

Return the value of this Envelope at the specified time.

Implements Loris::Envelope.

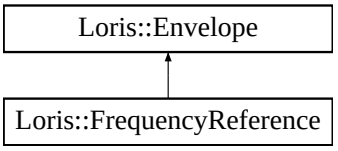
6.64.3. Member Data Documentation

6.64.3.1. std::auto_ptr< BreakpointEnvelope > Loris::FrequencyReference::_env [private]

Definition at line 60 of file FrequencyReference.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/FrequencyReference.h


Loris::Envelope
Loris::FrequencyReference

6.65. Loris::PartialUtils::FrequencyScaler Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::FrequencyScaler::

Public Member Functions

- FrequencyScaler (double x)
- FrequencyScaler (const Envelope &e)
- void operator() (Partial &p) const

Protected Attributes

- Envelope * env

6.65.1. Constructor & Destructor Documentation

6.65.1.1. Loris::PartialUtils::FrequencyScaler::FrequencyScaler (double *x*) [inline]

Definition at line 225 of file PartialUtils.h.

6.65.1.2. Loris::PartialUtils::FrequencyScaler::FrequencyScaler (const Envelope & *e*) [inline]

Definition at line 226 of file PartialUtils.h.

6.65.2. Member Function Documentation

6.65.2.1. void Loris::PartialUtils::FrequencyScaler::operator() (Partial & *p*) const [virtual]

Function call operator: apply a mutation factor to the specified Partial. Derived classes must implement this member.

Implements Loris::PartialUtils::PartialMutator.

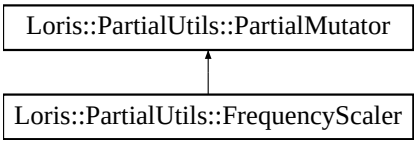
6.65.3. Member Data Documentation

6.65.3.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected, inherited]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::FrequencyScaler

6.66. FUNC Struct Reference

`#include <csoundCore.h>`

Public Attributes

- long flen
- long lenmask
- long lobits
- long lomask
- MYFLT lodiv
- MYFLT cvtbas
- MYFLT cpscvt
- short loopmode1
- short loopmode2
- long begin1
- long end1
- long begin2
- long end2
- long soundend
- long flenfrms
- long nchanls
- long fno
- GEN01ARGS gen01args
- MYFLT ftable [1]

6.66.1. Member Data Documentation

6.66.1.1. long FUNC::begin1

Definition at line 390 of file csoundCore.h.

6.66.1.2. long FUNC::begin2

Definition at line 391 of file csoundCore.h.

6.66.1.3. MYFLT FUNC::cpscvt

Definition at line 387 of file csoundCore.h.

6.66.1.4. MYFLT FUNC::cvtbas

Definition at line 387 of file csoundCore.h.

6.66.1.5. long FUNC::end1

Definition at line 390 of file csoundCore.h.

6.66.1.6. long FUNC::end2

Definition at line 391 of file csoundCore.h.

6.66.1.7. long FUNC::flen

Definition at line 382 of file csoundCore.h.

6.66.1.8. long FUNC::flenfrms

Definition at line 392 of file csoundCore.h.

6.66.1.9. long FUNC::fno

Definition at line 394 of file csoundCore.h.

6.66.1.10. MYFLT FUNC::ftable[1]

Definition at line 396 of file csoundCore.h.

6.66.1.11. GEN01ARGS FUNC::gen01args

Definition at line 395 of file csoundCore.h.

6.66.1.12. long FUNC::lenmask

Definition at line 383 of file csoundCore.h.

6.66.1.13. long FUNC::lobits

Definition at line 384 of file csoundCore.h.

6.66.1.14. MYFLT FUNC::lodiv

Definition at line 386 of file csoundCore.h.

6.66.1.15. long FUNC::lomask

Definition at line 385 of file csoundCore.h.

6.66.1.16. short FUNC::loopmode1

Definition at line 388 of file csoundCore.h.

6.66.1.17. short FUNC::loopmode2

Definition at line 389 of file csoundCore.h.

6.66.1.18. long FUNC::nchanls

Definition at line 393 of file csoundCore.h.

6.66.1.19. long FUNC::soundend

Definition at line 392 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.67. GEN01ARGS Struct Reference

`#include <csoundCore.h>`

Public Attributes

- MYFLT gen01
- MYFLT ifilno
- MYFLT iskptim
- MYFLT iformat
- MYFLT channel
- MYFLT sample_rate
- char strarg [SSTRSIZ]

6.67.1. Member Data Documentation

6.67.1.1. MYFLT GEN01ARGS::channel

Definition at line 376 of file csoundCore.h.

6.67.1.2. MYFLT GEN01ARGS::gen01

Definition at line 372 of file csoundCore.h.

6.67.1.3. MYFLT GEN01ARGS::ifilno

Definition at line 373 of file csoundCore.h.

6.67.1.4. MYFLT GEN01ARGS::iformat

Definition at line 375 of file csoundCore.h.

6.67.1.5. MYFLT GEN01ARGS::iskptim

Definition at line 374 of file csoundCore.h.

6.67.1.6. MYFLT GEN01ARGS::sample_rate

Definition at line 377 of file csoundCore.h.

6.67.1.7. char GEN01ARGS::strarg[SSTRSIZ]

Definition at line 378 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.68. Loris::BreakpointUtils::greater_amplitude Struct Reference

`#include <BreakpointUtils.h>`

Public Member Functions

- bool operator() (const Breakpoint &lhs, const Breakpoint &rhs) const

6.68.1. Member Function Documentation

6.68.1.1. bool Loris::BreakpointUtils::greater_amplitude::operator() (const Breakpoint & *lhs*, const Breakpoint & *rhs*) const [inline]

Definition at line 146 of file BreakpointUtils.h.

References Loris::Breakpoint::amplitude().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/BreakpointUtils.h

6.69. csound::Hocket Class Reference

`#include <Hocket.hpp>`

Inheritance diagram for csound::Hocket::

6.69.1. Detailed Description

Simplifies constructing complex hocketted scores.

Definition at line 40 of file Hocket.hpp.

Public Member Functions

- Hocket ()
- virtual ~Hocket ()
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &globalCoordinates, Score &score)
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- int modulus
- int startingIndex
- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- Score score
- ublas::matrix< double > localCoordinates

6.69.2. Constructor & Destructor Documentation

6.69.2.1. csound::Hocket::Hocket ()

6.69.2.2. virtual csound::Hocket::~Hocket () [virtual]

6.69.3. Member Function Documentation

6.69.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.69.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.69.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.69.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.69.3.5. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.69.3.6. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.69.3.7. virtual void csound::Hocket::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::ScoreNode.

6.69.3.8. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.69.3.9. virtual ublas::matrix<double> csound::Hocket::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented from csound::Node.

6.69.4. Member Data Documentation

6.69.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.69.4.2. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.69.4.3. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.69.4.4. int csound::Hocket::modulus

Definition at line 44 of file Hocket.hpp.

6.69.4.5. Score csound::ScoreNode::score [protected, inherited]

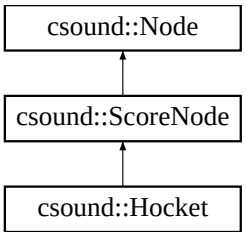
Definition at line 47 of file ScoreNode.hpp.

6.69.4.6. int csound::Hocket::startingIndex

Definition at line 45 of file Hocket.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Hocket.hpp


csound::Node
csound::ScoreNode
csound::Hocket

6.70. csound::ImageToScore Class Reference

`#include <ImageToScore.hpp>`

Inheritance diagram for csound::ImageToScore::

6.70.1. Detailed Description

Translates images in various RGB formats to scores. Hue is mapped to instrument, value is mapped to loudness.

Definition at line 43 of file ImageToScore.hpp.

Public Member Functions

- ImageToScore (void)
- virtual ~ImageToScore (void)
- virtual void setImageFilename (std::string imageFilename)
- virtual std::string getImageFilename () const
- virtual void setMaximumVoiceCount (size_t maximumVoiceCount)
- virtual size_t getMaximumVoiceCount () const
- virtual void setMinimumValue (double minimumValue)
- virtual double getMinimumValue () const
- virtual void generate ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Member Functions

- virtual void getPixel (size_t x, size_t y, double &hue, double &saturation, double &value) const
- virtual void translate (double x, double y, double hue, double value, Event &event) const

Static Protected Member Functions

- static void rgbToHsv (double r, double g, double b, double &h, double &s, double &v)

Protected Attributes

- std::string imageFilename
- Fl_Image * image
- size_t maximumVoiceCount
- double minimumValue
- Score score
- ublas::matrix< double > localCoordinates

6.70.2. Constructor & Destructor Documentation

6.70.2.1. csound::ImageToScore::ImageToScore (void)

6.70.2.2. virtual csound::ImageToScore::~ImageToScore (void) [virtual]

6.70.3. Member Function Documentation

6.70.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.70.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.70.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.70.3.4. virtual void csound::ImageToScore::generate () [virtual]

6.70.3.5. virtual std::string csound::ImageToScore::getImageFilename () const [virtual]

6.70.3.6. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.70.3.7. virtual size_t csound::ImageToScore::getMaximumVoiceCount () const [virtual]

6.70.3.8. virtual double csound::ImageToScore::getMinimumValue () const [virtual]

6.70.3.9. virtual void csound::ImageToScore::getPixel (size_t *x*, size_t *y*, double & *hue*, double & *saturation*, double & *value*) const [protected, virtual]

6.70.3.10. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.70.3.11. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.70.3.12. virtual void csound::ScoreNode::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual, inherited]

The default implementation does nothing.

Reimplemented from csound::Node.

Reimplemented in csound::Cell, csound::Hocket, csound::MCRM, and csound::Rescale.

6.70.3.13. static void csound::ImageToScore::rgbToHsv (double *r*, double *g*, double *b*, double & *h*, double & *s*, double & *v*) [static, protected]

6.70.3.14. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.70.3.15. virtual void csound::ImageToScore::setImageFilename (std::string *imageFilename*) [virtual]

6.70.3.16. virtual void csound::ImageToScore::setMaximumVoiceCount (size_t *maximumVoiceCount*) [virtual]

6.70.3.17. virtual void csound::ImageToScore::setMinimumValue (double *minimumValue*) [virtual]

6.70.3.18. virtual void csound::ImageToScore::translate (double *x*, double *y*, double *hue*, double *value*, Event & *event*) const [protected, virtual]

6.70.3.19. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.70.4. Member Data Documentation

6.70.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.70.4.2. Fl_Image* csound::ImageToScore::image [protected]

Definition at line 47 of file ImageToScore.hpp.

6.70.4.3. std::string csound::ImageToScore::imageFilename [protected]

Definition at line 46 of file ImageToScore.hpp.

6.70.4.4. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.70.4.5. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.70.4.6. size_t csound::ImageToScore::maximumVoiceCount [protected]

Definition at line 48 of file ImageToScore.hpp.

6.70.4.7. double csound::ImageToScore::minimumValue [protected]

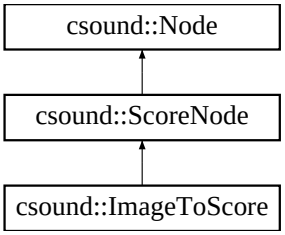
Definition at line 49 of file ImageToScore.hpp.

6.70.4.8. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/ImageToScore.hpp


csound::Node
csound::ScoreNode
csound::ImageToScore

6.71. Loris::ImportException Class Reference

`#include <ImportLemur.h>`

Inheritance diagram for Loris::ImportException::

Public Member Functions

- ImportException (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.71.1. Constructor & Destructor Documentation

6.71.1.1. Loris::ImportException::ImportException (const std::string & *str*, const std::string & *where* = "") [inline]

Definition at line 78 of file ImportLemur.h.

6.71.2. Member Function Documentation

6.71.2.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:
str is text to append to the exception description

Returns:
a reference to this Exception.

6.71.2.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.71.2.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.71.3. Member Data Documentation

6.71.3.1. std::string Loris::Exception::_sbuf [protected, inherited]

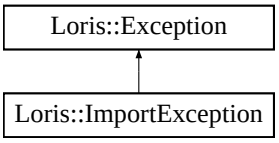
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/ImportLemur.h


Loris::Exception
Loris::ImportException

6.72. Loris::ImportLemur Class Reference

`#include <ImportLemur.h>`

Public Member Functions

- ImportLemur (const std::string &fname, double bweCutoff=1000)
- PartialList & partials (void)
- const PartialList & partials (void) const

Private Member Functions

- ImportLemur (const ImportLemur &other)
- ImportLemur & operator= (const ImportLemur &rhs)

Private Attributes

- PartialList _partials

6.72.1. Constructor & Destructor Documentation

6.72.1.1. Loris::ImportLemur::ImportLemur (const std::string & *fname*, double *bweCutoff* = 1000)

6.72.1.2. Loris::ImportLemur::ImportLemur (const ImportLemur & *other*) [private]

6.72.2. Member Function Documentation

6.72.2.1. ImportLemur& Loris::ImportLemur::operator= (const ImportLemur & *rhs*) [private]

6.72.2.2. const PartialList& Loris::ImportLemur::partials (void) const [inline]

Definition at line 60 of file ImportLemur.h.

References _partials.

6.72.2.3. PartialList& Loris::ImportLemur::partials (void) [inline]

Definition at line 59 of file ImportLemur.h.

References _partials.

6.72.3. Member Data Documentation

6.72.3.1. PartialList Loris::ImportLemur::_partials [private]

Definition at line 50 of file ImportLemur.h.

Referenced by partials().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/ImportLemur.h

6.73. Loris::IndexOutOfBounds Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::IndexOutOfBounds::

6.73.1. Detailed Description

Class of exceptions thrown when a subscriptable object is accessed with an index that is out of range.

Definition at line 138 of file Exception.h.

Public Member Functions

- IndexOutOfBounds (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.73.2. Constructor & Destructor Documentation

6.73.2.1. Loris::IndexOutOfBounds::IndexOutOfBounds (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 151 of file Exception.h.

6.73.3. Member Function Documentation

6.73.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.73.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.73.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.73.4. Member Data Documentation

6.73.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

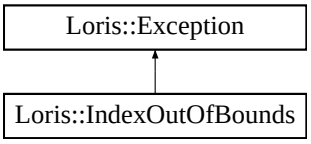
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::IndexOutOfBounds

6.74. insds Struct Reference

`#include <csoundCore.h>`

Public Attributes

- opds * nxti
- opds * nxtp
- insds * nxtinstance
- insds * prvinstance
- insds * nxtact
- insds * prvact
- insds * nxtoff
- FDCH fdch
- AUXCH auxch
- int xtratim
- MCHNBLK * m_chnbp
- insds * nxtolap
- short insno
- short m_sust
- unsigned char m_pitch
- unsigned char m_veloc
- char relesing
- char actflg
- double offbet
- double offtim
- void * pylocal
- ENVIRON_ * csound
- void * opcod_iobufs
- void * opcod_deact
- void * subins_deact
- void * nxtd
- MYFLT p0
- MYFLT p1
- MYFLT p2
- MYFLT p3

6.74.1. Member Data Documentation

6.74.1.1. char insds::actflg

Definition at line 298 of file csoundCore.h.

6.74.1.2. AUXCH insds::auxch

Definition at line 286 of file csoundCore.h.

6.74.1.3. struct ENVIRON_ * insds::csound

Definition at line 303 of file csoundCore.h.

6.74.1.4. FDCH insds::fdch

Definition at line 285 of file csoundCore.h.

6.74.1.5. short insds::insno

Definition at line 292 of file csoundCore.h.

6.74.1.6. MCHNBLK* insds::m_chnbp

Definition at line 289 of file csoundCore.h.

6.74.1.7. unsigned char insds::m_pitch

Definition at line 294 of file csoundCore.h.

6.74.1.8. short insds::m_sust

Definition at line 293 of file csoundCore.h.

6.74.1.9. unsigned char insds::m_veloc

Definition at line 295 of file csoundCore.h.

6.74.1.10. struct insds* insds::nxtact

Definition at line 282 of file csoundCore.h.

6.74.1.11. void* insds::nxtd

Definition at line 306 of file csoundCore.h.

6.74.1.12. struct opds* insds::nxti

Definition at line 278 of file csoundCore.h.

6.74.1.13. struct insds* insds::nxtinstance

Definition at line 280 of file csoundCore.h.

6.74.1.14. struct insds* insds::nxtoff

Definition at line 284 of file csoundCore.h.

6.74.1.15. struct insds* insds::nxtolap

Definition at line 291 of file csoundCore.h.

6.74.1.16. struct opds* insds::nxtp

Definition at line 279 of file csoundCore.h.

6.74.1.17. double insds::offbet

Definition at line 299 of file csoundCore.h.

6.74.1.18. double insds::offtim

Definition at line 300 of file csoundCore.h.

6.74.1.19. void* insds::opcod_deact

Definition at line 305 of file csoundCore.h.

6.74.1.20. void* insds::opcod_iobufs

Definition at line 304 of file csoundCore.h.

6.74.1.21. MYFLT insds::p0

Definition at line 307 of file csoundCore.h.

6.74.1.22. MYFLT insds::p1

Definition at line 309 of file csoundCore.h.

6.74.1.23. MYFLT insds::p2

Definition at line 310 of file csoundCore.h.

6.74.1.24. MYFLT insds::p3

Definition at line 311 of file csoundCore.h.

6.74.1.25. struct insds* insds::prvact

Definition at line 283 of file csoundCore.h.

6.74.1.26. struct insds* insds::prvinstance

Definition at line 281 of file csoundCore.h.

6.74.1.27. void* insds::pylocal

Definition at line 302 of file csoundCore.h.

6.74.1.28. char insds::relesing

Definition at line 296 of file csoundCore.h.

6.74.1.29. void * insds::subins_deact

Definition at line 305 of file csoundCore.h.

6.74.1.30. int insds::xtratim

Definition at line 287 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.75. instr Struct Reference

`#include <csoundCore.h>`

Public Attributes

- op * nxtop
- TEXT t
- int pmax
- int vmax
- int pextrab
- int mdepends
- int lclkcnt
- int lcldcnt
- int lclwcnt
- int lclacnt
- int lclpcnt
- int lclscnt
- int lclfixed
- int optxtcount
- short muted
- long localen
- long opdstot
- long * inslist
- MYFLT * psetdata
- insds * instance
- insds * lst_instance
- insds * act_instance
- instr * nxtinstxt
- int active
- int maxalloc
- MYFLT cpuload
- opcodinfo * opcode_info
- char * insname

6.75.1. Member Data Documentation

6.75.1.1. struct insds* instr::act_instance

Definition at line 201 of file csoundCore.h.

6.75.1.2. int instr::active

Definition at line 204 of file csoundCore.h.

6.75.1.3. MYFLT instr::cpuload

Definition at line 206 of file csoundCore.h.

6.75.1.4. long* instr::inslist

Definition at line 196 of file csoundCore.h.

6.75.1.5. char* instr::insname

Definition at line 208 of file csoundCore.h.

6.75.1.6. struct insds* instr::instance

Definition at line 198 of file csoundCore.h.

6.75.1.7. int instr::lclacnt

Definition at line 190 of file csoundCore.h.

6.75.1.8. int instr::lcldcnt

Definition at line 189 of file csoundCore.h.

6.75.1.9. int instr::lclfixed

Definition at line 192 of file csoundCore.h.

6.75.1.10. int instr::lclkcnt

Definition at line 189 of file csoundCore.h.

6.75.1.11. int instr::lclpcnt

Definition at line 191 of file csoundCore.h.

6.75.1.12. int instr::lclscnt

Definition at line 191 of file csoundCore.h.

6.75.1.13. int instr::lclwcnt

Definition at line 190 of file csoundCore.h.

6.75.1.14. long instr::localen

Definition at line 194 of file csoundCore.h.

6.75.1.15. struct insds* instr::lst_instance

Definition at line 200 of file csoundCore.h.

6.75.1.16. int instr::maxalloc

Definition at line 205 of file csoundCore.h.

6.75.1.17. int instr::mdepends

Definition at line 188 of file csoundCore.h.

6.75.1.18. short instr::muted

Definition at line 193 of file csoundCore.h.

6.75.1.19. struct instr* instr::nxtinstxt

Definition at line 203 of file csoundCore.h.

6.75.1.20. struct op* instr::nxtop

Definition at line 184 of file csoundCore.h.

6.75.1.21. struct opcodinfo* instr::opcode_info

Definition at line 207 of file csoundCore.h.

6.75.1.22. long instr::opdstot

Definition at line 195 of file csoundCore.h.

6.75.1.23. int instr::optxtcount

Definition at line 192 of file csoundCore.h.

6.75.1.24. int instr::pextrab

Definition at line 186 of file csoundCore.h.

6.75.1.25. int instr::pmax

Definition at line 186 of file csoundCore.h.

6.75.1.26. MYFLT* instr::psetdata

Definition at line 197 of file csoundCore.h.

6.75.1.27. TEXT instr::t

Definition at line 185 of file csoundCore.h.

6.75.1.28. int instr::vmax

Definition at line 186 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.76. Loris::InstrumentCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- Byte baseNote
- Byte detune
- Byte lowNote
- Byte highNote
- Byte lowVelocity
- Byte highVelocity
- Int_16 gain
- Loop sustainLoop
- Loop releaseLoop

Classes

- struct Loop

6.76.1. Member Data Documentation

6.76.1.1. Byte Loris::InstrumentCk::baseNote

Definition at line 149 of file AiffData.h.

6.76.1.2. Byte Loris::InstrumentCk::detune

Definition at line 150 of file AiffData.h.

6.76.1.3. Int_16 Loris::InstrumentCk::gain

Definition at line 155 of file AiffData.h.

6.76.1.4. CkHeader Loris::InstrumentCk::header

Definition at line 148 of file AiffData.h.

6.76.1.5. Byte Loris::InstrumentCk::highNote

Definition at line 152 of file AiffData.h.

6.76.1.6. Byte Loris::InstrumentCk::highVelocity

Definition at line 154 of file AiffData.h.

6.76.1.7. Byte Loris::InstrumentCk::lowNote

Definition at line 151 of file AiffData.h.

6.76.1.8. Byte Loris::InstrumentCk::lowVelocity

Definition at line 153 of file AiffData.h.

6.76.1.9. Loop Loris::InstrumentCk::releaseLoop

Definition at line 166 of file AiffData.h.

6.76.1.10. Loop Loris::InstrumentCk::sustainLoop

Definition at line 165 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.77. Loris::InstrumentCk::Loop Struct Reference

`#include <AiffData.h>`

Public Attributes

- Int_16 playMode
- Uint_16 beginLoop
- Uint_16 endLoop

6.77.1. Member Data Documentation

6.77.1.1. Uint_16 Loris::InstrumentCk::Loop::beginLoop

Definition at line 160 of file AiffData.h.

6.77.1.2. Uint_16 Loris::InstrumentCk::Loop::endLoop

Definition at line 162 of file AiffData.h.

6.77.1.3. Int_16 Loris::InstrumentCk::Loop::playMode

Definition at line 159 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.78. Loris::InvalidArgument Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::InvalidArgument::

Public Member Functions

- InvalidArgument (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.78.1. Constructor & Destructor Documentation

6.78.1.1. Loris::InvalidArgument::InvalidArgument (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 227 of file Exception.h.

6.78.2. Member Function Documentation

6.78.2.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.78.2.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:

a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.78.2.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.78.3. Member Data Documentation

6.78.3.1. std::string Loris::Exception::_sbuf [protected, inherited]

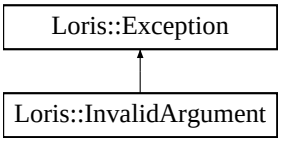
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::InvalidArgument

6.79. Loris::InvalidIterator Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::InvalidIterator::

6.79.1. Detailed Description

Class of exceptions thrown when an Iterator is found to be badly configured or otherwise invalid.

Definition at line 189 of file Exception.h.

Public Member Functions

- InvalidIterator (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.79.2. Constructor & Destructor Documentation

6.79.2.1. Loris::InvalidIterator::InvalidIterator (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 202 of file Exception.h.

6.79.3. Member Function Documentation

6.79.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.79.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.79.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.79.4. Member Data Documentation

6.79.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

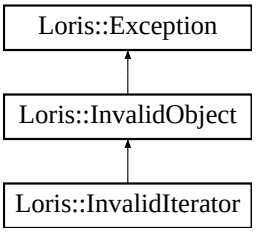
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::InvalidObject
Loris::InvalidIterator

6.80. Loris::InvalidObject Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::InvalidObject::

6.80.1. Detailed Description

Class of exceptions thrown when an object is found to be badly configured or otherwise invalid.

Definition at line 163 of file Exception.h.

Public Member Functions

- InvalidObject (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.80.2. Constructor & Destructor Documentation

6.80.2.1. Loris::InvalidObject::InvalidObject (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 176 of file Exception.h.

6.80.3. Member Function Documentation

6.80.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.80.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.80.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.80.4. Member Data Documentation

6.80.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

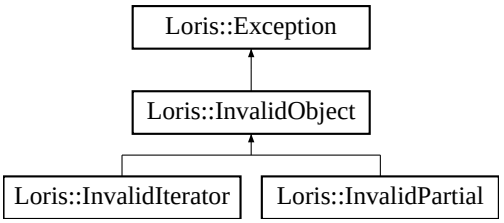
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::InvalidObject
Loris::InvalidIterator
Loris::InvalidPartial

6.81. Loris::InvalidPartial Class Reference

`#include <Partial.h>`

Inheritance diagram for Loris::InvalidPartial::

6.81.1. Detailed Description

Class of exceptions thrown when a Partial is found to be badly configured or otherwise invalid.

Definition at line 735 of file Partial.h.

Public Member Functions

- InvalidPartial (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.81.2. Constructor & Destructor Documentation

6.81.2.1. Loris::InvalidPartial::InvalidPartial (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically byt he Throw macro).

Definition at line 748 of file Partial.h.

6.81.3. Member Function Documentation

6.81.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.81.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.81.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.81.4. Member Data Documentation

6.81.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

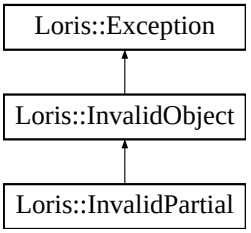
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Partial.h


Loris::Exception
Loris::InvalidObject
Loris::InvalidPartial

6.82. Loris::PartialUtils::isLabelEqual Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- isLabelEqual (int l)
- bool operator() (const Partial &p) const
- bool operator() (const Partial *p) const

Private Attributes

- int label

6.82.1. Constructor & Destructor Documentation

6.82.1.1. Loris::PartialUtils::isLabelEqual::isLabelEqual (int *l*) [inline]

Definition at line 423 of file PartialUtils.h.

References label.

6.82.2. Member Function Documentation

6.82.2.1. bool Loris::PartialUtils::isLabelEqual::operator() (const Partial * *p*) const [inline]

Definition at line 429 of file PartialUtils.h.

References label.

6.82.2.2. bool Loris::PartialUtils::isLabelEqual::operator() (const Partial & *p*) const [inline]

Definition at line 426 of file PartialUtils.h.

References label.

6.82.3. Member Data Documentation

6.82.3.1. int Loris::PartialUtils::isLabelEqual::label [private]

Definition at line 433 of file PartialUtils.h.

Referenced by isLabelEqual(), and operator()().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.83. Loris::PartialUtils::isLabelGreater Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- isLabelGreater (int l)
- bool operator() (const Partial &p) const
- bool operator() (const Partial *p) const

Private Attributes

- int label

6.83.1. Constructor & Destructor Documentation

6.83.1.1. Loris::PartialUtils::isLabelGreater::isLabelGreater (int *l*) [inline]

Definition at line 443 of file PartialUtils.h.

References label.

6.83.2. Member Function Documentation

6.83.2.1. bool Loris::PartialUtils::isLabelGreater::operator() (const Partial * *p*) const [inline]

Definition at line 449 of file PartialUtils.h.

References label.

6.83.2.2. bool Loris::PartialUtils::isLabelGreater::operator() (const Partial & *p*) const [inline]

Definition at line 446 of file PartialUtils.h.

References label.

6.83.3. Member Data Documentation

6.83.3.1. int Loris::PartialUtils::isLabelGreater::label [private]

Definition at line 453 of file PartialUtils.h.

Referenced by isLabelGreater(), and operator()().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.84. Loris::PartialUtils::isLabelLess Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- isLabelLess (int l)
- bool operator() (const Partial &p) const
- bool operator() (const Partial *p) const

Private Attributes

- int label

6.84.1. Constructor & Destructor Documentation

6.84.1.1. Loris::PartialUtils::isLabelLess::isLabelLess (int *l*) [inline]

Definition at line 463 of file PartialUtils.h.

References label.

6.84.2. Member Function Documentation

6.84.2.1. bool Loris::PartialUtils::isLabelLess::operator() (const Partial * *p*) const [inline]

Definition at line 469 of file PartialUtils.h.

References label.

6.84.2.2. bool Loris::PartialUtils::isLabelLess::operator() (const Partial & *p*) const [inline]

Definition at line 466 of file PartialUtils.h.

References label.

6.84.3. Member Data Documentation

6.84.3.1. int Loris::PartialUtils::isLabelLess::label [private]

Definition at line 473 of file PartialUtils.h.

Referenced by isLabelLess(), and operator()().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.85. Loris::KaiserWindow Class Reference

`#include <KaiserWindow.h>`

Static Public Member Functions

- static void create (std::vector< double > &samples, double shape)
- static double computeShape (double atten)
- static long computeLength (double width, double alpha)

Private Member Functions

- KaiserWindow (void)

6.85.1. Constructor & Destructor Documentation

6.85.1.1. Loris::KaiserWindow::KaiserWindow (void) [private]

6.85.2. Member Function Documentation

6.85.2.1. static long Loris::KaiserWindow::computeLength (double *width*, double *alpha*) [static]

6.85.2.2. static double Loris::KaiserWindow::computeShape (double *atten*) [static]

6.85.2.3. static void Loris::KaiserWindow::create (std::vector< double > & *samples*, double *shape*) [static]

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/KaiserWindow.h

6.86. lblblk Struct Reference

`#include <csoundCore.h>`

Public Attributes

- OPDS h
- OPDS * prvi
- OPDS * prvp

6.86.1. Member Data Documentation

6.86.1.1. OPDS lblblk::h

Definition at line 328 of file csoundCore.h.

6.86.1.2. OPDS* lblblk::prvi

Definition at line 329 of file csoundCore.h.

6.86.1.3. OPDS* lblblk::prvp

Definition at line 330 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.87. Loris::BreakpointUtils::less_frequency Struct Reference

`#include <BreakpointUtils.h>`

Public Member Functions

- bool operator() (const Breakpoint &lhs, const Breakpoint &rhs) const

6.87.1. Member Function Documentation

6.87.1.1. bool Loris::BreakpointUtils::less_frequency::operator() (const Breakpoint & *lhs*, const Breakpoint & *rhs*) const [inline]

Definition at line 135 of file BreakpointUtils.h.

References Loris::Breakpoint::frequency().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/BreakpointUtils.h

6.88. csound::Lindenmayer Class Reference

`#include <Lindenmayer.hpp>`

Inheritance diagram for csound::Lindenmayer::

6.88.1. Detailed Description

This class implements a Lindenmayer system in music space for a turtle that writes either notes into a score, or Jones-Parks grains into a memory soundfile. The Z dimension of note space is used for chirp rate. The actions of the turtle are rescaled to fit the specified bounding hypercube. The turtle commands are represented by letters (all n default to 1):

- G = Write the current state of the turtle into the soundfile as a grain.
- Mn = Translate the turtle by adding to its state its step times its orientation times n.
- Rabn = Rotate the turtle from dimension a to dimension b by angle 2 pi / (angleCount * n)
- Uan = Vary the turtle state on dimension a by a normalized (-1 through +1) uniformly distributed random variable times n.
- Gan = Vary the turtle state on dimension a by a normalized (-1 through +1) Gaussian random variable times n.
- T=an = Assign to dimension a of the turtle state the value n.
- T*an = Multiply dimension a of the turtle state by n.
- T/an = Divide dimension a of the turtle state by n.
- T+an = Add to dimension a of the turtle state the value n.
- T-an = Subtract from dimension a of the turtle state the value n.
- S=an = Assign to dimension a of the turtle step the value n.
- S*an = Multiply dimension a of the turtle step by n.
- S/an = Divide dimension a of the turtle step by n.
- S+an = Add to dimension a of the turtle step the value n.
- S-an = Subtract from dimension a of the turtle step the value n.
- [= Push the current state of the turtle state onto a stack.
-] = Pop the current state of the turtle from the stack.

Definition at line 77 of file Lindenmayer.hpp.

Public Member Functions

- Lindenmayer ()
- virtual ~Lindenmayer ()
- virtual int getIterationCount () const
- virtual void setIterationCount (int count)
- virtual double getAngle () const
- virtual void setAngle (double angle)
- virtual std::string getAxiom () const

- virtual void setAxiom (std::string axiom)
- virtual void addRule (std::string command, std::string replacement)
- virtual std::string getReplacement (std::string command)
- virtual void generate ()
- virtual void clear ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Member Functions

- virtual void interpret (std::string command, bool render)
- virtual int getDimension (char dimension) const
- virtual void rewrite ()
- virtual ublas::matrix< double > createRotation (int dimension1, int dimension2, double angle) const
- virtual void updateActual (Event &event)
- virtual void initialize ()

Protected Attributes

- int iterationCount
- double angle
- std::string axiom
- Event turtle
- Event turtleStep
- Event turtleOrientation
- std::map< std::string, std::string > rules
- std::stack< Event > turtleStack
- std::stack< Event > turtleStepStack
- std::stack< Event > turtleOrientationStack
- clock_t beganAt
- clock_t endedAt
- clock_t elapsed
- Score score
- ublas::matrix< double > localCoordinates

6.88.2. Constructor & Destructor Documentation

6.88.2.1. csound::Lindenmayer::Lindenmayer ()

6.88.2.2. virtual csound::Lindenmayer::~Lindenmayer () [virtual]

6.88.3. Member Function Documentation

6.88.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.88.3.2. virtual void csound::Lindenmayer::addRule (std::string *command*, std::string *replacement*) [virtual]

6.88.3.3. virtual void csound::Lindenmayer::clear () [virtual]

Reimplemented from csound::Node.

6.88.3.4. virtual ublas::matrix<double> csound::Lindenmayer::createRotation (int *dimension1*, int *dimension2*, double *angle*) const [protected, virtual]

6.88.3.5. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.88.3.6. virtual void csound::Lindenmayer::generate () [virtual]

6.88.3.7. virtual double csound::Lindenmayer::getAngle () const [virtual]

6.88.3.8. virtual std::string csound::Lindenmayer::getAxiom () const [virtual]

6.88.3.9. virtual int csound::Lindenmayer::getDimension (char *dimension*) const [protected, virtual]

6.88.3.10. virtual int csound::Lindenmayer::getIterationCount () const [virtual]

6.88.3.11. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.88.3.12. virtual std::string csound::Lindenmayer::getReplacement (std::string *command*) [virtual]

6.88.3.13. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.88.3.14. virtual void csound::Lindenmayer::initialize () [protected, virtual]

6.88.3.15. virtual void csound::Lindenmayer::interpret (std::string *command*, bool *render*) [protected, virtual]

6.88.3.16. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.88.3.17. virtual void csound::ScoreNode::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual, inherited]

The default implementation does nothing.

Reimplemented from csound::Node.

Reimplemented in csound::Cell, csound::Hocket, csound::MCRM, and csound::Rescale.

6.88.3.18. virtual void csound::Lindenmayer::rewrite () [protected, virtual]

6.88.3.19. virtual void csound::Lindenmayer::setAngle (double *angle*) [virtual]

6.88.3.20. virtual void csound::Lindenmayer::setAxiom (std::string *axiom*) [virtual]

6.88.3.21. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.88.3.22. virtual void csound::Lindenmayer::setIterationCount (int *count*) [virtual]

6.88.3.23. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.88.3.24. virtual void csound::Lindenmayer::updateActual (Event & *event*) [protected, virtual]

6.88.4. Member Data Documentation

6.88.4.1. double csound::Lindenmayer::angle [protected]

Definition at line 82 of file Lindenmayer.hpp.

6.88.4.2. std::string csound::Lindenmayer::axiom [protected]

Definition at line 83 of file Lindenmayer.hpp.

6.88.4.3. clock_t csound::Lindenmayer::beganAt [protected]

Definition at line 91 of file Lindenmayer.hpp.

6.88.4.4. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.
Definition at line 57 of file Node.hpp.

6.88.4.5. clock_t csound::Lindenmayer::elapsed [protected]

Definition at line 93 of file Lindenmayer.hpp.

6.88.4.6. clock_t csound::Lindenmayer::endedAt [protected]

Definition at line 92 of file Lindenmayer.hpp.

6.88.4.7. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.88.4.8. int csound::Lindenmayer::iterationCount [protected]

Definition at line 81 of file Lindenmayer.hpp.

6.88.4.9. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.88.4.10. std::map<std::string, std::string> csound::Lindenmayer::rules [protected]

Definition at line 87 of file Lindenmayer.hpp.

6.88.4.11. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

6.88.4.12. Event csound::Lindenmayer::turtle [protected]

Definition at line 84 of file Lindenmayer.hpp.

6.88.4.13. Event csound::Lindenmayer::turtleOrientation [protected]

Definition at line 86 of file Lindenmayer.hpp.

6.88.4.14. std::stack<Event> csound::Lindenmayer::turtleOrientationStack [protected]

Definition at line 90 of file Lindenmayer.hpp.

6.88.4.15. std::stack<Event> csound::Lindenmayer::turtleStack [protected]

Definition at line 88 of file Lindenmayer.hpp.

6.88.4.16. Event csound::Lindenmayer::turtleStep [protected]

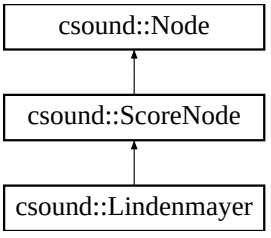
Definition at line 85 of file Lindenmayer.hpp.

6.88.4.17. std::stack<Event> csound::Lindenmayer::turtleStepStack [protected]

Definition at line 89 of file Lindenmayer.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Lindenmayer.hpp


csound::Node
csound::ScoreNode
csound::Lindenmayer

6.89. csound::Logger Class Reference

`#include <System.hpp>`

Public Member Functions

- Logger ()
- virtual ~Logger ()
- virtual void write (const char *text)

6.89.1. Constructor & Destructor Documentation

6.89.1.1. csound::Logger::Logger ()

6.89.1.2. virtual csound::Logger::~Logger () [virtual]

6.89.2. Member Function Documentation

6.89.2.1. virtual void csound::Logger::write (const char * *text*) [virtual]

The documentation for this class was generated from the following file:

- frontends/CsoundVST/System.hpp

6.90. Loris::Marker Class Reference

`#include <Marker.h>`

Public Member Functions

- Marker (void)
- Marker (double t, const std::string &s)
- Marker (const Marker &other)
- Marker & operator= (const Marker &rhs)
- bool operator< (const Marker &rhs) const
- std::string & name (void)
- const std::string & name (void) const
- double time (void) const
- void setName (const std::string &s)
- void setTime (double t)

Private Attributes

- double m_time
- std::string m_name

Classes

- struct sortByName
- struct sortByTime

6.90.1. Constructor & Destructor Documentation

6.90.1.1. Loris::Marker::Marker (void)

6.90.1.2. Loris::Marker::Marker (double *t*, const std::string & *s*)

6.90.1.3. Loris::Marker::Marker (const Marker & *other*)

6.90.2. Member Function Documentation

6.90.2.1. const std::string& Loris::Marker::name (void) const

6.90.2.2. std::string& Loris::Marker::name (void)

Referenced by Loris::Marker::sortByName::operator()().

6.90.2.3. bool Loris::Marker::operator< (const Marker & *rhs*) const

6.90.2.4. Marker& Loris::Marker::operator= (const Marker & *rhs*)

6.90.2.5. void Loris::Marker::setName (const std::string & *s*)

6.90.2.6. void Loris::Marker::setTime (double *t*)

6.90.2.7. double Loris::Marker::time (void) const

Referenced by Loris::Marker::sortByTime::operator()().

6.90.3. Member Data Documentation

6.90.3.1. std::string Loris::Marker::m_name [private]

Definition at line 58 of file Marker.h.

6.90.3.2. double Loris::Marker::m_time [private]

Definition at line 57 of file Marker.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Marker.h

6.91. Loris::Marker::sortByName Struct Reference

`#include <Marker.h>`

Public Member Functions

- bool operator() (const Marker &lhs, const Marker &rhs) const

6.91.1. Member Function Documentation

6.91.1.1. bool Loris::Marker::sortByName::operator() (const Marker & *lhs*, const Marker & *rhs*) const [inline]

Definition at line 115 of file Marker.h.

References Loris::Marker::name().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/Marker.h

6.92. Loris::Marker::sortByTime Struct Reference

`#include <Marker.h>`

Public Member Functions

- bool operator() (const Marker &lhs, const Marker &rhs) const

6.92.1. Member Function Documentation

6.92.1.1. bool Loris::Marker::sortByTime::operator() (const Marker & *lhs*, const Marker & *rhs*) const [inline]

Definition at line 127 of file Marker.h.

References Loris::Marker::time().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/Marker.h

6.93. Loris::MarkerCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- Uint_16 numMarkers
- std::vector< MarkerCk::Marker > markers

Classes

- struct Marker

6.93.1. Member Data Documentation

6.93.1.1. CkHeader Loris::MarkerCk::header

Definition at line 133 of file AiffData.h.

6.93.1.2. std::vector< MarkerCk::Marker > Loris::MarkerCk::markers

Definition at line 143 of file AiffData.h.

6.93.1.3. Uint_16 Loris::MarkerCk::numMarkers

Definition at line 134 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.94. Loris::MarkerCk::Marker Struct Reference

`#include <AiffData.h>`

Public Attributes

- Uint_16 markerID
- Uint_32 position
- std::string markerName

6.94.1. Member Data Documentation

6.94.1.1. Uint_16 Loris::MarkerCk::Marker::markerID

Definition at line 138 of file AiffData.h.

6.94.1.2. std::string Loris::MarkerCk::Marker::markerName

Definition at line 140 of file AiffData.h.

6.94.1.3. Uint_32 Loris::MarkerCk::Marker::position

Definition at line 139 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.95. mchnblk Struct Reference

`#include <csoundCore.h>`

Public Attributes

- short pgmno
- short insno
- short RegParNo
- short mono
- MONPCH * monobas
- MONPCH * monocur
- insds * kinsptr [128]
- MYFLT polyaft [128]
- MYFLT ctl_val [136]
- short pgm2ins [128]
- MYFLT aftouch
- MYFLT pchbend
- MYFLT pbensens
- MYFLT dummy_
- short ksuscnt
- short sustaining
- int dpmsb
- int dplsb
- int datenabl
- DKLST * klists
- DPARM * dparms

6.95.1. Member Data Documentation

6.95.1.1. MYFLT mchnblk::aftouch

Definition at line 262 of file csoundCore.h.

6.95.1.2. MYFLT mchnblk::ctl_val[136]

Definition at line 260 of file csoundCore.h.

6.95.1.3. int mchnblk::datenabl

Definition at line 270 of file csoundCore.h.

6.95.1.4. DPARM* mchnblk::dparms

Definition at line 272 of file csoundCore.h.

6.95.1.5. int mchnblk::dplsb

Definition at line 269 of file csoundCore.h.

6.95.1.6. int mchnblk::dpmsb

Definition at line 268 of file csoundCore.h.

6.95.1.7. MYFLT mchnblk::dummy_

Definition at line 265 of file csoundCore.h.

6.95.1.8. short mchnblk::insno

Definition at line 253 of file csoundCore.h.

6.95.1.9. struct insds* mchnblk::kinsptr[128]

Definition at line 258 of file csoundCore.h.

6.95.1.10. DKLST* mchnblk::klists

Definition at line 271 of file csoundCore.h.

6.95.1.11. short mchnblk::ksuscnt

Definition at line 266 of file csoundCore.h.

6.95.1.12. short mchnblk::mono

Definition at line 255 of file csoundCore.h.

6.95.1.13. MONPCH* mchnblk::monobas

Definition at line 256 of file csoundCore.h.

6.95.1.14. MONPCH* mchnblk::monocur

Definition at line 257 of file csoundCore.h.

6.95.1.15. MYFLT mchnblk::pbensens

Definition at line 264 of file csoundCore.h.

6.95.1.16. MYFLT mchnblk::pchbend

Definition at line 263 of file csoundCore.h.

6.95.1.17. short mchnblk::pgm2ins[128]

Definition at line 261 of file csoundCore.h.

6.95.1.18. short mchnblk::pgmno

Definition at line 252 of file csoundCore.h.

6.95.1.19. MYFLT mchnblk::polyaft[128]

Definition at line 259 of file csoundCore.h.

6.95.1.20. short mchnblk::RegParNo

Definition at line 254 of file csoundCore.h.

6.95.1.21. short mchnblk::sustaining

Definition at line 267 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.96. csound::MCRM Class Reference

`#include <MCRM.hpp>`

Inheritance diagram for csound::MCRM::

Public Member Functions

- MCRM ()
- virtual ~MCRM ()
- void setDepth (int depth)
- void resize (size_t transformations)
- void setTransformationElement (size_t index, size_t row, size_t column, double value)
- void setWeight (size_t precursor, size_t successor, double weight)
- void generate ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- Score score
- ublas::matrix< double > localCoordinates

Private Member Functions

- void iterate (int depth, size_t p, const Event &event, double weight)

Private Attributes

- std::vector< ublas::matrix< double > > transformations
- ublas::matrix< double > weights
- int depth

6.96.1. Constructor & Destructor Documentation

6.96.1.1. csound::MCRM::MCRM ()

6.96.1.2. virtual csound::MCRM::~MCRM () [virtual]

6.96.2. Member Function Documentation

6.96.2.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.96.2.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.96.2.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.96.2.4. void csound::MCRM::generate ()

6.96.2.5. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.96.2.6. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.96.2.7. void csound::MCRM::iterate (int *depth*, size_t *p*, const Event & *event*, double *weight*) [private]

6.96.2.8. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.96.2.9. virtual void csound::MCRM::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::ScoreNode.

6.96.2.10. void csound::MCRM::resize (size_t *transformations*)

6.96.2.11. void csound::MCRM::setDepth (int *depth*)

6.96.2.12. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.96.2.13. void csound::MCRM::setTransformationElement (size_t *index*, size_t *row*, size_t *column*, double *value*)

6.96.2.14. void csound::MCRM::setWeight (size_t *precursor*, size_t *successor*, double *weight*)

6.96.2.15. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.96.3. Member Data Documentation

6.96.3.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.96.3.2. int csound::MCRM::depth [private]

Definition at line 46 of file MCRM.hpp.

6.96.3.3. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.96.3.4. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.96.3.5. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

6.96.3.6. std::vector< ublas::matrix<double> > csound::MCRM::transformations [private]

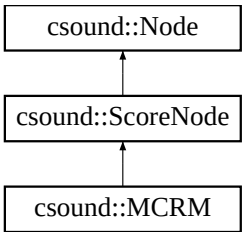
Definition at line 42 of file MCRM.hpp.

6.96.3.7. ublas::matrix<double> csound::MCRM::weights [private]

Definition at line 44 of file MCRM.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/MCRM.hpp


csound::Node
csound::ScoreNode
csound::MCRM

6.97. MEMFIL Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char filename [256]
- char * beginp
- char * endp
- long length
- MEMFIL * next

6.97.1. Member Data Documentation

6.97.1.1. char* MEMFIL::beginp

Definition at line 401 of file csoundCore.h.

6.97.1.2. char* MEMFIL::endp

Definition at line 402 of file csoundCore.h.

6.97.1.3. char MEMFIL::filename[256]

Definition at line 400 of file csoundCore.h.

6.97.1.4. long MEMFIL::length

Definition at line 403 of file csoundCore.h.

6.97.1.5. struct MEMFIL* MEMFIL::next

Definition at line 404 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.98. MEVENT Struct Reference

`#include <csoundCore.h>`

Public Attributes

- short type
- short chan
- short dat1
- short dat2

6.98.1. Member Data Documentation

6.98.1.1. short MEVENT::chan

Definition at line 465 of file csoundCore.h.

6.98.1.2. short MEVENT::dat1

Definition at line 466 of file csoundCore.h.

6.98.1.3. short MEVENT::dat2

Definition at line 467 of file csoundCore.h.

6.98.1.4. short MEVENT::type

Definition at line 464 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.99. csound::MidiEvent Class Reference

`#include <Midifile.hpp>`

6.99.1. Detailed Description

This class is used to store ALL Midi messages.

Definition at line 86 of file Midifile.hpp.

Public Member Functions

- MidiEvent (void)
- virtual ~MidiEvent (void)
- virtual void read (std::istream &stream, MidiFile &midiFile)
- virtual void write (std::ostream &stream, MidiFile &midiFile, int lastTick)
- virtual int getStatus (void)
- virtual int getStatusNybble (void)
- virtual int getChannelNybble (void)
- virtual int getKey (void)
- virtual int getVelocity (void)
- virtual int getMetaType (void)
- virtual unsigned char getMetaData (int i)
- virtual size_t getMetaSize (void)
- virtual unsigned char read (std::istream &stream)
- virtual bool isChannelVoiceMessage ()
- virtual bool isNoteOn (void)
- virtual bool isNoteOff (void)
- virtual bool isMatchingNoteOff (MidiEvent &offEvent)

Public Attributes

- int ticks
- double time

Friends

- bool operator< (const MidiEvent &a, MidiEvent &b)

6.99.2. Constructor & Destructor Documentation

6.99.2.1. csound::MidiEvent::MidiEvent (void)

6.99.2.2. virtual csound::MidiEvent::~MidiEvent (void) [virtual]

6.99.3. Member Function Documentation

6.99.3.1. virtual int csound::MidiEvent::getChannelNybble (void) [virtual]

6.99.3.2. virtual int csound::MidiEvent::getKey (void) [virtual]

6.99.3.3. virtual unsigned char csound::MidiEvent::getMetaData (int *i*) [virtual]

6.99.3.4. virtual size_t csound::MidiEvent::getMetaSize (void) [virtual]

6.99.3.5. virtual int csound::MidiEvent::getMetaType (void) [virtual]

6.99.3.6. virtual int csound::MidiEvent::getStatus (void) [virtual]

6.99.3.7. virtual int csound::MidiEvent::getStatusNybble (void) [virtual]

6.99.3.8. virtual int csound::MidiEvent::getVelocity (void) [virtual]

6.99.3.9. virtual bool csound::MidiEvent::isChannelVoiceMessage () [virtual]

6.99.3.10. virtual bool csound::MidiEvent::isMatchingNoteOff (MidiEvent & *offEvent*) [virtual]

6.99.3.11. virtual bool csound::MidiEvent::isNoteOff (void) [virtual]

6.99.3.12. virtual bool csound::MidiEvent::isNoteOn (void) [virtual]

6.99.3.13. virtual unsigned char csound::MidiEvent::read (std::istream & *stream*) [virtual]

6.99.3.14. virtual void csound::MidiEvent::read (std::istream & *stream*, MidiFile & *midiFile*) [virtual]

6.99.3.15. virtual void csound::MidiEvent::write (std::ostream & *stream*, MidiFile & *midiFile*, int *lastTick*) [virtual]

6.99.4. Friends And Related Function Documentation

6.99.4.1. bool operator< (const MidiEvent & *a*, MidiEvent & *b*) [friend]

6.99.5. Member Data Documentation

6.99.5.1. int csound::MidiEvent::ticks

Definition at line 89 of file Midifile.hpp.

6.99.5.2. double csound::MidiEvent::time

Definition at line 90 of file Midifile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp

6.100. csound::MidiFile Class Reference

`#include <Midifile.hpp>`

6.100.1. Detailed Description

Reads and writes format 0 and format 1 standard MIDI files.

Definition at line 130 of file Midifile.hpp.

Public Types

- enum MidiEventTypes {

  CHANNEL_NOTE_OFF = 0x80, CHANNEL_NOTE_ON = 0x90, CHANNEL_KEY_PRESSURE = 0xa0, CHANNEL_CONTROL_CHANGE = 0xb0,

  CHANNEL_PROGRAM_CHANGE = 0xc0, CHANNEL_AFTER_TOUCH = 0xd0, CHANNEL_PITCH_BEND = 0xe0, SYSTEM_EXCLUSIVE = 0xf0,

  SYSTEM_MIDI_TIME_CODE = 0xf1, SYSTEM_SONG_POSITION_POINTER = 0xf2, SYSTEM_SONG_SELECT = 0xf3, SYSTEM_TUNE_REQUEST = 0xf6,

  SYSTEM_END_OF_EXCLUSIVE = 0xf7, SYSTEM_TIMING_CLOCK = 0xf8, SYSTEM_START = 0xfa, SYSTEM_CONTINUE = 0xfb,

  SYSTEM_STOP = 0xfc, SYSTEM_ACTIVE_SENSING = 0xfe, META_EVENT = 0xff }
- enum MetaEventTypes {

  META_SEQUENCE_NUMBER = 0x00, META_TEXT_EVENT = 0x01, META_COPYRIGHT_NOTICE = 0x02, META_SEQUENCE_NAME = 0x03,

  META_INSTRUMENT_NAME = 0x04, META_LYRIC = 0x05, META_MARKER = 0x06, META_CUE_POINT = 0x07,

  META_CHANNEL_PREFIX = 0x20, META_END_OF_TRACK = 0x2f, META_SET_TEMPO = 0x51, META_SMPTE_OFFSET = 0x54,

  META_TIME_SIGNATURE = 0x58, META_KEY_SIGNATURE = 0x59, META_SEQUENCER_SPECIFIC = 0x74 }
- enum MidiControllers {

  CONTROLLER_MOD_WHEEL = 1, CONTROLLER_BREATH = 2, CONTROLLER_FOOT = 4, CONTROLLER_BALANCE = 8,

  CONTROLLER_PAN = 10, CONTROLLER_EXPRESSION = 11, CONTROLLER_DAMPER_PEDAL = 0x40, CONTROLLER_PORTAMENTO = 0x41,

  CONTROLLER_SOSTENUTO = 0x42, CONTROLLER_SOFT_PEDAL = 0x43, CONTROLLER_GENERAL_4 = 0x44, CONTROLLER_HOLD_2 = 0x45,

  CONTROLLER_7GENERAL_5 = 0x50, CONTROLLER_GENERAL_6 = 0x51, CONTROLLER_GENERAL_7 = 0x52, CONTROLLER_GENERAL_8 = 0x53,

  CONTROLLER_TREMOLO_DEPTH = 0x5c, CONTROLLER_CHORUS_DEPTH = 0x5d, CONTROLLER_DETUNE = 0x5e, CONTROLLER_PHASER_DEPTH = 0x5f,

  CONTROLLER_DATA_INC = 0x60, CONTROLLER_DATA_DEC = 0x61, CONTROLLER_NON_REG_LSB = 0x62, CONTROLLER_NON_REG_MSB = 0x63,

  CONTROLLER_REG_LSB = 0x64, CONTROLLER_REG_MSG = 0x65, CONTROLLER_CONTINUOUS_AFTERTOUCH = 128 }

Public Member Functions

- void computeTimes (void)
- MidiFile (void)
- virtual ~MidiFile (void)
- virtual void clear (void)
- virtual void read (std::istream &stream)
- virtual void write (std::ostream &stream)
- virtual void load (std::string filename)
- virtual void save (std::string filename)
- virtual void dump (std::ostream &stream)
- virtual void sort (void)

Static Public Member Functions

- static int readVariableLength (std::istream &stream)
- static void writeVariableLength (std::ostream &stream, int value)
- static int toInt (int c1, int c2, int c3, int c4)
- static short toShort (int c1, int c2)
- static int readInt (std::istream &stream)
- static void writeInt (std::ostream &stream, int value)
- static short readShort (std::istream &stream)
- static void writeShort (std::ostream &stream, short value)
- static int chunkName (int a, int b, int c, int d)

Public Attributes

- int currentTick
- double currentTime
- double currentSecondsPerTick
- double microsecondsPerQuarterNote
- unsigned char lastStatus
- MidiHeader midiHeader
- TempoMap tempoMap
- std::vector< MidiTrack > midiTracks

6.100.2. Member Enumeration Documentation

6.100.2.1. enum csound::MidiFile::MetaEventTypes

Enumeration values:

META_SEQUENCE_NUMBER

META_TEXT_EVENT

META_COPYRIGHT_NOTICE

META_SEQUENCE_NAME

META_INSTRUMENT_NAME

META_LYRIC

META_MARKER

META_CUE_POINT

META_CHANNEL_PREFIX

META_END_OF_TRACK
META_SET_TEMPO
META_SMPTE_OFFSET
META_TIME_SIGNATURE
META_KEY_SIGNATURE
META_SEQUENCER_SPECIFIC

Definition at line 154 of file Midifile.hpp.

6.100.2.2. enum csound::MidiFile::MidiControllers

Enumeration values:

CONTROLLER_MOD_WHEEL
CONTROLLER_BREATH
CONTROLLER_FOOT
CONTROLLER_BALANCE
CONTROLLER_PAN
CONTROLLER_EXPRESSION
CONTROLLER_DAMPER_PEDAL
CONTROLLER_PORTAMENTO
CONTROLLER_SOSTENUTO
CONTROLLER_SOFT_PEDAL
CONTROLLER_GENERAL_4
CONTROLLER_HOLD_2
CONTROLLER_7GENERAL_5
CONTROLLER_GENERAL_6
CONTROLLER_GENERAL_7
CONTROLLER_GENERAL_8
CONTROLLER_TREMOLO_DEPTH
CONTROLLER_CHORUS_DEPTH
CONTROLLER_DETUNE
CONTROLLER_PHASER_DEPTH
CONTROLLER_DATA_INC
CONTROLLER_DATA_DEC
CONTROLLER_NON_REG_LSB
CONTROLLER_NON_REG_MSB
CONTROLLER_REG_LSB
CONTROLLER_REG_MSG
CONTROLLER_CONTINUOUS_AFTERTOUCH

Definition at line 171 of file Midifile.hpp.

6.100.2.3. enum csound::MidiFile::MidiEventTypes

Enumeration values:

CHANNEL_NOTE_OFF

CHANNEL_NOTE_ON

CHANNEL_KEY_PRESSURE

CHANNEL_CONTROL_CHANGE

CHANNEL_PROGRAM_CHANGE

CHANNEL_AFTER_TOUCH

CHANNEL_PITCH_BEND

SYSTEM_EXCLUSIVE

SYSTEM_MIDI_TIME_CODE

SYSTEM_SONG_POSITION_POINTER

SYSTEM_SONG_SELECT

SYSTEM_TUNE_REQUEST

SYSTEM_END_OF_EXCLUSIVE

SYSTEM_TIMING_CLOCK

SYSTEM_START

SYSTEM_CONTINUE

SYSTEM_STOP

SYSTEM_ACTIVE_SENSING

META_EVENT

Definition at line 133 of file Midifile.hpp.

6.100.3. Constructor & Destructor Documentation

6.100.3.1. csound::MidiFile::MidiFile (void)

6.100.3.2. virtual csound::MidiFile::~MidiFile (void) [virtual]

6.100.4. Member Function Documentation

6.100.4.1. static int csound::MidiFile::chunkName (int *a*, int *b*, int *c*, int *d*) [static]

6.100.4.2. virtual void csound::MidiFile::clear (void) [virtual]

6.100.4.3. void csound::MidiFile::computeTimes (void)

6.100.4.4. virtual void csound::MidiFile::dump (std::ostream & *stream*) [virtual]

6.100.4.5. virtual void csound::MidiFile::load (std::string *filename*) [virtual]

6.100.4.6. virtual void csound::MidiFile::read (std::istream & *stream*) [virtual]

6.100.4.7. static int csound::MidiFile::readInt (std::istream & *stream*) [static]

6.100.4.8. static short csound::MidiFile::readShort (std::istream & *stream*) [static]

6.100.4.9. static int csound::MidiFile::readVariableLength (std::istream & *stream*) [static]

6.100.4.10. virtual void csound::MidiFile::save (std::string *filename*) [virtual]

6.100.4.11. virtual void csound::MidiFile::sort (void) [virtual]

6.100.4.12. static int csound::MidiFile::toInt (int *c1*, int *c2*, int *c3*, int *c4*) [static]

6.100.4.13. static short csound::MidiFile::toShort (int *c1*, int *c2*) [static]

6.100.4.14. virtual void csound::MidiFile::write (std::ostream & *stream*) [virtual]

6.100.4.15. static void csound::MidiFile::writeInt (std::ostream & *stream*, int *value*) [static]

6.100.4.16. static void csound::MidiFile::writeShort (std::ostream & *stream*, short *value*) [static]

6.100.4.17. static void csound::MidiFile::writeVariableLength (std::ostream & *stream*, int *value*) [static]

6.100.5. Member Data Documentation

6.100.5.1. double csound::MidiFile::currentSecondsPerTick

Definition at line 215 of file Midifile.hpp.

6.100.5.2. int csound::MidiFile::currentTick

Definition at line 213 of file Midifile.hpp.

6.100.5.3. double csound::MidiFile::currentTime

Definition at line 214 of file Midifile.hpp.

6.100.5.4. unsigned char csound::MidiFile::lastStatus

Definition at line 217 of file Midifile.hpp.

6.100.5.5. double csound::MidiFile::microsecondsPerQuarterNote

Definition at line 216 of file Midifile.hpp.

6.100.5.6. MidiHeader csound::MidiFile::midiHeader

Definition at line 218 of file Midifile.hpp.

6.100.5.7. std::vector<MidiTrack> csound::MidiFile::midiTracks

Definition at line 220 of file Midifile.hpp.

6.100.5.8. TempoMap csound::MidiFile::tempoMap

Definition at line 219 of file Midifile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp

6.101. midiglobals Struct Reference

`#include <csoundCore.h>`

Public Attributes

- MEVENT * Midevtblk
- int sexp
- int MIDIoutDONE
- int MIDIINbufIndex
- MIDIMESSAGE MIDIINbuffer2 [MIDIINBUFMAX]
- int(* MidiInOpenCallback)(void *, void **, const char *)
- int(* MidiReadCallback)(void *, void *, unsigned char *, int)
- int(* MidiInCloseCallback)(void *, void *)
- int(* MidiOutOpenCallback)(void *, void **, const char *)
- int(* MidiWriteCallback)(void *, void *, unsigned char *, int)
- int(* MidiOutCloseCallback)(void *, void *)
- char *(* MidiErrorStringCallback)(int)
- void * midiInUserData
- void * midiOutUserData
- void * midiFileData
- int rawControllerMode
- char muteTrackList [256]
- unsigned char mbuf [MBUFSIZ]
- unsigned char * bufp
- unsigned char * endatp
- short datreq
- short datcnt

6.101.1. Member Data Documentation

6.101.1.1. unsigned char* midiglobals::bufp

Definition at line 489 of file csoundCore.h.

6.101.1.2. short midiglobals::datcnt

Definition at line 490 of file csoundCore.h.

6.101.1.3. short midiglobals::datreq

Definition at line 490 of file csoundCore.h.

6.101.1.4. unsigned char * midiglobals::endatp

Definition at line 489 of file csoundCore.h.

6.101.1.5. unsigned char midiglobals::mbuf[MBUFSIZ]

Definition at line 488 of file csoundCore.h.

6.101.1.6. MEVENT* midiglobals::Midevtblk

Definition at line 471 of file csoundCore.h.

6.101.1.7. char*(* midiglobals::MidiErrorStringCallback)(int)

6.101.1.8. void* midiglobals::midiFileData

Definition at line 485 of file csoundCore.h.

6.101.1.9. MIDIMESSAGE midiglobals::MIDIINbuffer2[MIDIINBUFMAX]

Definition at line 475 of file csoundCore.h.

6.101.1.10. int midiglobals::MIDIINbufIndex

Definition at line 474 of file csoundCore.h.

6.101.1.11. int(* midiglobals::MidiInCloseCallback)(void *, void *)

6.101.1.12. int(* midiglobals::MidiInOpenCallback)(void *, void **, const char *)

6.101.1.13. void* midiglobals::midiInUserData

Definition at line 483 of file csoundCore.h.

6.101.1.14. int(* midiglobals::MidiOutCloseCallback)(void *, void *)

6.101.1.15. int midiglobals::MIDIoutDONE

Definition at line 473 of file csoundCore.h.

6.101.1.16. int(* midiglobals::MidiOutOpenCallback)(void *, void **, const char *)

6.101.1.17. void* midiglobals::midiOutUserData

Definition at line 484 of file csoundCore.h.

6.101.1.18. int(* midiglobals::MidiReadCallback)(void *, void *, unsigned char *, int)

6.101.1.19. int(* midiglobals::MidiWriteCallback)(void *, void *, unsigned char *, int)

6.101.1.20. char midiglobals::muteTrackList[256]

Definition at line 487 of file csoundCore.h.

6.101.1.21. int midiglobals::rawControllerMode

Definition at line 486 of file csoundCore.h.

6.101.1.22. int midiglobals::sexp

Definition at line 472 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.102. csound::MidiHeader Class Reference

`#include <Midifile.hpp>`

Inheritance diagram for csound::MidiHeader::

Public Member Functions

- MidiHeader (void)
- virtual ~MidiHeader (void)
- virtual void clear (void)
- virtual void read (std::istream &stream)
- virtual void write (std::ostream &stream)
- virtual void markChunkSize (std::ostream &stream)
- virtual void markChunkStart (std::ostream &stream)
- virtual void markChunkEnd (std::ostream &stream)

Public Attributes

- short type
- short trackCount
- short timeFormat
- int id
- int chunkSize
- int chunkSizePosition
- int chunkStart
- int chunkEnd

6.102.1. Constructor & Destructor Documentation

6.102.1.1. csound::MidiHeader::MidiHeader (void)

6.102.1.2. virtual csound::MidiHeader::~MidiHeader (void) [virtual]

6.102.2. Member Function Documentation

6.102.2.1. virtual void csound::MidiHeader::clear (void) [virtual]

6.102.2.2. virtual void csound::Chunk::markChunkEnd (std::ostream & *stream*) [virtual, inherited]

6.102.2.3. virtual void csound::Chunk::markChunkSize (std::ostream & *stream*) [virtual, inherited]

6.102.2.4. virtual void csound::Chunk::markChunkStart (std::ostream & *stream*) [virtual, inherited]

6.102.2.5. virtual void csound::MidiHeader::read (std::istream & *stream*) [virtual]

Reimplemented from csound::Chunk.

6.102.2.6. virtual void csound::MidiHeader::write (std::ostream & *stream*) [virtual]

Reimplemented from csound::Chunk.

6.102.3. Member Data Documentation

6.102.3.1. int csound::Chunk::chunkEnd [inherited]

Definition at line 60 of file Midifile.hpp.

6.102.3.2. int csound::Chunk::chunkSize [inherited]

Definition at line 57 of file Midifile.hpp.

6.102.3.3. int csound::Chunk::chunkSizePosition [inherited]

Definition at line 58 of file Midifile.hpp.

6.102.3.4. int csound::Chunk::chunkStart [inherited]

Definition at line 59 of file Midifile.hpp.

6.102.3.5. int csound::Chunk::id [inherited]

Definition at line 56 of file Midifile.hpp.

6.102.3.6. short csound::MidiHeader::timeFormat

Definition at line 75 of file Midifile.hpp.

6.102.3.7. short csound::MidiHeader::trackCount

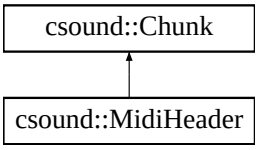
Definition at line 74 of file Midifile.hpp.

6.102.3.8. short csound::MidiHeader::type

Definition at line 73 of file Midifile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp


csound::Chunk
csound::MidiHeader

6.103. MIDIMESSAGE Union Reference

`#include <csoundCore.h>`

Public Attributes

- unsigned long dwData
- unsigned char bData [4]

6.103.1. Member Data Documentation

6.103.1.1. unsigned char MIDIMESSAGE::bData[4]

Definition at line 460 of file csoundCore.h.

6.103.1.2. unsigned long MIDIMESSAGE::dwData

Definition at line 459 of file csoundCore.h.

The documentation for this union was generated from the following file:

- H/csoundCore.h

6.104. csound::MidiTrack Class Reference

`#include <Midifile.hpp>`

Inheritance diagram for csound::MidiTrack::

Public Member Functions

- MidiTrack (void)
- virtual ~MidiTrack (void)
- virtual void read (std::istream &stream, MidiFile &midiFile)
- virtual void write (std::ostream &stream, MidiFile &midiFile)
- virtual void sort (void)
- virtual void read (std::istream &stream)
- virtual void write (std::ostream &stream)
- virtual void markChunkSize (std::ostream &stream)
- virtual void markChunkStart (std::ostream &stream)
- virtual void markChunkEnd (std::ostream &stream)

Public Attributes

- int id
- int chunkSize
- int chunkSizePosition
- int chunkStart
- int chunkEnd

6.104.1. Constructor & Destructor Documentation

6.104.1.1. csound::MidiTrack::MidiTrack (void)

6.104.1.2. virtual csound::MidiTrack::~MidiTrack (void) [virtual]

6.104.2. Member Function Documentation

6.104.2.1. virtual void csound::Chunk::markChunkEnd (std::ostream & *stream*) [virtual, inherited]

6.104.2.2. virtual void csound::Chunk::markChunkSize (std::ostream & *stream*) [virtual, inherited]

6.104.2.3. virtual void csound::Chunk::markChunkStart (std::ostream & *stream*) [virtual, inherited]

6.104.2.4. virtual void csound::Chunk::read (std::istream & *stream*) [virtual, inherited]

Reimplemented in csound::MidiHeader.

6.104.2.5. virtual void csound::MidiTrack::read (std::istream & *stream*, MidiFile & *midiFile*) [virtual]

6.104.2.6. virtual void csound::MidiTrack::sort (void) [virtual]

6.104.2.7. virtual void csound::Chunk::write (std::ostream & *stream*) [virtual, inherited]

Reimplemented in csound::MidiHeader.

6.104.2.8. virtual void csound::MidiTrack::write (std::ostream & *stream*, MidiFile & *midiFile*) [virtual]

6.104.3. Member Data Documentation

6.104.3.1. int csound::Chunk::chunkEnd [inherited]

Definition at line 60 of file Midifile.hpp.

6.104.3.2. int csound::Chunk::chunkSize [inherited]

Definition at line 57 of file Midifile.hpp.

6.104.3.3. int csound::Chunk::chunkSizePosition [inherited]

Definition at line 58 of file Midifile.hpp.

6.104.3.4. int csound::Chunk::chunkStart [inherited]

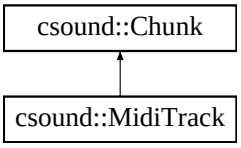
Definition at line 59 of file Midifile.hpp.

6.104.3.5. int csound::Chunk::id [inherited]

Definition at line 56 of file Midifile.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp


csound::Chunk
csound::MidiTrack

6.105. monblk Struct Reference

`#include <csoundCore.h>`

Public Attributes

- short pch
- monblk * prv

6.105.1. Member Data Documentation

6.105.1.1. short monblk::pch

Definition at line 230 of file csoundCore.h.

6.105.1.2. struct monblk* monblk::prv

Definition at line 231 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.106. Loris::Morpher Class Reference

`#include <Morpher.h>`

6.106.1. Detailed Description

Class Morpher performs sound morphing and Partial parameter envelope interpolation according to a trio of frequency, amplitude, and bandwidth morphing functions, described by Envelopes. Sound morphing is achieved by interpolating the time-varying frequencies, amplitudes, and bandwidths of corresponding partials obtained from reassigned bandwidth-enhanced analysis of the source and target sounds. Partial correspondences may be established by labeling, using instances of the Channelizer and Distiller classes.

The Morpher collects morphed Partials in a PartialList, that is accessible to clients.

For more information about sound morphing using the Reassigned Bandwidth-Enhanced Additive Sound Model, refer to the Loris website: www.cerlsoundgroup.org/Loris/.

Morpher is a leaf class, do not subclass.

Definition at line 67 of file Morpher.h.

Public Member Functions

- Morpher (const Envelope &f)
- Morpher (const Envelope &ff, const Envelope &af, const Envelope &bwf)
- Morpher (const Morpher &rhs)
- ~Morpher (void)
- Morpher & operator= (const Morpher &rhs)
- Partial morphPartial (const Partial &src, const Partial &tgt, int assignLabel)
- void morph (PartialList::const_iterator beginSrc, PartialList::const_iterator endSrc, PartialList::const_iterator beginTgt, PartialList::const_iterator endTgt)
- void crossfade (PartialList::const_iterator beginSrc, PartialList::const_iterator endSrc, PartialList::const_iterator beginTgt, PartialList::const_iterator endTgt, Partial::label_type label=0)
- Breakpoint morphBreakpoints (const Breakpoint &srcBkpt, const Breakpoint &tgtBkpt, double time) const
- Breakpoint morphSrcBreakpoint (const Breakpoint &bp, const Partial &tgtPartial, double time) const
- Breakpoint morphTgtBreakpoint (const Breakpoint &bp, const Partial &tgtPartial, double time) const
- Breakpoint fadeSrcBreakpoint (Breakpoint bp, double time) const
- Breakpoint fadeTgtBreakpoint (Breakpoint bp, double time) const
- void setFrequencyFunction (const Envelope &f)
- void setAmplitudeFunction (const Envelope &f)
- void setBandwidthFunction (const Envelope &f)
- const Envelope & frequencyFunction (void) const
- const Envelope & amplitudeFunction (void) const
- const Envelope & bandwidthFunction (void) const
- double amplitudeShape (void) const
- void setAmplitudeShape (double x)
- double minBreakpointGap (void) const
- void setMinBreakpointGap (double x)
- Partial::label_type sourceReferenceLabel (void) const

- Partial::label_type targetReferenceLabel (void) const
- void setSourceReferenceLabel (Partial::label_type l)
- void setTargetReferenceLabel (Partial::label_type l)
- PartialList & partials (void)
- const PartialList & partials (void) const

Private Types

- typedef std::map< Partial::label_type, PartialPtrPair > PartialCorrespondence

Private Member Functions

- void morph_aux (PartialCorrespondence &correspondence)
- Partial makePartialFromReference (Partial scaleMe, double fscale)
- void appendMorphedSrc (const Breakpoint &srcBkpt, const Partial &tgtPartial, double time, Partial &newp)
- void appendMorphedTgt (const Breakpoint &tgtBkpt, const Partial &srcPartial, double time, Partial &newp)

Private Attributes

- std::auto_ptr< Envelope > _freqFunction
- std::auto_ptr< Envelope > _ampFunction
- std::auto_ptr< Envelope > _bwFunction
- PartialList _partials
- Partial::label_type _refLabel0
- Partial::label_type _refLabel1
- double _freqFixThresholdDb
- double _ampMorphShape
- double _minBreakpointGapSec

Classes

- struct PartialPtrPair

6.106.2. Member Typedef Documentation

6.106.2.1. typedef std::map< Partial::label_type, PartialPtrPair > Loris::Morpher::PartialCorrespondence [private]

Definition at line 436 of file Morpher.h.

6.106.3. Constructor & Destructor Documentation

6.106.3.1. Loris::Morpher::Morpher (const Envelope & *f*)

Construct a new Morpher using the same morphing envelope for frequency, amplitude, and bandwidth (noisiness).

Parameters:

f is the Envelope to clone for all three morphing functions.

6.106.3.2. Loris::Morpher::Morpher (const Envelope & *ff*, const Envelope & *af*, const Envelope & *bwf*)

Construct a new Morpher using the specified morphing envelopes for frequency, amplitude, and bandwidth (noisiness).

Parameters:

ff is the Envelope to clone for the frequency morphing function

af is the Envelope to clone for the amplitude morphing function

bwf is the Envelope to clone for the bandwidth morphing function

6.106.3.3. Loris::Morpher::Morpher (const Morpher & *rhs*)

Construct a new Morpher that is a duplicate of rhs.

Parameters:

rhs is the Morpher to duplicate

6.106.3.4. Loris::Morpher::~Morpher (void)

Destroy this Morpher.

6.106.4. Member Function Documentation

6.106.4.1. const Envelope& Loris::Morpher::amplitudeFunction (void) const

Return a reference to this Morpher's amplitude morphing envelope.

6.106.4.2. double Loris::Morpher::amplitudeShape (void) const

Return the shaping parameter for the amplitude moprhing function (only used in new log-amplitude morphing). This shaping parameter controls the slope of the amplitude morphing function, for values greater than 1, this function gets nearly linear (like the old amplitude morphing function), for values much less than 1 (e.g. 1E-5) the slope is gently curved and sounds pretty "linear", for very small values (e.g. 1E-12) the curve is very steep and sounds un-natural because of the huge jump from zero amplitude to very small amplitude.

6.106.4.3. void Loris::Morpher::appendMorphedSrc (const Breakpoint & *srcBkpt*, const Partial & *tgtPartial*, double *time*, Partial & *newp*) [private]

Compute morphed parameter values at the specified time, using the source Breakpoint (assumed to correspond exactly to the specified time) and the target Partial (whose parameters are examined at the specified time). Append the morphed Breakpoint to newp only if the target should contribute to the morph at the specified time.

Precondition:

the target Partial may not be a dummy Partial (no Breakpoints).

Parameters:

srcBkpt is the Breakpoint corresponding to a morph function value of 0.

tgtPartial is the Partial corresponding to a morph function value of 1, evaluated at the specified time.

time is the time corresponding to srcBkpt (used to evaluate the morphing functions and tgtPartial).

newp is the morphed Partial under construction, the morphed Breakpoint is added to this Partial.

6.106.4.4. void Loris::Morpher::appendMorphedTgt (const Breakpoint & *tgtBkpt*, const Partial & *srcPartial*, double *time*, Partial & *newp*) [private]

Compute morphed parameter values at the specified time, using the target Breakpoint (assumed to correspond exactly to the specified time) and the source Partial (whose parameters are examined at the specified time). Append the morphed Breakpoint to newp only if the target should contribute to the morph at the specified time.

Precondition:
the source Partial may not be a dummy Partial (no Breakpoints).

Parameters:
tgtBkpt is the Breakpoint corresponding to a morph function value of 1.

srcPartial is the Partial corresponding to a morph function value of 0, evaluated at the specified time.

time is the time corresponding to srcBkpt (used to evaluate the morphing functions and srcPartial).

newp is the morphed Partial under construction, the morphed Breakpoint is added to this Partial.

6.106.4.5. const Envelope& Loris::Morpher::bandwidthFunction (void) const

Return a reference to this Morpher's bandwidth morphing envelope.

6.106.4.6. void Loris::Morpher::crossfade (PartialList::const_iterator *beginSrc*, PartialList::const_iterator *endSrc*, PartialList::const_iterator *beginTgt*, PartialList::const_iterator *endTgt*, Partial::label_type *label* = 0)

Crossfade Partials with no correspondences.

Unlabeled Partials (having the specified label) are considered to have no correspondences, so they are just faded out, and not actually morphed. Consistent with the morphing behavior, crossfaded Partials are thinned, if necssary, so that no two Breakpoints are closer in time than the minBreakpointGap.

The Partials in the first range are treated as components of the source sound, corresponding to a morph function value of 0, and those in the second are treated as components of the target sound, corresponding to a morph function value of 1.

The crossfaded Partials are stored in the Morpher's PartialList.

Parameters:
beginSrc is the beginning of the sequence of Partials corresponding to a morph function value of 0.

endSrc is (one past) the end of the sequence of Partials corresponding to a morph function value of 0.

beginTgt is the beginning of the sequence of Partials corresponding to a morph function value of 1.

endTgt is (one past) the end of the sequence of Partials corresponding to a morph function value of 1.

label is the label to associate with unlabeled Partials (default is 0).

6.106.4.7. Breakpoint Loris::Morpher::fadeSrcBreakpoint (Breakpoint *bp*, double *time*) const

Compute morphed parameter values at the specified time, using the source Breakpoint, assumed to correspond exactly to the specified time, and assuming that there is no corresponding target Partial, so the source Breakpoint should be simply faded.

Parameters:

bp is the Breakpoint corresponding to a morph function value of 0.

time is the time corresponding to bp (used to evaluate the morphing functions).

Returns:

the faded Breakpoint

6.106.4.8. Breakpoint Loris::Morpher::fadeTgtBreakpoint (Breakpoint *bp*, double *time*) const

Compute morphed parameter values at the specified time, using the target Breakpoint, assumed to correspond exactly to the specified time, and assuming that there is not corresponding source Partial, so the target Breakpoint should be simply faded.

Parameters:

bp is the Breakpoint corresponding to a morph function value of 1.

time is the time corresponding to bp (used to evaluate the morphing functions).

Returns:

the faded Breakpoint

6.106.4.9. const Envelope& Loris::Morpher::frequencyFunction (void) const

Return a reference to this Morpher's frequency morphing envelope.

6.106.4.10. Partial Loris::Morpher::makePartialFromReference (Partial *scaleMe*, double *fscale*) [private]

Helper function to construct a Partial for morphing by scaling the frequencies of the reference Partial. This is used when only one of the sources in a morph has a Partial with a particular label.

6.106.4.11. double Loris::Morpher::minBreakpointGap (void) const

Return the minimum time gap (secs) between two Breakpoints in the morphed Partials. Morphing two Partials can generate a third Partial having Breakpoints arbitrarily close together in time, and this makes morphs huge. Raising this threshold limits the Breakpoint density in the morphed Partials. Default is 1/10 ms.

6.106.4.12. void Loris::Morpher::morph (PartialList::const_iterator *beginSrc*, PartialList::const_iterator *endSrc*, PartialList::const_iterator *beginTgt*, PartialList::const_iterator *endTgt*)

Morph two sounds (collections of Partials labeled to indicate correspondences) into a single labeled collection of Partials. Unlabeled Partials (having label 0) are crossfaded. The morphed and crossfaded Partials are stored in the Morpher's PartialList.

The Partials in the first range are treated as components of the source sound, corresponding to a morph function value of 0, and those in the second are treated as components of the target sound, corresponding to a morph function value of 1.

See also:
crossfade, morphPartial

Parameters:

beginSrc is the beginning of the sequence of Partials corresponding to a morph function value of 0.

endSrc is (one past) the end of the sequence of Partials corresponding to a morph function value of 0.

beginTgt is the beginning of the sequence of Partials corresponding to a morph function value of 1.

endTgt is (one past) the end of the sequence of Partials corresponding to a morph function value of 1.

6.106.4.13. void Loris::Morpher::morph_aux (PartialCorrespondence & *correspondence*) [private]

Helper function that performs the morph between corresponding pairs of Partials identified in a PartialCorrespondence. Called by the morph() implementation accepting two sequences of Partials.

6.106.4.14. Breakpoint Loris::Morpher::morphBreakpoints (const Breakpoint & *srcBkpt*, const Breakpoint & *tgtBkpt*, double *time*) const

Compute morphed parameter values at the specified time, using the source and target Breakpoints (assumed to correspond exactly to the specified time).

Parameters:

srcBkpt is the Breakpoint corresponding to a morph function value of 0.

tgtBkpt is the Breakpoint corresponding to a morph function value of 1.

time is the time corresponding to srcBkpt (used to evaluate the morphing functions and tgtPartial).

Returns:
the morphed Breakpoint

6.106.4.15. Partial Loris::Morpher::morphPartial (const Partial & *src*, const Partial & *tgt*, int *assignLabel*)

Morph a pair of Partials to yield a new morphed Partial. Dummy Partials (having no Breakpoints) don't contribute to the morph, except to cause their opposite to fade out. Either (or neither) the

source or target Partial may be a dummy Partial (no Breakpoints), but not both. The morphed Partial has Breakpoints at times corresponding to every Breakpoint in both source Partials, omitting Breakpoints that would be closer than the minBreakpointGap to their predecessor. The new morphed Partial is assigned the specified label and returned.

Parameters:

src is the Partial corresponding to a morph function value of 0, evaluated at the specified time.

tgt is the Partial corresponding to a morph function value of 1, evaluated at the specified time.

assignLabel is the label assigned to the morphed Partial

Returns:

the morphed Partial

6.106.4.16. Breakpoint Loris::Morpher::morphSrcBreakpoint (const Breakpoint & *bp*, const Partial & *tgtPartial*, double *time*) const

Compute morphed parameter values at the specified time, using the source Breakpoint (assumed to correspond exactly to the specified time) and the target Partial (whose parameters are examined at the specified time).

Precondition:

the target Partial may not be a dummy Partial (no Breakpoints).

Parameters:

srcBkpt is the Breakpoint corresponding to a morph function value of 0.

tgtPartial is the Partial corresponding to a morph function value of 1, evaluated at the specified time.

time is the time corresponding to srcBkpt (used to evaluate the morphing functions and tgtPartial).

newpis the morphed Partial under construction, the morphed Breakpoint is added to this Partial.

6.106.4.17. Breakpoint Loris::Morpher::morphTgtBreakpoint (const Breakpoint & *bp*, const Partial & *tgtPartial*, double *time*) const

Compute morphed parameter values at the specified time, using the target Breakpoint (assumed to correspond exactly to the specified time) and the source Partial (whose parameters are examined at the specified time).

Precondition:

the source Partial may not be a dummy Partial (no Breakpoints).

Parameters:

tgtBkpt is the Breakpoint corresponding to a morph function value of 1.

srcPartial is the Partial corresponding to a morph function value of 0, evaluated at the specified time.

time is the time corresponding to srcBkpt (used to evaluate the morphing functions and tgtPartial).

newpis the morphed Partial under construction, the morphed Breakpoint is added to this Partial.

6.106.4.18. Morpher& Loris::Morpher::operator= (const Morpher & *rhs*)

Parameters:
rhs is the Morpher to duplicate

6.106.4.19. const PartialList& Loris::Morpher::partials (void) const

Return a const reference to this Morpher's list of morphed Partials.

6.106.4.20. PartialList& Loris::Morpher::partials (void)

Return a reference to this Morpher's list of morphed Partials.

6.106.4.21. void Loris::Morpher::setAmplitudeFunction (const Envelope & *f*)

Assign a new amplitude morphing envelope to this Morpher.

6.106.4.22. void Loris::Morpher::setAmplitudeShape (double *x*)

Set the shaping parameter for the amplitude moprhing function (only used in new log-amplitude morphing). This shaping parameter controls the slope of the amplitude morphing function, for values greater than 1, this function gets nearly linear (like the old amplitude morphing function), for values much less than 1 (e.g. 1E-5) the slope is gently curved and sounds pretty "linear", for very small values (e.g. 1E-12) the curve is very steep and sounds un-natural because of the huge jump from zero amplitude to very small amplitude.

Parameters:
x is the new shaping parameter, it must be positive.

6.106.4.23. void Loris::Morpher::setBandwidthFunction (const Envelope & *f*)

Assign a new bandwidth morphing envelope to this Morpher.

6.106.4.24. void Loris::Morpher::setFrequencyFunction (const Envelope & *f*)

Assign a new frequency morphing envelope to this Morpher.

6.106.4.25. void Loris::Morpher::setMinBreakpointGap (double *x*)

Set the minimum time gap (secs) between two Breakpoints in the morphed Partials. Morphing two Partials can generate a third Partial having Breakpoints arbitrarily close together in time, and this makes morphs huge. Raising this threshold limits the Breakpoint density in the morphed Partials. Default is 1/10 ms.

Parameters:
x is the new minimum gap in seconds, it must be positive

Exceptions:
InvalidArgument if the specified gap is not positive

6.106.4.26. void Loris::Morpher::setSourceReferenceLabel (Partial::label_type l)

Set the label of the Partial to be used as a reference Partial for the source sequence in a morph of two Partial sequences. The reference partial is used to compute frequencies for very low-amplitude Partials whose frequency estimates are not considered reliable. The reference Partial is considered to have good frequency estimates throughout. Setting the reference label to 0 indicates that no reference Partial should be used for the source sequence.

6.106.4.27. void Loris::Morpher::setTargetReferenceLabel (Partial::label_type l)

Set the label of the Partial to be used as a reference Partial for the target sequence in a morph of two Partial sequences. The reference partial is used to compute frequencies for very low-amplitude Partials whose frequency estimates are not considered reliable. The reference Partial is considered to have good frequency estimates throughout. Setting the reference label to 0 indicates that no reference Partial should be used for the target sequence.

6.106.4.28. Partial::label_type Loris::Morpher::sourceReferenceLabel (void) const

Return the label of the Partial to be used as a reference Partial for the source sequence in a morph of two Partial sequences. The reference partial is used to compute frequencies for very low-amplitude Partials whose frequency estimates are not considered reliable. The reference Partial is considered to have good frequency estimates throughout. The default label of 0 indicates that no reference Partial should be used for the source sequence.

6.106.4.29. Partial::label_type Loris::Morpher::targetReferenceLabel (void) const

Return the label of the Partial to be used as a reference Partial for the target sequence in a morph of two Partial sequences. The reference partial is used to compute frequencies for very low-amplitude Partials whose frequency estimates are not considered reliable. The reference Partial is considered to have good frequency estimates throughout. The default label of 0 indicates that no reference Partial should be used for the target sequence.

6.106.5. Member Data Documentation

6.106.5.1. std::auto_ptr< Envelope > Loris::Morpher::_ampFunction [private]

frequency morphing function

Definition at line 72 of file Morpher.h.

6.106.5.2. double Loris::Morpher::_ampMorphShape [private]

frequencies are corrected according to a reference Partial, if specified.

Definition at line 86 of file Morpher.h.

6.106.5.3. std::auto_ptr< Envelope > Loris::Morpher::_bwFunction [private]

amplitude morphing function

Definition at line 73 of file Morpher.h.

6.106.5.4. double Loris::Morpher::_freqFixThresholdDb [private]

morphing sequences of labeled Partials, default 0 implies no reference Partial

Definition at line 82 of file Morpher.h.

6.106.5.5. std::auto_ptr< Envelope > Loris::Morpher::_freqFunction [private]

Definition at line 71 of file Morpher.h.

6.106.5.6. double Loris::Morpher::_minBreakpointGapSec [private]

slope of the amplitude morphing function, for values greater than 1, this function gets nearly linear (like the old amplitude morphing function), for values much less than 1 (e.g. 1E-5) the slope is gently curved and sounds pretty "linear", for very small values (e.g. 1E-12) the curve is very steep and sounds un-natural because of the huge jump from zero amplitude to very small amplitude.

Definition at line 98 of file Morpher.h.

6.106.5.7. PartialList Loris::Morpher::_partials [private]

bandwidth morphing function

Definition at line 75 of file Morpher.h.

6.106.5.8. Partial::label_type Loris::Morpher::_refLabel0 [private]

collect Partials here

Definition at line 77 of file Morpher.h.

6.106.5.9. Partial::label_type Loris::Morpher::_refLabel1 [private]

labels of the reference Partials

Definition at line 78 of file Morpher.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Morpher.h

6.107. Loris::Morpher::PartialPtrPair Struct Reference

Public Member Functions

- PartialPtrPair (void)

Public Attributes

- const Partial * first
- const Partial * second

6.107.1. Constructor & Destructor Documentation

6.107.1.1. Loris::Morpher::PartialPtrPair::PartialPtrPair (void) [inline]

Definition at line 433 of file Morpher.h.

References first, and second.

6.107.2. Member Data Documentation

6.107.2.1. const Partial* Loris::Morpher::PartialPtrPair::first

Definition at line 431 of file Morpher.h.

Referenced by PartialPtrPair().

6.107.2.2. const Partial* Loris::Morpher::PartialPtrPair::second

Definition at line 432 of file Morpher.h.

Referenced by PartialPtrPair().

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/Morpher.h

6.108. csound::MusicModel Class Reference

`#include <MusicModel.hpp>`

Inheritance diagram for csound::MusicModel::

6.108.1. Detailed Description

Base class for compositions that use the principle of a music graph to generate a score. A music graph is a directed acyclic graph of nodes including empty nodes, nodes that contain only child nodes, score nodes, event generator nodes, event transformer nodes, and others. Each node is associated with a local transformation of coordinate system in music space using a 12 x 12 homogeneous matrix. To generate the score, the music graph is traversed depth first, and each node postconcatenates its local transformation of coordinate system with the coordinate system of its parent to derive a new local coordinate system, which is applied to all child events.

Definition at line 54 of file MusicModel.hpp.

Public Member Functions

- MusicModel ()
- virtual ~MusicModel ()
- virtual void initialize ()
- virtual void generate ()
- virtual void clear ()
- virtual std::string getFilename () const
- virtual void setFilename (std::string filename)
- virtual std::string getMidiFilename ()
- virtual std::string getOutputSoundfileName ()
- virtual long getThis ()
- virtual Node * getThisNode ()
- virtual void createCsoundScore (std::string addToScore="")
- virtual void render ()
- virtual void perform ()
- virtual Score & getScore ()
- virtual void setCppSound (CppSound *orchestra)
- virtual CppSound * getCppSound ()
- virtual void write (const char *text)
- virtual void setTonesPerOctave (double tonesPerOctave)
- virtual double getTonesPerOctave () const
- virtual void setConformPitches (bool conformPitches)
- virtual bool getConformPitches () const
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &globalCoordinates, Score &score)
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Static Public Member Functions

- static std::string generateFilename ()

Public Attributes

- std::vector< Node * > children

Protected Attributes

- Score score
- double tonesPerOctave
- bool conformPitches
- CppSound cppSound_
- CppSound * cppSound
- ublas::matrix< double > localCoordinates

Private Attributes

- std::string filename

6.108.2. Constructor & Destructor Documentation

6.108.2.1. csound::MusicModel::MusicModel ()

6.108.2.2. virtual csound::MusicModel::~MusicModel () [virtual]

6.108.3. Member Function Documentation

6.108.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.108.3.2. virtual void csound::MusicModel::clear () [virtual]

Clear all contents of this. Probably should be overridden in derived classes.

Reimplemented from csound::Composition.

6.108.3.3. virtual void csound::Composition::createCsoundScore (std::string *addToScore* = "") [virtual, inherited]

Translate the generated score to a Csound score and export it for performance.

6.108.3.4. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.108.3.5. virtual void csound::MusicModel::generate () [virtual]

Generate performance events and store them in the score. Must be overidden in derived classes.

Reimplemented from csound::Composition.

6.108.3.6. static std::string csound::MusicModel::generateFilename () [static]

6.108.3.7. virtual bool csound::Composition::getConformPitches () const [virtual, inherited]

6.108.3.8. virtual CppSound* csound::Composition::getCppSound () [virtual, inherited]

Return the self-contained Orchestra.

6.108.3.9. virtual std::string csound::MusicModel::getFilename () const [virtual]

6.108.3.10. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.108.3.11. virtual std::string csound::MusicModel::getMidiFilename () [virtual]

6.108.3.12. virtual std::string csound::MusicModel::getOutputSoundfileName () [virtual]

6.108.3.13. virtual Score& csound::Composition::getScore () [virtual, inherited]

Return the self-contained Score.

6.108.3.14. virtual long csound::MusicModel::getThis () [virtual]

6.108.3.15. virtual Node* csound::MusicModel::getThisNode () [virtual]

6.108.3.16. virtual double csound::Composition::getTonesPerOctave () const [virtual, inherited]

6.108.3.17. virtual void csound::MusicModel::initialize () [virtual]

6.108.3.18. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.108.3.19. virtual void csound::Composition::perform () [virtual, inherited]

Uses csound to perform the current score.

6.108.3.20. virtual void csound::Node::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual, inherited]

The default implementation does nothing.

Reimplemented in csound::Cell, csound::CounterpointNode, csound::Hocket, csound::MCRM, csound::Random, csound::Rescale, and csound::ScoreNode.

6.108.3.21. virtual void csound::Composition::render () [virtual, inherited]

Convenience function that erases the existing score, appends optional text to it, invokes generate(), invokes createCsoundScore(), and invokes perform().

6.108.3.22. virtual void csound::Composition::setConformPitches (bool *conformPitches*) [virtual, inherited]

6.108.3.23. virtual void csound::Composition::setCppSound (CppSound * *orchestra*) [virtual, inherited]

Sets the self-contained Orchestra.

6.108.3.24. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.108.3.25. virtual void csound::MusicModel::setFilename (std::string *filename*) [virtual]

6.108.3.26. virtual void csound::Composition::setTonesPerOctave (double *tonesPerOctave*) [virtual, inherited]

6.108.3.27. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.108.3.28. virtual void csound::Composition::write (const char * *text*) [virtual, inherited]

Write as if to stdout or stderr.

6.108.4. Member Data Documentation

6.108.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.108.4.2. bool csound::Composition::conformPitches [protected, inherited]

Definition at line 48 of file Composition.hpp.

6.108.4.3. CppSound* csound::Composition::cppSound [protected, inherited]

Definition at line 50 of file Composition.hpp.

6.108.4.4. CppSound csound::Composition::cppSound_ [protected, inherited]

Definition at line 49 of file Composition.hpp.

6.108.4.5. std::string csound::MusicModel::filename [private]

Definition at line 58 of file MusicModel.hpp.

6.108.4.6. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.108.4.7. Score csound::Composition::score [protected, inherited]

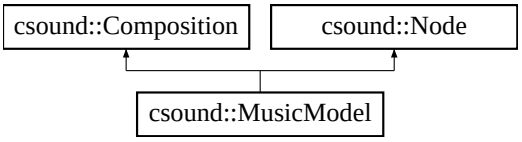
Definition at line 46 of file Composition.hpp.

6.108.4.8. double csound::Composition::tonesPerOctave [protected, inherited]

Definition at line 47 of file Composition.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/MusicModel.hpp


csound::Composition
csound::Node
csound::MusicModel

6.109. names Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char * mac
- names * next

6.109.1. Member Data Documentation

6.109.1.1. char* names::mac

Definition at line 499 of file csoundCore.h.

6.109.1.2. struct names* names::next

Definition at line 500 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.110. NGFENS Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char * word
- void(* fn)(void)

6.110.1. Member Data Documentation

6.110.1.1. void(* NGFENS::fn)(void)

6.110.1.2. char* NGFENS::word

Definition at line 334 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.111. csound::Node Class Reference

`#include <Node.hpp>`

Inheritance diagram for csound::Node::

6.111.1. Detailed Description

Base class for all music graph nodes in the Silence system. Nodes can transform silence::Events produced by child nodes. Nodes can produce silence::Events.

Definition at line 49 of file Node.hpp.

Public Member Functions

- Node ()
- virtual ~Node ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::vector< Node * > children

Protected Attributes

- ublas::matrix< double > localCoordinates

6.111.2. Constructor & Destructor Documentation

6.111.2.1. csound::Node::Node ()

6.111.2.2. virtual csound::Node::~Node () [virtual]

6.111.3. Member Function Documentation

6.111.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual]

6.111.3.2. virtual void csound::Node::clear () [virtual]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.111.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual]

6.111.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.111.3.5. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual]

6.111.3.6. virtual void csound::Node::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented in csound::Cell, csound::CounterpointNode, csound::Hocket, csound::MCRM, csound::Random, csound::Rescale, and csound::ScoreNode.

6.111.3.7. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual]

6.111.3.8. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.111.4. Member Data Documentation

6.111.4.1. std::vector<Node *> csound::Node::children

Child Nodes, if any.

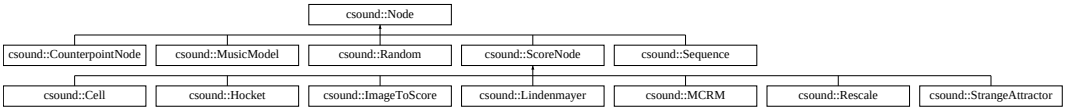
Definition at line 57 of file Node.hpp.

6.111.4.2. ublas::matrix<double> csound::Node::localCoordinates [protected]

Definition at line 52 of file Node.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Node.hpp


csound::Node
csound::CounterpointNode
csound::MusicModel
csound::Random
csound::ScoreNode
csound::Sequence
csound::Cell
csound::Hocket
csound::ImageToScore
csound::Lindenmayer
csound::MCRM
csound::Rescale
csound::StrangeAttractor

6.112. Loris::NoiseGenerator Class Reference

`#include <NoiseGenerator.h>`

Public Member Functions

- NoiseGenerator (double initSeed=1.0)
- NoiseGenerator (const Filter &f, double initSeed=1.0)
- void reset (double newSeed)
- double current (void) const
- double next (void)
- double operator() (void)
- double next (double mean, double stddev=1.)
- double operator() (double mean, double stddev=1.)

Private Member Functions

- double uniform (void)
- double gaussian_normal (void)

Private Attributes

- double sample
- Filter filter
- double u_seed
- double gset
- bool iset

6.112.1. Constructor & Destructor Documentation

6.112.1.1. Loris::NoiseGenerator::NoiseGenerator (double *initSeed* = 1.0) [explicit]

6.112.1.2. Loris::NoiseGenerator::NoiseGenerator (const Filter & *f*, double *initSeed* = 1.0)

6.112.2. Member Function Documentation

6.112.2.1. double Loris::NoiseGenerator::current (void) const [inline]

Definition at line 59 of file NoiseGenerator.h.

References sample.

6.112.2.2. double Loris::NoiseGenerator::gaussian_normal (void) [inline, private]

6.112.2.3. double Loris::NoiseGenerator::next (double *mean*, double *stddev* = 1.)

6.112.2.4. double Loris::NoiseGenerator::next (void)

Referenced by operator()().

6.112.2.5. double Loris::NoiseGenerator::operator() (double *mean*, double *stddev* = 1.) [inline]

Definition at line 67 of file NoiseGenerator.h.

References next().

6.112.2.6. double Loris::NoiseGenerator::operator() (void) [inline]

Definition at line 64 of file NoiseGenerator.h.

References next().

6.112.2.7. void Loris::NoiseGenerator::reset (double *newSeed*)

6.112.2.8. double Loris::NoiseGenerator::uniform (void) [inline, private]

6.112.3. Member Data Documentation

6.112.3.1. Filter Loris::NoiseGenerator::filter [private]

Definition at line 77 of file NoiseGenerator.h.

6.112.3.2. double Loris::NoiseGenerator::gset [private]

Definition at line 81 of file NoiseGenerator.h.

6.112.3.3. bool Loris::NoiseGenerator::iset [private]

Definition at line 82 of file NoiseGenerator.h.

6.112.3.4. double Loris::NoiseGenerator::sample [private]

Definition at line 76 of file NoiseGenerator.h.

Referenced by current().

6.112.3.5. double Loris::NoiseGenerator::u_seed [private]

Definition at line 80 of file NoiseGenerator.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/NoiseGenerator.h

6.113. Loris::PartialUtils::NoiseRatioScaler Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::NoiseRatioScaler::

Public Member Functions

- NoiseRatioScaler (double x)
- NoiseRatioScaler (const Envelope &e)
- void operator() (Partial &p) const

Protected Attributes

- Envelope * env

6.113.1. Constructor & Destructor Documentation

6.113.1.1. Loris::PartialUtils::NoiseRatioScaler::NoiseRatioScaler (double *x*) [inline]

Definition at line 258 of file PartialUtils.h.

6.113.1.2. Loris::PartialUtils::NoiseRatioScaler::NoiseRatioScaler (const Envelope & *e*) [inline]

Definition at line 259 of file PartialUtils.h.

6.113.2. Member Function Documentation

6.113.2.1. void Loris::PartialUtils::NoiseRatioScaler::operator() (Partial & *p*) const [virtual]

Function call operator: apply a mutation factor to the specified Partial. Derived classes must implement this member.

Implements Loris::PartialUtils::PartialMutator.

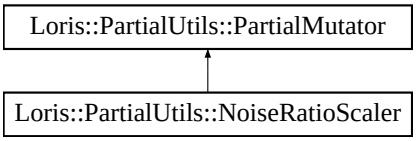
6.113.3. Member Data Documentation

6.113.3.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected, inherited]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::NoiseRatioScaler

6.114. OCTDAT Struct Reference

`#include <csoundCore.h>`

Public Attributes

- MYFLT * begp
- MYFLT * curp
- MYFLT * endp
- MYFLT feedback [6]
- long scount

6.114.1. Member Data Documentation

6.114.1.1. MYFLT* OCTDAT::begp

Definition at line 339 of file csoundCore.h.

6.114.1.2. MYFLT * OCTDAT::curp

Definition at line 339 of file csoundCore.h.

6.114.1.3. MYFLT * OCTDAT::endp

Definition at line 339 of file csoundCore.h.

6.114.1.4. MYFLT OCTDAT::feedback[6]

Definition at line 339 of file csoundCore.h.

6.114.1.5. long OCTDAT::scount

Definition at line 340 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.115. oentry Struct Reference

`#include <csound.h>`

Public Attributes

- char * opname
- unsigned short dsblksiz
- unsigned short thread
- char * outypes
- char * intypes
- int(* iopadr)(void *csound, void *p)
- int(* kopadr)(void *csound, void *p)
- int(* aopadr)(void *csound, void *p)
- void * useropinfo
- int prvnum

6.115.1. Member Data Documentation

6.115.1.1. int(* oentry::aopadr)(void *csound, void *p)

6.115.1.2. unsigned short oentry::dsblksiz

Definition at line 676 of file csound.h.

6.115.1.3. char* oentry::intypes

Definition at line 679 of file csound.h.

6.115.1.4. int(* oentry::iopadr)(void *csound, void *p)

6.115.1.5. int(* oentry::kopadr)(void *csound, void *p)

6.115.1.6. char* oentry::opname

Definition at line 675 of file csound.h.

6.115.1.7. char* oentry::outypes

Definition at line 678 of file csound.h.

6.115.1.8. int oentry::prvnum

Definition at line 684 of file csound.h.

6.115.1.9. unsigned short oentry::thread

Definition at line 677 of file csound.h.

6.115.1.10. void* oentry::useropinfo

Definition at line 683 of file csound.h.

The documentation for this struct was generated from the following file:

- H/csound.h

6.116. op Struct Reference

`#include <csoundCore.h>`

Public Attributes

- op * nxtop
- TEXT t

6.116.1. Member Data Documentation

6.116.1.1. struct op* op::nxtop

Definition at line 214 of file csoundCore.h.

6.116.1.2. TEXT op::t

Definition at line 215 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.117. OPARMS Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int odebug
- int sfread
- int sfwrite
- int sfheader
- int filetyp
- int inbufsamps
- int outbufsamps
- int informat
- int outformat
- int insampsiz
- int sfsampsize
- int displays
- int graphsoff
- int postscript
- int msglevel
- int Beatmode
- int cmdTempo
- int oMaxLag
- int usingcscore
- int Linein
- int RTevents
- int Midiin
- int FMidiin
- int ringbell
- int termifend
- int stdoutfd
- int rewrt_hdr
- int heartbeat
- int gen01defer
- int expr_opt
- long sr_override
- long kr_override
- long instxtcount
- long optxtsize
- long poolcount
- long gblfixed
- long gblacount
- long gblscount
- long argoffsize
- long filnamsize
- char * argoffspace
- char * filnamspace
- char * infilename
- char * outfilename
- char * playscore
- char * Linename
- char * Midiname
- char * FMidiname
- char * Midioutname

6.117.1. Member Data Documentation

6.117.1.1. long OPARMS::argoffsize

Definition at line 137 of file csoundCore.h.

6.117.1.2. char* OPARMS::argoffspace

Definition at line 138 of file csoundCore.h.

6.117.1.3. int OPARMS::Beatmode

Definition at line 128 of file csoundCore.h.

6.117.1.4. int OPARMS::cmdTempo

Definition at line 128 of file csoundCore.h.

6.117.1.5. int OPARMS::displays

Definition at line 127 of file csoundCore.h.

6.117.1.6. int OPARMS::expr_opt

Definition at line 133 of file csoundCore.h.

6.117.1.7. int OPARMS::filetyp

Definition at line 123 of file csoundCore.h.

6.117.1.8. long OPARMS::filnamsize

Definition at line 137 of file csoundCore.h.

6.117.1.9. char * OPARMS::filnamspace

Definition at line 138 of file csoundCore.h.

6.117.1.10. int OPARMS::FMidiin

Definition at line 130 of file csoundCore.h.

6.117.1.11. char * OPARMS::FMidiname

Definition at line 140 of file csoundCore.h.

6.117.1.12. long OPARMS::gblacount

Definition at line 136 of file csoundCore.h.

6.117.1.13. long OPARMS::gblfixed

Definition at line 136 of file csoundCore.h.

6.117.1.14. long OPARMS::gblscount

Definition at line 136 of file csoundCore.h.

6.117.1.15. int OPARMS::gen01defer

Definition at line 132 of file csoundCore.h.

6.117.1.16. int OPARMS::graphsoff

Definition at line 127 of file csoundCore.h.

6.117.1.17. int OPARMS::heartbeat

Definition at line 132 of file csoundCore.h.

6.117.1.18. int OPARMS::inbufsamps

Definition at line 124 of file csoundCore.h.

6.117.1.19. char* OPARMS::infilename

Definition at line 139 of file csoundCore.h.

6.117.1.20. int OPARMS::informat

Definition at line 125 of file csoundCore.h.

6.117.1.21. int OPARMS::insampsiz

Definition at line 126 of file csoundCore.h.

6.117.1.22. long OPARMS::instxtcount

Definition at line 135 of file csoundCore.h.

6.117.1.23. long OPARMS::kr_override

Definition at line 134 of file csoundCore.h.

6.117.1.24. int OPARMS::Linein

Definition at line 129 of file csoundCore.h.

6.117.1.25. char* OPARMS::Linename

Definition at line 140 of file csoundCore.h.

6.117.1.26. int OPARMS::Midiin

Definition at line 130 of file csoundCore.h.

6.117.1.27. char * OPARMS::Midiname

Definition at line 140 of file csoundCore.h.

6.117.1.28. char* OPARMS::Midioutname

Definition at line 141 of file csoundCore.h.

6.117.1.29. int OPARMS::msglevel

Definition at line 127 of file csoundCore.h.

6.117.1.30. int OPARMS::odebug

Definition at line 122 of file csoundCore.h.

6.117.1.31. int OPARMS::oMaxLag

Definition at line 128 of file csoundCore.h.

6.117.1.32. long OPARMS::optxtsize

Definition at line 135 of file csoundCore.h.

6.117.1.33. int OPARMS::outbufsamps

Definition at line 124 of file csoundCore.h.

6.117.1.34. char * OPARMS::outfilename

Definition at line 139 of file csoundCore.h.

6.117.1.35. int OPARMS::outformat

Definition at line 125 of file csoundCore.h.

6.117.1.36. char * OPARMS::playscore

Definition at line 139 of file csoundCore.h.

6.117.1.37. long OPARMS::poolcount

Definition at line 136 of file csoundCore.h.

6.117.1.38. int OPARMS::postscript

Definition at line 127 of file csoundCore.h.

6.117.1.39. int OPARMS::rewrt_hdr

Definition at line 132 of file csoundCore.h.

6.117.1.40. int OPARMS::ringbell

Definition at line 131 of file csoundCore.h.

6.117.1.41. int OPARMS::RTevents

Definition at line 130 of file csoundCore.h.

6.117.1.42. int OPARMS::sfheader

Definition at line 123 of file csoundCore.h.

6.117.1.43. int OPARMS::sfread

Definition at line 123 of file csoundCore.h.

6.117.1.44. int OPARMS::sfsampsize

Definition at line 126 of file csoundCore.h.

6.117.1.45. int OPARMS::sfwrite

Definition at line 123 of file csoundCore.h.

6.117.1.46. long OPARMS::sr_override

Definition at line 134 of file csoundCore.h.

6.117.1.47. int OPARMS::stdoutfd

Definition at line 131 of file csoundCore.h.

6.117.1.48. int OPARMS::termifend

Definition at line 131 of file csoundCore.h.

6.117.1.49. int OPARMS::usingcscore

Definition at line 129 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.118. OpcodeBase< T > Class Template Reference

`#include <OpcodeBase.hpp>`

6.118.1. Detailed Description

template<typename T> class OpcodeBase< T >

Template base class, or pseudo-virtual base class, for writing Csound opcodes in C++. Derive opcode implementation classes like this:

DerivedClass : public OpcodeBase<DerivedClass> { public: // All output fields must be declared first as MYFLT *: MYFLT *aret1; // All input fields must be declared next as MYFLT *: MYFLT *iarg1; MYFLT *karg2; MYFLT *aarg3; // All internal state variables must be declared after that: size_t state1; double state2; MYFLT state3; // Declare and implement only whichever of these are required: void init(); void kontrol(); void audio; void noteoff(); void deinit(); };

Definition at line 38 of file OpcodeBase.hpp.

Public Member Functions

- int init (ENVIRON *csound)
- int kontrol (ENVIRON *csound)
- int audio (ENVIRON *csound)
- int noteoff (ENVIRON *csound)
- void log (ENVIRON *csound, const char *format,...)
- void warn (ENVIRON *csound, const char *format,...)

Static Public Member Functions

- static int init_ (void *csound_, void *opcode_)
- static int kontrol_ (void *csound, void *opcode)
- static int audio_ (void *csound, void *opcode)
- static int noteoff_ (void *csound, void *opcode)

Public Attributes

- OPDS h

6.118.2. Member Function Documentation

6.118.2.1. template<typename T> int OpcodeBase< T >::audio (ENVIRON * *csound*) [inline]

Definition at line 62 of file OpcodeBase.hpp.

References NOTOK.

6.118.2.2. template<typename T> static int OpcodeBase< T >::audio_ (void * *csound*, void * *opcode*) [inline, static]

Definition at line 66 of file OpcodeBase.hpp.

6.118.2.3. template<typename T> int OpcodeBase< T >::init (ENVIRON * csound) [inline]

Definition at line 41 of file OpcodeBase.hpp.

References NOTOK.

6.118.2.4. template<typename T> static int OpcodeBase< T >::init_ (void * csound_, void * opcode_) [inline, static]

Definition at line 45 of file OpcodeBase.hpp.

References OpcodeBase< T >::noteoff_(), ENVIRON_::RegisterDeinitCallback, ENVIRON_::reinitflag, and ENVIRON_::tieflag.

6.118.2.5. template<typename T> int OpcodeBase< T >::kontrol (ENVIRON * csound) [inline]

Definition at line 54 of file OpcodeBase.hpp.

References NOTOK.

6.118.2.6. template<typename T> static int OpcodeBase< T >::kontrol_ (void * csound, void * opcode) [inline, static]

Definition at line 58 of file OpcodeBase.hpp.

6.118.2.7. template<typename T> void OpcodeBase< T >::log (ENVIRON * csound, const char * format, ...) [inline]

Definition at line 78 of file OpcodeBase.hpp.

References ENVIRON_::MessageV.

6.118.2.8. template<typename T> int OpcodeBase< T >::noteoff (ENVIRON * csound) [inline]

Definition at line 70 of file OpcodeBase.hpp.

References OK.

6.118.2.9. template<typename T> static int OpcodeBase< T >::noteoff_ (void * csound, void * opcode) [inline, static]

Definition at line 74 of file OpcodeBase.hpp.

Referenced by OpcodeBase< T >::init_().

6.118.2.10. template<typename T> void OpcodeBase< T >::warn (ENVIRON * csound, const char * format, ...) [inline]

Definition at line 90 of file OpcodeBase.hpp.

References ENVIRON_::MessageV, and WARNMSG.

6.118.3. Member Data Documentation

6.118.3.1. template<typename T> OPDS OpcodeBase< T >::h

Definition at line 105 of file OpcodeBase.hpp.

The documentation for this class was generated from the following file:

- H/OpcodeBase.hpp

6.119. opcodinfo Struct Reference

`#include <csoundCore.h>`

Public Attributes

- long instno
- char * name
- char * intypes
- char * outtypes
- short inchns
- short outchns
- short perf_incnt
- short perf_outcnt
- short * in_ndx_list
- short * out_ndx_list
- INSTRTXT * ip
- opcodinfo * prv

6.119.1. Member Data Documentation

6.119.1.1. short* opcodinfo::in_ndx_list

Definition at line 440 of file csoundCore.h.

6.119.1.2. short opcodinfo::inchns

Definition at line 439 of file csoundCore.h.

6.119.1.3. long opcodinfo::instno

Definition at line 437 of file csoundCore.h.

6.119.1.4. char * opcodinfo::intypes

Definition at line 438 of file csoundCore.h.

6.119.1.5. INSTRTXT* opcodinfo::ip

Definition at line 441 of file csoundCore.h.

6.119.1.6. char* opcodinfo::name

Definition at line 438 of file csoundCore.h.

6.119.1.7. short * opcodinfo::out_ndx_list

Definition at line 440 of file csoundCore.h.

6.119.1.8. short opcodinfo::outchns

Definition at line 439 of file csoundCore.h.

6.119.1.9. char * opcodinfo::outtypes

Definition at line 438 of file csoundCore.h.

6.119.1.10. short opcodinfo::perf_incnt

Definition at line 439 of file csoundCore.h.

6.119.1.11. short opcodinfo::perf_outcnt

Definition at line 439 of file csoundCore.h.

6.119.1.12. struct opcodinfo* opcodinfo::prv

Definition at line 442 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.120. opds Struct Reference

`#include <csoundCore.h>`

Public Attributes

- opds * nxti
- opds * nxtp
- SUBR iopadr
- SUBR opadr
- OPTXT * optext
- INSDS * insdshead

6.120.1. Member Data Documentation

6.120.1.1. INSDS* opds::insdshead

Definition at line 324 of file csoundCore.h.

6.120.1.2. SUBR opds::iopadr

Definition at line 321 of file csoundCore.h.

6.120.1.3. struct opds* opds::nxti

Definition at line 319 of file csoundCore.h.

6.120.1.4. struct opds* opds::nxtp

Definition at line 320 of file csoundCore.h.

6.120.1.5. SUBR opds::opadr

Definition at line 322 of file csoundCore.h.

6.120.1.6. OPTXT* opds::optext

Definition at line 323 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.121. Loris::Oscillator Class Reference

`#include <Oscillator.h>`

Public Member Functions

- Oscillator (void)
- void resetEnvelopes (const Breakpoint &bp, double srate)
- void resetPhase (double ph)
- void oscillate (double *begin, double *end, const Breakpoint &bp, double srate)
- double amplitude (void) const
- double bandwidth (void) const
- double phase (void) const
- double radianFreq (void) const

Private Attributes

- NoiseGenerator bwModulator
- double i_frequency
- double i_amplitude
- double i_bandwidth
- double determ_phase

6.121.1. Constructor & Destructor Documentation

6.121.1.1. Loris::Oscillator::Oscillator (void)

6.121.2. Member Function Documentation

6.121.2.1. double Loris::Oscillator::amplitude (void) const [inline]

Definition at line 119 of file Oscillator.h.

References i_amplitude.

6.121.2.2. double Loris::Oscillator::bandwidth (void) const [inline]

Definition at line 120 of file Oscillator.h.

References i_bandwidth.

6.121.2.3. void Loris::Oscillator::oscillate (double * *begin*, double * *end*, const Breakpoint & *bp*, double *srate*)

6.121.2.4. double Loris::Oscillator::phase (void) const [inline]

Definition at line 121 of file Oscillator.h.

References determ_phase.

6.121.2.5. double Loris::Oscillator::radianFreq (void) const [inline]

Definition at line 122 of file Oscillator.h.

References i_frequency.

6.121.2.6. void Loris::Oscillator::resetEnvelopes (const Breakpoint & *bp*, double *srate*)

6.121.2.7. void Loris::Oscillator::resetPhase (double *ph*)

6.121.3. Member Data Documentation

6.121.3.1. NoiseGenerator Loris::Oscillator::bwModulator [private]

Definition at line 62 of file Oscillator.h.

6.121.3.2. double Loris::Oscillator::determ_phase [private]

Definition at line 70 of file Oscillator.h.

Referenced by phase().

6.121.3.3. double Loris::Oscillator::i_amplitude [private]

Definition at line 66 of file Oscillator.h.

Referenced by amplitude().

6.121.3.4. double Loris::Oscillator::i_bandwidth [private]

Definition at line 67 of file Oscillator.h.

Referenced by bandwidth().

6.121.3.5. double Loris::Oscillator::i_frequency [private]

Definition at line 65 of file Oscillator.h.

Referenced by radianFreq().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Oscillator.h

6.122. Loris::Partial Class Reference

`#include <Partial.h>`

6.122.1. Detailed Description

An instance of class Partial represents a single component in the reassigned bandwidth-enhanced additive model. A Partial consists of a chain of Breakpoints describing the time-varying frequency, amplitude, and bandwidth (or noisiness) envelopes of the component, and a 4-byte label. The Breakpoints are non-uniformly distributed in time. For more information about Reassigned Bandwidth-Enhanced Analysis and the Reassigned Bandwidth-Enhanced Additive Sound Model, refer to the Loris website: www.cerlsoundgroup.org/Loris/.

The constituent time-tagged Breakpoints are accessible through Partial:iterator and Partial::const_iterator interfaces. These iterator classes implement the interface for bidirectional iterators in the STL, including pre and post-increment and decrement, and dereferencing. Dereferencing a Partial::itertator or Partial::const_itertator yields a reference to a Breakpoint. Additionally, these iterator classes have breakpoint() and time() members, returning the Breakpoint (by reference) at the current iterator position and the time (by value) corresponding to that Breakpoint.

Partial is a leaf class, do not subclass.

Most of the implementation of Partial delegates to a few container-dependent members. The following members are container-dependent, the other members are implemented in terms of these: default construction copy (construction) operator= (assign) operator== (equivalence) size insert(pos, Breakpoint) erase(b, e) findAfter(time) begin (const and non-const) end (const and non-const) first (const and non-const) last (const and non-const)

Definition at line 93 of file Partial.h.

Public Types

- typedef std::map< double, Breakpoint > container_type
- typedef int label_type
- typedef Partial_Iterator iterator
- typedef Partial_ConstIterator const_iterator
- typedef container_type::size_type size_type

Public Member Functions

- Partial (void)
- Partial (const_iterator beg, const_iterator end)
- Partial (const Partial &other)
- ~Partial (void)
- Partial & operator= (const Partial &other)
- iterator begin (void)
- const_iterator begin (void) const
- iterator end (void)
- const_iterator end (void) const
- iterator erase (iterator beg, iterator end)
- iterator findAfter (double time)
- const_iterator findAfter (double time) const
- iterator insert (double time, const Breakpoint &bp)

- size_type size (void) const
- double duration (void) const
- double endTime (void) const
- Breakpoint & first (void)
- const Breakpoint & first (void) const
- double initialPhase (void) const
- label_type label (void) const
- Breakpoint & last (void)
- const Breakpoint & last (void) const
- size_type numBreakpoints (void) const
- double startTime (void) const
- void absorb (const Partial &other)
- void setLabel (label_type l)
- iterator erase (iterator pos)
- iterator findNearest (double time)
- const_iterator findNearest (double time) const
- Partial split (iterator pos)
- double amplitudeAt (double time, double fadeTime=ShortestSafeFadeTime) const
- double bandwidthAt (double time) const
- double frequencyAt (double time) const
- double phaseAt (double time) const
- Breakpoint parametersAt (double time, double fadeTime=ShortestSafeFadeTime) const

Static Public Attributes

- static const double ShortestSafeFadeTime

Private Attributes

- label_type _label
- container_type _breakpoints

6.122.2. Member Typedef Documentation

6.122.2.1. typedef Partial_ConstIterator Loris::Partial::const_iterator

const iterator over (time, Breakpoint) pairs in this Partial

Definition at line 115 of file Partial.h.

6.122.2.2. typedef std::map< double, Breakpoint > Loris::Partial::container_type

Definition at line 102 of file Partial.h.

6.122.2.3. typedef Partial_Iterator Loris::Partial::iterator

non-const iterator over (time, Breakpoint) pairs in this Partial

Definition at line 112 of file Partial.h.

6.122.2.4. typedef int Loris::Partial::label_type

32 bit type for labeling Partials

Definition at line 109 of file Partial.h.

6.122.2.5. typedef container_type::size_type Loris::Partial::size_type

size type for number of Breakpoints in this Partial

Definition at line 118 of file Partial.h.

6.122.3. Constructor & Destructor Documentation

6.122.3.1. Loris::Partial::Partial (void)

Retun a new empty (no Breakpoints) Partial.

6.122.3.2. Loris::Partial::Partial (const_iterator *beg*, const_iterator *end*)

Retun a new Partial from a half-open (const) iterator range of time-Breakpoint pairs.

Parameters:

beg is the beginning of the range of time-Breakpoint pairs to insert into the new Partial.

end is the end of the range of time-Breakpoint pairs to insert into the new Partial.

6.122.3.3. Loris::Partial::Partial (const Partial & *other*)

Return a new Partial that is an exact copy (has an identical set of Breakpoints, at identical times, and the same label) of another Partial.

Parameters:

other is the Partial to copy.

6.122.3.4. Loris::Partial::~Partial (void)

Destroy this Partial.

6.122.4. Member Function Documentation

6.122.4.1. void Loris::Partial::absorb (const Partial & *other*)

Absorb another Partial's energy as noise (bandwidth), by accumulating the other's energy as noise energy in the portion of this Partial's envelope that overlaps (in time) with the other Partial's envelope.

Parameters:

other is the Partial to absorb.

6.122.4.2. double Loris::Partial::amplitudeAt (double *time*, double *fadeTime* = ShortestSafeFadeTime) const

Return the interpolated amplitude of this Partial at the specified time. If non-zero fadeTime is specified, then the amplitude at the ends of the Partial is computed using a linear fade. The default fadeTime is ShortestSafeFadeTime, see the definition of ShortestSafeFadeTime, above.

Parameters:

time is the time in seconds at which to evaluate the Partial.

fadeTime is the duration in seconds over which Partial amplitudes fade at the ends. The default value is ShortestSafeFadeTime, 1 ns.

Returns:

The amplitude of this Partial at the specified time.

Precondition:

The Partial must have at least one Breakpoint.

Exceptions:

InvalidPartial if the Partial has no Breakpoints.

6.122.4.3. double Loris::Partial::bandwidthAt (double *time*) const

Return the interpolated bandwidth (noisiness) coefficient of this Partial at the specified time. At times beyond the ends of the Partial, return the bandwidth coefficient at the nearest envelope endpoint.

Parameters:

time is the time in seconds at which to evaluate the Partial.

Returns:

The bandwidth of this Partial at the specified time.

Precondition:

The Partial must have at least one Breakpoint.

Exceptions:

InvalidPartial if the Partial has no Breakpoints.

6.122.4.4. const_iterator Loris::Partial::begin (void) const

Return a const iterator refering to the position of the first Breakpoint in this Partial's envelope, or end() if there are no Breakpoints in the Partial.

6.122.4.5. iterator Loris::Partial::begin (void)

Return an iterator refering to the position of the first Breakpoint in this Partial's envelope, or end() if there are no Breakpoints in the Partial.

6.122.4.6. double Loris::Partial::duration (void) const

Return the duration (in seconds) spanned by the Breakpoints in this Partial. Note that the synthesized onset time will differ, depending on the fade time used to synthesize this Partial (see class Synthesizer).

6.122.4.7. const_iterator Loris::Partial::end (void) const

Return a const iterator refering to the position past the last Breakpoint in this Partial's envelope. The iterator returned by end() (like the iterator returned by the end() member of any STL container) does not refer to a valid Breakpoint.

6.122.4.8. iterator Loris::Partial::end (void)

Return an iterator refering to the position past the last Breakpoint in this Partial's envelope. The iterator returned by end() (like the iterator returned by the end() member of any STL container) does not refer to a valid Breakpoint.

6.122.4.9. double Loris::Partial::endTime (void) const

Return the time (in seconds) of the last Breakpoint in this Partial. Note that the synthesized onset time will differ, depending on the fade time used to synthesize this Partial (see class Synthesizer).

6.122.4.10. iterator Loris::Partial::erase (iterator *pos*)

Remove the Breakpoint at the position of the given iterator, invalidating the iterator. Return a iterator referring to the next valid position, or to the end of the Partial if the last Breakpoint is removed.

Parameters:

pos is the position of the time-Breakpoint pair to be removed.

Returns:

The position (iterator) of the time-Breakpoint pair after the one that was removed.

Postcondition:

The iterator pos is invalid.

6.122.4.11. iterator Loris::Partial::erase (iterator *beg*, iterator *end*)

Breakpoint removal: erase the Breakpoints in the specified range, and return an iterator referring to the position after the, erased range.

Parameters:

beg is the beginning of the range of Breakpoints to erase

end is the end of the range of Breakpoints to erase

Returns:

The position of the first Breakpoint after the range of removed Breakpoints, or end() if the last Breakpoint in the Partial was removed.

6.122.4.12. const_iterator Loris::Partial::findAfter (double *time*) const

Return a const iterator refering to the insertion position for a Breakpoint at the specified time (that is, the position of the first Breakpoint at a time later than the specified time).

Parameters:
time is the time in seconds to find

Returns:
The last position (iterator) at which a Breakpoint at the specified time could be inserted (the position of the first Breakpoint later than time).

6.122.4.13. iterator Loris::Partial::findAfter (double *time*)

Return an iterator refering to the insertion position for a Breakpoint at the specified time (that is, the position of the first Breakpoint at a time later than the specified time).

Parameters:
time is the time in seconds to find

Returns:
The last position (iterator) at which a Breakpoint at the specified time could be inserted (the position of the first Breakpoint later than time).

6.122.4.14. const_iterator Loris::Partial::findNearest (double *time*) const

Return a const iterator refering to the position of the Breakpoint in this Partial nearest the specified time.

Parameters:
time is the time to find.

Returns:
The position (iterator) of the time-Breakpoint pair nearest (in time) to the specified time.

6.122.4.15. iterator Loris::Partial::findNearest (double *time*)

Return an iterator refering to the position of the Breakpoint in this Partial nearest the specified time.

Parameters:
time is the time to find.

Returns:
The position (iterator) of the time-Breakpoint pair nearest (in time) to the specified time.

6.122.4.16. const Breakpoint& Loris::Partial::first (void) const

Return a const reference to the first Breakpoint in the Partial's envelope.

Exceptions:
InvalidPartial if there are no Breakpoints.

6.122.4.17. Breakpoint& Loris::Partial::first (void)

Return a reference to the first Breakpoint in the Partial's envelope.

Exceptions:
InvalidPartial if there are no Breakpoints.

6.122.4.18. double Loris::Partial::frequencyAt (double *time*) const

Return the interpolated frequency (in Hz) of this Partial at the specified time. At times beyond the ends of the Partial, return the frequency at the nearest envelope endpoint.

Parameters:
time is the time in seconds at which to evaluate the Partial.

Returns:
The frequency of this Partial at the specified time.

Precondition:
The Partial must have at least one Breakpoint.

Exceptions:
InvalidPartial if the Partial has no Breakpoints.

6.122.4.19. double Loris::Partial::initialPhase (void) const

Return the phase (in radians) of this Partial at its start time (the phase of the first Breakpoint). Note that the initial synthesized phase will differ, depending on the fade time used to synthesize this Partial (see class Synthesizer).

6.122.4.20. iterator Loris::Partial::insert (double *time*, const Breakpoint & *bp*)

Breakpoint insertion: insert a copy of the specified Breakpoint in the parameter envelope at time (seconds), and return an iterator refering to the position of the inserted Breakpoint.

Parameters:
time is the time in seconds at which to insert the new Breakpoint.

bp is the new Breakpoint to insert.

Returns:
the position (iterator) of the newly-inserted time-Breakpoint pair.

6.122.4.21. label_type Loris::Partial::label (void) const

Return the 32-bit label for this Partial as an integer.

6.122.4.22. const Breakpoint& Loris::Partial::last (void) const

Return a const reference to the last Breakpoint in the Partial's envelope.

Exceptions:
InvalidPartial if there are no Breakpoints.

6.122.4.23. Breakpoint& Loris::Partial::last (void)

Return a reference to the last Breakpoint in the Partial's envelope.

Exceptions:
InvalidPartial if there are no Breakpoints.

6.122.4.24. size_type Loris::Partial::numBreakpoints (void) const

Same as size(). Return the number of Breakpoints in this Partial.

6.122.4.25. Partial& Loris::Partial::operator= (const Partial & *other*)

Make this Partial an exact copy (has an identical set of Breakpoints, at identical times, and the same label) of another Partial.

Parameters:
other is the Partial to copy.

6.122.4.26. Breakpoint Loris::Partial::parametersAt (double *time*, double *fadeTime* = ShortestSafeFadeTime) const

Return the interpolated parameters of this Partial at the specified time, same as building a Breakpoint from the results of frequencyAt, ampitudeAt, bandwidthAt, and phaseAt, but performs only one Breakpoint envelope search. If non-zero fadeTime is specified, then the amplitude at the ends of the Partial is coomputed using a linear fade. The default fadeTime is ShortestSafeFadeTime.

Parameters:
time is the time in seconds at which to evaluate the Partial.

fadeTime is the duration in seconds over which Partial amplitudes fade at the ends. The default value is ShortestSafeFadeTime, 1 ns.

Returns:
A Breakpoint describing the parameters of this Partial at the specified time.

Precondition:
The Partial must have at least one Breakpoint.

Exceptions:
InvalidPartial if the Partial has no Breakpoints.

6.122.4.27. double Loris::Partial::phaseAt (double *time*) const

Return the interpolated phase (in radians) of this Partial at the specified time. At times beyond the ends of the Partial, return the extrapolated from the nearest envelope endpoint (assuming constant frequency, as reported by frequencyAt()).

Parameters:
time is the time in seconds at which to evaluate the Partial.

Returns:
The phase of this Partial at the specified time.

Precondition:
The Partial must have at least one Breakpoint.

Exceptions:
InvalidPartial if the Partial has no Breakpoints.

6.122.4.28. void Loris::Partial::setLabel (label_type *l*)

Set the label for this Partial to the specified 32-bit value.

6.122.4.29. size_type Loris::Partial::size (void) const

Return the number of Breakpoints in this Partial.

Returns:
The number of Breakpoints in this Partial.

6.122.4.30. Partial Loris::Partial::split (iterator *pos*)

Break this Partial at the specified position (iterator). The Breakpoint at the specified position becomes the first Breakpoint in a new Partial. Breakpoints at the specified position and subsequent positions are removed from this Partial and added to the new Partial, which is returned.

Parameters:
pos is the position at which to split this Partial.

Returns:
A new Partial consisting of time-Breakpoint pairs beginning with pos and extending to the end of this Partial.

Postcondition:
All positions beginning with pos and extending to the end of this Partial have been removed.

6.122.4.31. double Loris::Partial::startTime (void) const

Return the time (in seconds) of the first Breakpoint in this Partial. Note that the synthesized onset time will differ, depending on the fade time used to synthesize this Partial (see class Synthesizer).

6.122.5. Member Data Documentation

6.122.5.1. container_type Loris::Partial::_breakpoints [private]

Definition at line 420 of file Partial.h.

6.122.5.2. label_type Loris::Partial::_label [private]

Definition at line 419 of file Partial.h.

6.122.5.3. const double Loris::Partial::ShortestSafeFadeTime [static]

Define the default fade time for computing amplitude at the ends of a Partial. Floating point round-off errors make fadeTime == 0.0 dangerous and unpredictable. 1 ns is short enough to prevent rounding errors in the least significant bit of a 48-bit mantissa for times up to ten hours.

1 nanosecond, see Partial.C

Definition at line 344 of file Partial.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Partial.h

6.123. Loris::Partial_ConstIterator Class Reference

`#include <Partial.h>`

6.123.1. Detailed Description

Const iterator for the Loris::Partial Breakpoint map. Wraps the non-const iterator for the (time,Breakpoint) pair container Partial::container_type. Partial_Iterator implements a bidirectional iterator interface, and additionally offers time and Breakpoint (reference) access through time() and breakpoint() members.

Definition at line 595 of file Partial.h.

Public Types

- typedef BaseIterator::iterator_category iterator_category
- typedef Breakpoint value_type
- typedef BaseIterator::difference_type difference_type
- typedef const Breakpoint * pointer
- typedef const Breakpoint & reference

Public Member Functions

- Partial_ConstIterator (void)
- Partial_ConstIterator (const Partial_Iterator &other)
- Partial_ConstIterator & operator++ ()
- Partial_ConstIterator & operator– ()
- Partial_ConstIterator operator++ (int)
- Partial_ConstIterator operator– (int)
- const Breakpoint & operator * (void) const
- const Breakpoint * operator → (void) const
- const Breakpoint & breakpoint (void) const
- double time (void) const

Private Types

- typedef Partial::container_type BaseContainer
- typedef BaseContainer::const_iterator BaseIterator

Private Member Functions

- Partial_ConstIterator (BaseIterator it)

Private Attributes

- BaseIterator _iter

Friends

- class Partial
- bool operator== (const Partial_ConstIterator &lhs, const Partial_ConstIterator &rhs)
- bool operator!= (const Partial_ConstIterator &lhs, const Partial_ConstIterator &rhs)

6.123.2. Member Typedef Documentation

6.123.2.1. typedef Partial::container_type Loris::Partial_ConstIterator::BaseContainer [private]

Definition at line 598 of file Partial.h.

6.123.2.2. typedef BaseContainer::const_iterator Loris::Partial_ConstIterator::BaseIterator [private]

Definition at line 599 of file Partial.h.

6.123.2.3. typedef BaseIterator::difference_type Loris::Partial_ConstIterator::difference_type

Definition at line 607 of file Partial.h.

6.123.2.4. typedef BaseIterator::iterator_category Loris::Partial_ConstIterator::iterator_category

Definition at line 605 of file Partial.h.

6.123.2.5. typedef const Breakpoint* Loris::Partial_ConstIterator::pointer

Definition at line 608 of file Partial.h.

6.123.2.6. typedef const Breakpoint& Loris::Partial_ConstIterator::reference

Definition at line 609 of file Partial.h.

6.123.2.7. typedef Breakpoint Loris::Partial_ConstIterator::value_type

Definition at line 606 of file Partial.h.

6.123.3. Constructor & Destructor Documentation

6.123.3.1. Loris::Partial_ConstIterator::Partial_ConstIterator (void) [inline]

Construct a new iterator referring to no position in any Partial.

Definition at line 615 of file Partial.h.

Referenced by operator++(), and operator–().

6.123.3.2. Loris::Partial_ConstIterator::Partial_ConstIterator (const Partial_Iterator & *other*) [inline]

Construct a new const iterator from a non-const iterator.

Parameters:
other a non-const iterator from which to make a read-only copy.

Definition at line 621 of file Partial.h.

References _iter.

6.123.3.3. Loris::Partial_ConstIterator::Partial_ConstIterator (BaseIterator *it*) [inline, private]

Definition at line 722 of file Partial.h.

References _iter.

6.123.4. Member Function Documentation

6.123.4.1. const Breakpoint& Loris::Partial_ConstIterator::breakpoint (void) const [inline]

Breakpoint accessor.

Returns:
A const reference to the Breakpoint at the position of this iterator.

Definition at line 709 of file Partial.h.

References _iter.

Referenced by operator *(), and operator →().

6.123.4.2. const Breakpoint& Loris::Partial_ConstIterator::operator * (void) const [inline]

Dereference operator.

Returns:
A const reference to the Breakpoint at the position of this iterator.

Definition at line 673 of file Partial.h.

References breakpoint().

6.123.4.3. Partial_ConstIterator Loris::Partial_ConstIterator::operator++ (int) [inline]

Post-increment operator - advance the position of the iterator and return a copy of the iterator before it was advanced. The int argument is unused compiler magic.

Returns:
An iterator that is a copy of this iterator before being advanced.

Precondition:
The iterator must be a valid position before the end in some Partial.

Definition at line 654 of file Partial.h.

References _iter, and Partial_ConstIterator().

6.123.4.4. Partial_ConstIterator& Loris::Partial_ConstIterator::operator++ () [inline]

Pre-increment operator - advance the position of the iterator and return the iterator itself.

Returns:
This iterator (reference to self).

Precondition:
The iterator must be a valid position before the end in some Partial.

Definition at line 634 of file Partial.h.

References _iter.

6.123.4.5. Partial_ConstIterator Loris::Partial_ConstIterator::operator– (int) [inline]

Post-decrement operator - move the position of the iterator back by one and return a copy of the iterator before it was decremented. The int argument is unused compiler magic.

Returns:
An iterator that is a copy of this iterator before being decremented.

Precondition:
The iterator must be a valid position after the beginning in some Partial.

Definition at line 664 of file Partial.h.

References _iter, and Partial_ConstIterator().

6.123.4.6. Partial_ConstIterator& Loris::Partial_ConstIterator::operator– () [inline]

Pre-decrement operator - move the position of the iterator back by one and return the iterator itself.

Returns:
This iterator (reference to self).

Precondition:
The iterator must be a valid position after the beginning in some Partial.

Definition at line 642 of file Partial.h.

References _iter.

6.123.4.7. const Breakpoint* Loris::Partial_ConstIterator::operator → (void) const [inline]

Pointer operator.

Returns:
A const pointer to the Breakpoint at the position of this iterator.

Definition at line 679 of file Partial.h.

References breakpoint().

6.123.4.8. double Loris::Partial_ConstIterator::time (void) const [inline]

Time accessor.

Returns:
The time in seconds of the Breakpoint at the position of this iterator.

Definition at line 716 of file Partial.h.

References _iter.

6.123.5. Friends And Related Function Documentation

6.123.5.1. bool operator!= (const Partial_ConstIterator & *lhs*, const Partial_ConstIterator & *rhs*) [friend]

Inequality comparison operator.

Parameters:
lhs the iterator on the left side of the operator.

rhs the iterator on the right side of the operator.

Returns:
false if the two iterators refer to the same position in the same Partial, true otherwise.

Definition at line 699 of file Partial.h.

6.123.5.2. bool operator== (const Partial_ConstIterator & *lhs*, const Partial_ConstIterator & *rhs*) [friend]

Equality comparison operator.

Parameters:
lhs the iterator on the left side of the operator.

rhs the iterator on the right side of the operator.

Returns:
true if the two iterators refer to the same position in the same Partial, false otherwise.

Definition at line 689 of file Partial.h.

6.123.5.3. friend class Partial [friend]

Definition at line 725 of file Partial.h.

6.123.6. Member Data Documentation

6.123.6.1. BaseIterator Loris::Partial_ConstIterator::_iter [private]

Definition at line 600 of file Partial.h.

Referenced by breakpoint(), operator++(), operator–(), Partial_ConstIterator(), and time().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Partial.h

6.124. Loris::Partial_Iterator Class Reference

`#include <Partial.h>`

6.124.1. Detailed Description

Non-const iterator for the Loris::Partial Breakpoint map. Wraps the non-const iterator for the (time,Breakpoint) pair container Partial::container_type. Partial_Iterator implements a bidirectional iterator interface, and additionally offers time and Breakpoint (reference) access through time() and breakpoint() members.

Definition at line 434 of file Partial.h.

Public Types

- typedef BaseIterator::iterator_category iterator_category
- typedef Breakpoint value_type
- typedef BaseIterator::difference_type difference_type
- typedef Breakpoint * pointer
- typedef Breakpoint & reference

Public Member Functions

- Partial_Iterator (void)
- Partial_Iterator & operator++ ()
- Partial_Iterator & operator– ()
- Partial_Iterator operator++ (int)
- Partial_Iterator operator– (int)
- Breakpoint & operator * (void) const
- Breakpoint * operator → (void) const
- Breakpoint & breakpoint (void) const
- double time (void) const

Private Types

- typedef Partial::container_type BaseContainer
- typedef BaseContainer::iterator BaseIterator

Private Member Functions

- Partial_Iterator (const BaseIterator &it)

Private Attributes

- BaseIterator _iter

Friends

- class Partial
- class Partial_ConstIterator
- bool operator== (const Partial_Iterator &lhs, const Partial_Iterator &rhs)
- bool operator!= (const Partial_Iterator &lhs, const Partial_Iterator &rhs)

6.124.2. Member Typedef Documentation

6.124.2.1. typedef Partial::container_type Loris::Partial_Iterator::BaseContainer [private]

Definition at line 437 of file Partial.h.

6.124.2.2. typedef BaseContainer::iterator Loris::Partial_Iterator::BaseIterator [private]

Definition at line 438 of file Partial.h.

6.124.2.3. typedef BaseIterator::difference_type Loris::Partial_Iterator::difference_type

Definition at line 446 of file Partial.h.

6.124.2.4. typedef BaseIterator::iterator_category Loris::Partial_Iterator::iterator_category

Definition at line 444 of file Partial.h.

6.124.2.5. typedef Breakpoint* Loris::Partial_Iterator::pointer

Definition at line 447 of file Partial.h.

6.124.2.6. typedef Breakpoint& Loris::Partial_Iterator::reference

Definition at line 448 of file Partial.h.

6.124.2.7. typedef Breakpoint Loris::Partial_Iterator::value_type

Definition at line 445 of file Partial.h.

6.124.3. Constructor & Destructor Documentation

6.124.3.1. Loris::Partial_Iterator::Partial_Iterator (void) [inline]

Construct a new iterator referring to no position in any Partial.

Definition at line 454 of file Partial.h.

Referenced by operator++(), and operator–().

6.124.3.2. Loris::Partial_Iterator::Partial_Iterator (const BaseIterator & *it*) [inline, private]

Definition at line 574 of file Partial.h.

References _iter.

6.124.4. Member Function Documentation

6.124.4.1. Breakpoint& Loris::Partial_Iterator::breakpoint (void) const [inline]

Breakpoint accessor.

Returns:
A const reference to the Breakpoint at the position of this iterator.

Definition at line 554 of file Partial.h.

References _iter.

Referenced by operator *(), and operator →().

6.124.4.2. Breakpoint& Loris::Partial_Iterator::operator * (void) const [inline]

Dereference operator.

Returns:
A reference to the Breakpoint at the position of this iterator.

Definition at line 505 of file Partial.h.

References breakpoint().

6.124.4.3. Partial_Iterator Loris::Partial_Iterator::operator++ (int) [inline]

Post-increment operator - advance the position of the iterator and return a copy of the iterator before it was advanced. The int argument is unused compiler magic.

Returns:
An iterator that is a copy of this iterator before being advanced.

Precondition:
The iterator must be a valid position before the end in some Partial.

Definition at line 486 of file Partial.h.

References _iter, and Partial_Iterator().

6.124.4.4. Partial_Iterator& Loris::Partial_Iterator::operator++ () [inline]

Pre-increment operator - advance the position of the iterator and return the iterator itself.

Returns:
This iterator (reference to self).

Precondition:
The iterator must be a valid position before the end in some Partial.

Definition at line 466 of file Partial.h.

References _iter.

6.124.4.5. Partial_Iterator Loris::Partial_Iterator::operator– (int) [inline]

Post-decrement operator - move the position of the iterator back by one and return a copy of the iterator before it was decremented. The int argument is unused compiler magic.

Returns:
An iterator that is a copy of this iterator before being decremented.

Precondition:
The iterator must be a valid position after the beginning in some Partial.

Definition at line 496 of file Partial.h.

References _iter, and Partial_Iterator().

6.124.4.6. Partial_Iterator& Loris::Partial_Iterator::operator– () [inline]

Pre-decrement operator - move the position of the iterator back by one and return the iterator itself.

Returns:
This iterator (reference to self).

Precondition:
The iterator must be a valid position after the beginning in some Partial.

Definition at line 474 of file Partial.h.

References _iter.

6.124.4.7. Breakpoint* Loris::Partial_Iterator::operator → (void) const [inline]

Pointer operator.

Returns:
A pointer to the Breakpoint at the position of this iterator.

Definition at line 518 of file Partial.h.

References breakpoint().

6.124.4.8. double Loris::Partial_Iterator::time (void) const [inline]

Time accessor.

Returns:
The time in seconds of the Breakpoint at the position of this iterator.

Definition at line 568 of file Partial.h.

References _iter.

6.124.5. Friends And Related Function Documentation

6.124.5.1. bool operator!= (const Partial_Iterator & *lhs*, const Partial_Iterator & *rhs*) [friend]

Inequality comparison operator.

Parameters:
lhs the iterator on the left side of the operator.

rhs the iterator on the right side of the operator.

Returns:
false if the two iterators refer to the same position in the same Partial, true otherwise.

Definition at line 544 of file Partial.h.

6.124.5.2. bool operator== (const Partial_Iterator & *lhs*, const Partial_Iterator & *rhs*) [friend]

Equality comparison operator.

Parameters:
lhs the iterator on the left side of the operator.

rhs the iterator on the right side of the operator.

Returns:
true if the two iterators refer to the same position in the same Partial, false otherwise.

Definition at line 534 of file Partial.h.

6.124.5.3. friend class Partial [friend]

Definition at line 577 of file Partial.h.

6.124.5.4. friend class Partial_ConstIterator [friend]

Definition at line 581 of file Partial.h.

6.124.6. Member Data Documentation

6.124.6.1. BaseIterator Loris::Partial_Iterator::_iter [private]

Definition at line 439 of file Partial.h.

Referenced by breakpoint(), operator++(), operator–(), Partial_Iterator(), and time().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Partial.h

6.125. Loris::PartialBuilder Class Reference

`#include <PartialBuilder.h>`

Public Member Functions

- PartialBuilder (double drift)
- PartialBuilder (double drift, const Envelope &env)
- void formPartials (Peaks &peaks, double frameTime)
- PartialList & fixPartialFrequencies (void)
- PartialList & partials (void)

Private Attributes

- PartialList partials_
- PartialPtrs eligiblePartials
- PartialPtrs newlyEligible
- const Envelope & reference
- double freqDrift

6.125.1. Constructor & Destructor Documentation

6.125.1.1. Loris::PartialBuilder::PartialBuilder (double *drift*) [explicit]

6.125.1.2. Loris::PartialBuilder::PartialBuilder (double *drift*, const Envelope & *env*)

6.125.2. Member Function Documentation

6.125.2.1. PartialList& Loris::PartialBuilder::fixPartialFrequencies (void)

6.125.2.2. void Loris::PartialBuilder::formPartials (Peaks & *peaks*, double *frameTime*)

6.125.2.3. PartialList& Loris::PartialBuilder::partials (void) [inline]

Definition at line 77 of file PartialBuilder.h.

References partials_.

6.125.3. Member Data Documentation

6.125.3.1. PartialPtrs Loris::PartialBuilder::eligiblePartials [private]

Definition at line 83 of file PartialBuilder.h.

6.125.3.2. double Loris::PartialBuilder::freqDrift [private]

Definition at line 85 of file PartialBuilder.h.

6.125.3.3. PartialPtrs Loris::PartialBuilder::newlyEligible [private]

Definition at line 83 of file PartialBuilder.h.

6.125.3.4. PartialList Loris::PartialBuilder::partials_ [private]

Definition at line 81 of file PartialBuilder.h.

Referenced by partials().

6.125.3.5. const Envelope& Loris::PartialBuilder::reference [private]

Definition at line 84 of file PartialBuilder.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialBuilder.h

6.126. Loris::PartialUtils::PartialMutator Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::PartialMutator::

6.126.1. Detailed Description

PartialMutator is an abstract base class for Partial mutators, functors that operate on Partials according to a time-varying envelope. The base class manages a polymorphic Envelope instance that provides the time-varying mutation parameters.

Invariant:
env is a non-zero pointer to a valid instance of a class derived from the abstract class Envelope.

Definition at line 66 of file PartialUtils.h.

Public Member Functions

- PartialMutator (double x)
- PartialMutator (const Envelope &e)
- PartialMutator (const PartialMutator &rhs)
- virtual ~PartialMutator (void)
- PartialMutator & operator= (const PartialMutator &rhs)
- virtual void operator() (Partial &p) const =0

Protected Attributes

- Envelope * env

6.126.2. Constructor & Destructor Documentation

6.126.2.1. Loris::PartialUtils::PartialMutator::PartialMutator (double *x*)

Construct a new PartialMutator from a constant mutation factor.

6.126.2.2. Loris::PartialUtils::PartialMutator::PartialMutator (const Envelope & *e*)

Construct a new PartialMutator from an Envelope representing a time-varying mutation factor.

6.126.2.3. Loris::PartialUtils::PartialMutator::PartialMutator (const PartialMutator & *rhs*)

Construct a new PartialMutator that is a copy of another.

6.126.2.4. virtual Loris::PartialUtils::PartialMutator::~PartialMutator (void) [virtual]

Destroy this PartialMutator, deleting its Envelope.

6.126.3. Member Function Documentation

6.126.3.1. virtual void Loris::PartialUtils::PartialMutator::operator() (Partial & *p*) const [pure virtual]

Function call operator: apply a mutation factor to the specified Partial. Derived classes must implement this member.

Implemented in Loris::PartialUtils::AmplitudeScaler, Loris::PartialUtils::BandwidthScaler, Loris::PartialUtils::FrequencyScaler, Loris::PartialUtils::NoiseRatioScaler, and Loris::PartialUtils::PitchShifter.

6.126.3.2. PartialMutator& Loris::PartialUtils::PartialMutator::operator= (const PartialMutator & *rhs*)

Make this PartialMutator a duplicate of another one.

Parameters:

rhs is the PartialMutator to copy.

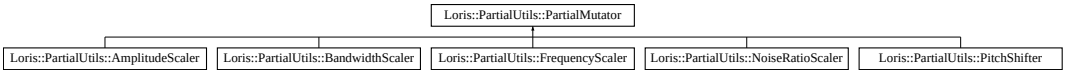
6.126.4. Member Data Documentation

6.126.4.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::AmplitudeScaler
Loris::PartialUtils::BandwidthScaler
Loris::PartialUtils::FrequencyScaler
Loris::PartialUtils::NoiseRatioScaler
Loris::PartialUtils::PitchShifter

6.127. Loris::PartialUtils::PitchShifter Class Reference

`#include <PartialUtils.h>`

Inheritance diagram for Loris::PartialUtils::PitchShifter::

Public Member Functions

- PitchShifter (double x)
- PitchShifter (const Envelope &e)
- void operator() (Partial &p) const

Protected Attributes

- Envelope * env

6.127.1. Constructor & Destructor Documentation

6.127.1.1. Loris::PartialUtils::PitchShifter::PitchShifter (double *x*) [inline]

Definition at line 291 of file PartialUtils.h.

6.127.1.2. Loris::PartialUtils::PitchShifter::PitchShifter (const Envelope & *e*) [inline]

Definition at line 292 of file PartialUtils.h.

6.127.2. Member Function Documentation

6.127.2.1. void Loris::PartialUtils::PitchShifter::operator() (Partial & *p*) const [virtual]

Function call operator: apply a mutation factor to the specified Partial. Derived classes must implement this member.

Implements Loris::PartialUtils::PartialMutator.

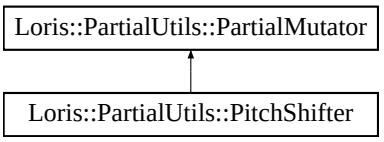
6.127.3. Member Data Documentation

6.127.3.1. Envelope* Loris::PartialUtils::PartialMutator::env [protected, inherited]

Definition at line 94 of file PartialUtils.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h


Loris::PartialUtils::PartialMutator
Loris::PartialUtils::PitchShifter

6.128. Plots Class Reference

`#include <plots.hpp>`

Public Member Functions

- Plots (int x, int y, int w, int h, const char *label=0)
- void append (Curve *)
- void clear ()

Private Member Functions

- void choice (Fl_Widget *)

Static Private Member Functions

- static void choice_callback (Fl_Widget *, void *)

Private Attributes

- Fl_Menu_Item m_menu_items [menu_size+1]
- Curve * m_curves [menu_size]
- Fl_Choice * m_choice
- Canvas * m_canvas
- int m_selection

Static Private Attributes

- static const int menu_size = 50

6.128.1. Constructor & Destructor Documentation

6.128.1.1. Plots::Plots (int *x*, int *y*, int *w*, int *h*, const char * *label* = 0)

6.128.2. Member Function Documentation

6.128.2.1. void Plots::append (Curve *)

6.128.2.2. void Plots::choice (Fl_Widget *) [private]

6.128.2.3. static void Plots::choice_callback (Fl_Widget *, void *) [static, private]

6.128.2.4. void Plots::clear ()

6.128.3. Member Data Documentation

6.128.3.1. Canvas* Plots::m_canvas [private]

Definition at line 35 of file plots.hpp.

6.128.3.2. Fl_Choice* Plots::m_choice [private]

Definition at line 34 of file plots.hpp.

6.128.3.3. Curve* Plots::m_curves[menu_size] [private]

Definition at line 33 of file plots.hpp.

6.128.3.4. Fl_Menu_Item Plots::m_menu_items[menu_size+1] [private]

Definition at line 32 of file plots.hpp.

6.128.3.5. int Plots::m_selection [private]

Definition at line 36 of file plots.hpp.

6.128.3.6. const int Plots::menu_size = 50 [static, private]

Definition at line 31 of file plots.hpp.

The documentation for this class was generated from the following file:

- frontends/flcsound/plots.hpp

6.129. polish Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char opcod [12]
- int incount
- char * arg [4]

6.129.1. Member Data Documentation

6.129.1.1. char* polish::arg[4]

Definition at line 151 of file csoundCore.h.

6.129.1.2. int polish::incount

Definition at line 150 of file csoundCore.h.

6.129.1.3. char polish::opcod[12]

Definition at line 149 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.130. Preset Class Reference

`#include <CsoundVST.hpp>`

6.130.1. Detailed Description

C S O U N D V S T

A VST plugin version of Csound, with Python scripting.

L I C E N S E

This software is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Definition at line 50 of file CsoundVST.hpp.

Public Attributes

- std::string name
- std::string text

6.130.2. Member Data Documentation

6.130.2.1. std::string Preset::name

Definition at line 53 of file CsoundVST.hpp.

6.130.2.2. std::string Preset::text

Definition at line 54 of file CsoundVST.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CsoundVST.hpp

6.131. pvx_memfile_ Struct Reference

#include <csoundCore.h>

Public Attributes

- char * filename
- pvx_memfile_ * nxt
- float * data
- unsigned long nframes
- int format
- int fftsize
- int overlap
- int winsize
- int wintype
- int chans
- MYFLT srate

6.131.1. Member Data Documentation

6.131.1.1. int pvx_memfile_::chans

Definition at line 536 of file csoundCore.h.

6.131.1.2. float* pvx_memfile_::data

Definition at line 529 of file csoundCore.h.

6.131.1.3. int pvx_memfile_::fftsize

Definition at line 532 of file csoundCore.h.

6.131.1.4. char* pvx_memfile_::filename

Definition at line 527 of file csoundCore.h.

6.131.1.5. int pvx_memfile_::format

Definition at line 531 of file csoundCore.h.

6.131.1.6. unsigned long pvx_memfile_::nframes

Definition at line 530 of file csoundCore.h.

6.131.1.7. struct pvx_memfile_ * pvx_memfile_::nxt

Definition at line 528 of file csoundCore.h.

6.131.1.8. int pvx_memfile_::overlap

Definition at line 533 of file csoundCore.h.

6.131.1.9. MYFLT pvx_memfile_::srate

Definition at line 537 of file csoundCore.h.

6.131.1.10. int pvx_memfile_::winsize

Definition at line 534 of file csoundCore.h.

6.131.1.11. int pvx_memfile_::wintype

Definition at line 535 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.132. csound::Random Class Reference

`#include <Random.hpp>`

Inheritance diagram for csound::Random::

6.132.1. Detailed Description

A random value will be sampled from the specified distribution, translated and scaled as specified, and set in the specified row and column of the local coordinates. The resulting matrix will be used in place of the local coordinates when traversing the music graph. If eventCount is greater than zero, a new event will be created for each of eventCount samples, which will be transformed by the newly sampled local coordinates.

Definition at line 57 of file Random.hpp.

Public Member Functions

- Random ()
- virtual ~Random ()
- virtual double sample () const
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual void createDistribution (std::string distribution)
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &globalCoordinates)
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Static Public Member Functions

- static void seed (int s)

Public Attributes

- std::string distribution
- int row
- int column
- int eventCount
- bool incrementTime
- double minimum
- double maximum
- double q
- double a
- double b
- double c
- double Lambda
- double mean
- double sigma
- std::vector< Node * > children

Static Public Attributes

- static boost::mt19937 mersenneTwister

Protected Attributes

- void * generator_
- boost::variate_generator< boost::mt19937, boost::uniform_smallint<> > * uniform_smallint_generator
- boost::variate_generator< boost::mt19937, boost::uniform_int<> > * uniform_int_generator
- boost::variate_generator< boost::mt19937, boost::uniform_real<> > * uniform_real_generator
- boost::variate_generator< boost::mt19937, boost::bernoulli_distribution<> > * bernoulli_distribution_generator
- boost::variate_generator< boost::mt19937, boost::geometric_distribution<> > * geometric_distribution_generator
- boost::variate_generator< boost::mt19937, boost::triangle_distribution<> > * triangle_distribution_generator
- boost::variate_generator< boost::mt19937, boost::exponential_distribution<> > * exponential_distribution_generator
- boost::variate_generator< boost::mt19937, boost::normal_distribution<> > * normal_distribution_generator
- boost::variate_generator< boost::mt19937, boost::lognormal_distribution<> > * lognormal_distribution_generator
- ublas::matrix< double > localCoordinates

6.132.2. Constructor & Destructor Documentation

6.132.2.1. csound::Random::Random ()

6.132.2.2. virtual csound::Random::~Random () [virtual]

6.132.3. Member Function Documentation

6.132.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.132.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.132.3.3. virtual void csound::Random::createDistribution (std::string *distribution*) [virtual]

6.132.3.4. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.132.3.5. virtual ublas::matrix<double> csound::Random::getLocalCoordinates () const [virtual]

Returns the local transformation of coordinate system.

Reimplemented from csound::Node.

6.132.3.6. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.132.3.7. virtual void csound::Random::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *globalCoordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::Node.

6.132.3.8. virtual double csound::Random::sample () const [virtual]

6.132.3.9. static void csound::Random::seed (int *s*) [static]

6.132.3.10. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.132.3.11. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.132.4. Member Data Documentation

6.132.4.1. double csound::Random::a

Definition at line 81 of file Random.hpp.

6.132.4.2. double csound::Random::b

Definition at line 82 of file Random.hpp.

6.132.4.3. boost::variate_generator<boost::mt19937, boost::bernoulli_distribution<> >* csound::Random::bernoulli_distribution_generator [protected]

Definition at line 65 of file Random.hpp.

6.132.4.4. double csound::Random::c

Definition at line 83 of file Random.hpp.

6.132.4.5. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.132.4.6. int csound::Random::column

Definition at line 75 of file Random.hpp.

6.132.4.7. std::string csound::Random::distribution

Definition at line 73 of file Random.hpp.

6.132.4.8. int csound::Random::eventCount

Definition at line 76 of file Random.hpp.

6.132.4.9. boost::variate_generator<boost::mt19937, boost::exponential_distribution<> >* csound::Random::exponential_distribution_generator [protected]

Definition at line 68 of file Random.hpp.

6.132.4.10. void* csound::Random::generator_ [protected]

Definition at line 61 of file Random.hpp.

6.132.4.11. boost::variate_generator<boost::mt19937, boost::geometric_distribution<> >* csound::Random::geometric_distribution_generator [protected]

Definition at line 66 of file Random.hpp.

6.132.4.12. bool csound::Random::incrementTime

Definition at line 77 of file Random.hpp.

6.132.4.13. double csound::Random::Lambda

Definition at line 84 of file Random.hpp.

6.132.4.14. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.132.4.15. boost::variate_generator<boost::mt19937, boost::lognormal_distribution<> >* csound::Random::lognormal_distribution_generator [protected]

Definition at line 70 of file Random.hpp.

6.132.4.16. double csound::Random::maximum

Definition at line 79 of file Random.hpp.

6.132.4.17. double csound::Random::mean

Definition at line 85 of file Random.hpp.

6.132.4.18. boost::mt19937 csound::Random::mersenneTwister [static]

Definition at line 72 of file Random.hpp.

6.132.4.19. double csound::Random::minimum

Definition at line 78 of file Random.hpp.

6.132.4.20. boost::variate_generator<boost::mt19937, boost::normal_distribution<> >* csound::Random::normal_distribution_generator [protected]

Definition at line 69 of file Random.hpp.

6.132.4.21. double csound::Random::q

Definition at line 80 of file Random.hpp.

6.132.4.22. int csound::Random::row

Definition at line 74 of file Random.hpp.

6.132.4.23. double csound::Random::sigma

Definition at line 86 of file Random.hpp.

6.132.4.24. boost::variate_generator<boost::mt19937, boost::triangle_distribution<> >* csound::Random::triangle_distribution_generator [protected]

Definition at line 67 of file Random.hpp.

6.132.4.25. boost::variate_generator<boost::mt19937, boost::uniform_int<> >* csound::Random::uniform_int_generator [protected]

Definition at line 63 of file Random.hpp.

6.132.4.26. boost::variate_generator<boost::mt19937, boost::uniform_real<> >* csound::Random::uniform_real_generator [protected]

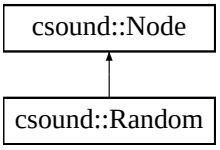
Definition at line 64 of file Random.hpp.

6.132.4.27. boost::variate_generator<boost::mt19937, boost::uniform_smallint<> >* csound::Random::uniform_smallint_generator [protected]

Definition at line 62 of file Random.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Random.hpp


csound::Node
csound::Random

6.133. Loris::ReassignedSpectrum Class Reference

`#include <ReassignedSpectrum.h>`

Public Member Functions

- ReassignedSpectrum (const std::vector< double > &window)
- ~ReassignedSpectrum (void)
- void transform (const double *sampsBegin, const double *pos, const double *sampsEnd)
- long size (void) const
- const std::vector< double > & window (void) const
- double reassignedFrequency (unsigned long idx) const
- double reassignedTime (unsigned long idx) const
- double reassignedPhase (long idx, double fracFreqSample, double timeCorrection) const
- double reassignedMagnitude (double fracBinNum, long intBinNumber) const
- const std::complex< double > & operator[] (unsigned long idx) const
- double frequencyCorrection (long sample) const
- double timeCorrection (long sample) const

Private Attributes

- FourierTransform _transform
- FourierTransform _ratransform
- std::vector< double > _window
- std::vector< std::complex< double > > _rawindow
- double _windowMagnitudeScale
- double _phaseSlope

6.133.1. Constructor & Destructor Documentation

6.133.1.1. Loris::ReassignedSpectrum::ReassignedSpectrum (const std::vector< double > & *window*)

6.133.1.2. Loris::ReassignedSpectrum::~ReassignedSpectrum (void)

6.133.2. Member Function Documentation

6.133.2.1. double Loris::ReassignedSpectrum::frequencyCorrection (long *sample*) const

6.133.2.2.]

const std::complex< double >& Loris::ReassignedSpectrum::operator[] (unsigned long *idx*) const `[inline]`

Definition at line 89 of file ReassignedSpectrum.h.

References _transform.

6.133.2.3. double Loris::ReassignedSpectrum::reassignedFrequency (unsigned long *idx*) const

6.133.2.4. double Loris::ReassignedSpectrum::reassignedMagnitude (double *fracBinNum*, long *intBinNumber*) const

6.133.2.5. double Loris::ReassignedSpectrum::reassignedPhase (long *idx*, double *fracFreqSample*, double *timeCorrection*) const

6.133.2.6. double Loris::ReassignedSpectrum::reassignedTime (unsigned long *idx*) const

6.133.2.7. long Loris::ReassignedSpectrum::size (void) const [inline]

Definition at line 77 of file ReassignedSpectrum.h.

References _transform, and Loris::FourierTransform::size().

6.133.2.8. double Loris::ReassignedSpectrum::timeCorrection (long *sample*) const

6.133.2.9. void Loris::ReassignedSpectrum::transform (const double * *sampsBegin*, const double * *pos*, const double * *sampsEnd*)

6.133.2.10. const std::vector< double >& Loris::ReassignedSpectrum::window (void) const [inline]

Definition at line 81 of file ReassignedSpectrum.h.

References _window.

6.133.3. Member Data Documentation

6.133.3.1. double Loris::ReassignedSpectrum::_phaseSlope [private]

Definition at line 65 of file ReassignedSpectrum.h.

6.133.3.2. FourierTransform Loris::ReassignedSpectrum::_ratransform [private]

Definition at line 53 of file ReassignedSpectrum.h.

6.133.3.3. std::vector< std::complex< double > > Loris::ReassignedSpectrum::_rawindow [private]

Definition at line 57 of file ReassignedSpectrum.h.

6.133.3.4. FourierTransform Loris::ReassignedSpectrum::_transform [private]

Definition at line 53 of file ReassignedSpectrum.h.

Referenced by operator[](), and size().

6.133.3.5. std::vector< double > Loris::ReassignedSpectrum::_window [private]

Definition at line 56 of file ReassignedSpectrum.h.

Referenced by window().

6.133.3.6. double Loris::ReassignedSpectrum::_windowMagnitudeScale [private]

Definition at line 60 of file ReassignedSpectrum.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/ReassignedSpectrum.h

6.134. Loris::Resampler Class Reference

`#include <Resampler.h>`

6.134.1. Detailed Description

Class Resampler represents an algorithm for resampling Partial envelopes at regular time intervals. Resampling makes the envelope data more suitable for exchange (as SDIF data, for example) with other applications that cannot process raw (continuously-distributed) reassigned data. Resampling will often greatly reduce the size of the data (by greatly reducing the number of Breakpoints in the Partials) without adversely affecting the quality of the reconstruction.

Definition at line 58 of file Resampler.h.

Public Member Functions

- Resampler (double sampleInterval)
- void resample (Partial &p) const
- void operator() (Partial &p) const
- template<typename Iter> void resample (Iter begin, Iter end) const
- template<typename Iter> void operator() (Iter begin, Iter end) const

Static Public Member Functions

- template<typename Iter> static void resample (Iter begin, Iter end, double sampleInterval)

Private Attributes

- double interval_

6.134.2. Constructor & Destructor Documentation

6.134.2.1. Loris::Resampler::Resampler (double *sampleInterval*) [explicit]

Construct a new Resampler using the specified sampling interval.

Parameters:

sampleInterval is the resampling interval in seconds, Breakpoint data is computed at integer multiples of sampleInterval seconds.

Exceptions:

InvalidArgument if sampleInterval is not positive.

6.134.3. Member Function Documentation

6.134.3.1. template<typename Iter> void Loris::Resampler::operator() (Iter *begin*, Iter *end*) const [inline]

Function call operator: same as resample(begin, end).

Definition at line 120 of file Resampler.h.

References resample().

6.134.3.2. void Loris::Resampler::operator() (Partial & *p*) const [inline]

Function call operator: same as resample(p).

Definition at line 89 of file Resampler.h.

References resample().

6.134.3.3. template<typename Iter> void Loris::Resampler::resample (Iter *begin*, Iter *end*, double *sampleInterval*) [static]

Static member that constructs an instance and applies it to a sequence of Partials. Construct a Resampler using the specified resampling interval, and use it to channelize a sequence of Partials.

Parameters:

begin is the beginning of a sequence of Partials to resample.

end is the end of a sequence of Partials to resample.

sampleInterval is the resampling interval in seconds, Breakpoint data is computed at integer multiples of sampleInterval seconds.

Exceptions:

InvalidArgument if sampleInterval is not positive.

If compiled with NO_TEMPLATE_MEMBERS defined, then begin and end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 221 of file Resampler.h.

References resample().

6.134.3.4. template<typename Iter> void Loris::Resampler::resample (Iter *begin*, Iter *end*) const

Resample all Partials in the specified (half-open) range using this Resampler's stored sampling interval, so that the Breakpoints in the Partial envelopes will all lie on a common temporal grid. The Breakpoint times in the resampled Partial will comprise a contiguous sequence of integer multiples of the sampling interval, beginning with the multiple nearest to the Partial's start time and ending with the multiple nearest to the Partial's end time. Resampling is performed in-place.

Parameters:

begin is the beginning of the range of Partials to resample

end is (one-past) the end of the range of Partials to resample

If compiled with NO_TEMPLATE_MEMBERS defined, then begin and end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 186 of file Resampler.h.

6.134.3.5. void Loris::Resampler::resample (Partial & *p*) const

is performed in-place.

Parameters:

p is the Partial to resample

Referenced by operator()(), and resample().

6.134.4. Member Data Documentation

6.134.4.1. double Loris::Resampler::interval_ [private]

the resampling interval in seconds

Definition at line 161 of file Resampler.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Resampler.h

6.135. csound::Rescale Class Reference

`#include <Rescale.hpp>`

Inheritance diagram for csound::Rescale::

6.135.1. Detailed Description

Rescales all child events to fit a bounding hypercube in music space. No, some, or all dimensions may be rescaled to fit the minimum alone, the range alone, or both the minimum and the range.

Definition at line 42 of file Rescale.hpp.

Public Member Functions

- Rescale ()
- virtual ~Rescale ()
- virtual void initialize ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual void setRescale (int dimension, bool rescaleMinimum, bool rescaleRange, double targetMinimum, double targetRange)
- virtual void getRescale (int dimension, bool &rescaleMinimum, bool &rescaleRange, double &targetMinimum, double &targetRange)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- Score score
- ublas::matrix< double > localCoordinates

Static Private Attributes

- static bool initialized
- static std::map< std::string, size_t > dimensions

6.135.2. Constructor & Destructor Documentation

6.135.2.1. csound::Rescale::Rescale ()

6.135.2.2. virtual csound::Rescale::~Rescale () [virtual]

6.135.3. Member Function Documentation

6.135.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.135.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.135.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.135.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.135.3.5. virtual void csound::Rescale::getRescale (int *dimension*, bool & *rescaleMinimum*, bool & *rescaleRange*, double & *targetMinimum*, double & *targetRange*) [virtual]

6.135.3.6. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.135.3.7. virtual void csound::Rescale::initialize () [virtual]

6.135.3.8. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.135.3.9. virtual void csound::Rescale::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::ScoreNode.

6.135.3.10. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.135.3.11. virtual void csound::Rescale::setRescale (int *dimension*, bool *rescaleMinimum*, bool *rescaleRange*, double *targetMinimum*, double *targetRange*) [virtual]

6.135.3.12. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.135.4. Member Data Documentation

6.135.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.135.4.2. std::map<std::string, size_t> csound::Rescale::dimensions [static, private]

Definition at line 46 of file Rescale.hpp.

6.135.4.3. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.135.4.4. bool csound::Rescale::initialized [static, private]

Definition at line 45 of file Rescale.hpp.

6.135.4.5. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

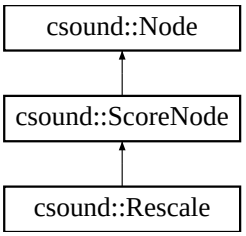
Definition at line 52 of file Node.hpp.

6.135.4.6. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Rescale.hpp


csound::Node
csound::ScoreNode
csound::Rescale

6.136. RTCLOCK_S Struct Reference

`#include <csound.h>`

Public Attributes

- int_least64_t starttime_real
- int_least64_t starttime_CPU

6.136.1. Member Data Documentation

6.136.1.1. int_least64_t RTCLOCK_S::starttime_CPU

Definition at line 904 of file csound.h.

6.136.1.2. int_least64_t RTCLOCK_S::starttime_real

Definition at line 903 of file csound.h.

The documentation for this struct was generated from the following file:

- H/csound.h

6.137. Loris::RuntimeError Class Reference

`#include <Exception.h>`

Inheritance diagram for Loris::RuntimeError::

6.137.1. Detailed Description

Class of exceptions thrown when an unanticipated runtime error is encountered.

Definition at line 240 of file Exception.h.

Public Member Functions

- RuntimeError (const std::string &str, const std::string &where="")
- const char * what (void) const throw ()
- Exception & append (const std::string &str)
- const std::string & str (void) const

Protected Attributes

- std::string _sbuf

6.137.2. Constructor & Destructor Documentation

6.137.2.1. Loris::RuntimeError::RuntimeError (const std::string & *str*, const std::string & *where* = "") [inline]

string automatically using __FILE__ and __LINE__.

Parameters:

str is a string describing the exceptional condition

where is an option string describing the location in the source code from which the exception was thrown (generated automatically by the Throw macro).

Definition at line 253 of file Exception.h.

6.137.3. Member Function Documentation

6.137.3.1. Exception& Loris::Exception::append (const std::string & *str*) [inherited]

Append the specified string to this Exception's description, and return a reference to this Exception.

Parameters:

str is text to append to the exception description

Returns:

a reference to this Exception.

6.137.3.2. const std::string& Loris::Exception::str (void) const [inline, inherited]

Return a read-only refernce to this Exception's description string.

Returns:
a string describing the exceptional condition

Definition at line 92 of file Exception.h.

References Loris::Exception::_sbuf.

6.137.3.3. const char* Loris::Exception::what (void) const throw () [inline, inherited]

C-style string (char pointer). Overrides std::exception::what.

Returns:
a C-style string describing the exceptional condition.

Definition at line 79 of file Exception.h.

References Loris::Exception::_sbuf.

6.137.4. Member Data Documentation

6.137.4.1. std::string Loris::Exception::_sbuf [protected, inherited]

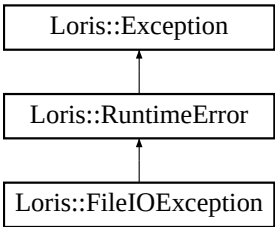
string for storing the exception description

Definition at line 101 of file Exception.h.

Referenced by Loris::Exception::str(), and Loris::Exception::what().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Exception.h


Loris::Exception
Loris::RuntimeError
Loris::FileIOException

6.138. csound::Score Class Reference

`#include <Score.hpp>`

6.138.1. Detailed Description

Base class for collections of events in music space. Can order events by time.

The implementation is a std::vector of Events. The elements of the vector are value objects, not references.

Definition at line 50 of file Score.hpp.

Public Member Functions

- Score ()
- virtual ~Score ()
- virtual void initialize ()
- virtual void append (Event event)
- virtual void append (double time, double duration, double status, double channel, double key, double velocity, double phase=0, double pan=0, double depth=0, double height=0, double pitches=4095)
- virtual void load (std::string filename)
- virtual void load (std::istream &stream)
- virtual void load (MidiFile &midiFile)
- virtual void save (std::string filename)
- virtual void save (std::ostream &stream)
- virtual void save (MidiFile &midiFile)
- virtual void findScale ()
- virtual void rescale ()
- virtual void rescale (Event &event)
- virtual void sort ()
- virtual void dump (std::ostream &stream)
- virtual std::string toString ()
- virtual double getDuration ()
- virtual void rescale (int dimension, bool rescaleMinimum, double minimum, bool rescaleRange=false, double range=0.0)
- virtual std::string getCsoundScore (double tonesPerOctave=12.0, bool conformPitches=false)

Static Public Member Functions

- static void getScale (std::vector< Event > &score, int dimension, size_t beginAt, size_t endAt, double &minimum, double &range)
- static void setScale (std::vector< Event > &score, int dimension, bool rescaleMinimum, bool rescaleRange, size_t beginAt, size_t endAt, double targetMinimum, double targetRange)

Public Attributes

- Event scaleTargetMinima
- std::vector< bool > rescaleMinima
- Event scaleTargetRanges
- std::vector< bool > rescaleRanges

- Event scaleActualMinima
- Event scaleActualRanges
- MidiFile midifile

Protected Member Functions

- void createMusicModel ()

6.138.2. Constructor & Destructor Documentation

6.138.2.1. csound::Score::Score ()

6.138.2.2. virtual csound::Score::~Score () [virtual]

6.138.3. Member Function Documentation

6.138.3.1. virtual void csound::Score::append (double *time*, double *duration*, double *status*, double *channel*, double *key*, double *velocity*, double *phase* = 0, double *pan* = 0, double *depth* = 0, double *height* = 0, double *pitches* = 4095) [virtual]

6.138.3.2. virtual void csound::Score::append (Event *event*) [virtual]

6.138.3.3. void csound::Score::createMusicModel () [protected]

6.138.3.4. virtual void csound::Score::dump (std::ostream & *stream*) [virtual]

6.138.3.5. virtual void csound::Score::findScale () [virtual]

6.138.3.6. virtual std::string csound::Score::getCsoundScore (double *tonesPerOctave* = 12.0, bool *conformPitches* = false) [virtual]

Translate the Silence events in this to a Csound score, that is, to a list of i statements. The Silence events are rounded off to the nearest equally tempered pitch by the specified number of tones per octave; if this argument is zero, the pitch is not tempered. The Silence events are conformed to the nearest pitch-class set in the pitch-class set dimension of the event, if the conform pitches argument is true; otherwise, the pitches are not conformed.

6.138.3.7. virtual double csound::Score::getDuration () [virtual]

6.138.3.8. static void csound::Score::getScale (std::vector< Event > & *score*, int *dimension*, size_t *beginAt*, size_t *endAt*, double & *minimum*, double & *range*) [static]

6.138.3.9. virtual void csound::Score::initialize () [virtual]

6.138.3.10. virtual void csound::Score::load (MidiFile & *midiFile*) [virtual]

6.138.3.11. virtual void csound::Score::load (std::istream & *stream*) [virtual]

6.138.3.12. virtual void csound::Score::load (std::string *filename*) [virtual]

6.138.3.13. virtual void csound::Score::rescale (int *dimension*, bool *rescaleMinimum*, double *minimum*, bool *rescaleRange* = false, double *range* = 0.0) [virtual]

6.138.3.14. virtual void csound::Score::rescale (Event & *event*) [virtual]

6.138.3.15. virtual void csound::Score::rescale () [virtual]

6.138.3.16. virtual void csound::Score::save (MidiFile & *midiFile*) [virtual]

6.138.3.17. virtual void csound::Score::save (std::ostream & *stream*) [virtual]

6.138.3.18. virtual void csound::Score::save (std::string *filename*) [virtual]

6.138.3.19. static void csound::Score::setScale (std::vector< Event > & *score*, int *dimension*, bool *rescaleMinimum*, bool *rescaleRange*, size_t *beginAt*, size_t *endAt*, double *targetMinimum*, double *targetRange*) [static]

6.138.3.20. virtual void csound::Score::sort () [virtual]

Sort all events in the score by time, instrument number, pitch, duration, loudness, and other dimensions as given by Event::SORT_ORDER.

6.138.3.21. virtual std::string csound::Score::toString () [virtual]

6.138.4. Member Data Documentation

6.138.4.1. MidiFile csound::Score::midifile

Definition at line 62 of file Score.hpp.

6.138.4.2. std::vector<bool> csound::Score::rescaleMinima

Definition at line 57 of file Score.hpp.

6.138.4.3. std::vector<bool> csound::Score::rescaleRanges

Definition at line 59 of file Score.hpp.

6.138.4.4. Event csound::Score::scaleActualMinima

Definition at line 60 of file Score.hpp.

6.138.4.5. Event csound::Score::scaleActualRanges

Definition at line 61 of file Score.hpp.

6.138.4.6. Event csound::Score::scaleTargetMinima

Definition at line 56 of file Score.hpp.

6.138.4.7. Event csound::Score::scaleTargetRanges

Definition at line 58 of file Score.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Score.hpp

6.139. csound::ScoreNode Class Reference

`#include <ScoreNode.hpp>`

Inheritance diagram for csound::ScoreNode::

6.139.1. Detailed Description

Node class that produces events from the contained score, which can be built up programmatically or imported from a standard MIDI file.

Definition at line 43 of file ScoreNode.hpp.

Public Member Functions

- ScoreNode ()
- virtual ~ScoreNode ()
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- Score score
- ublas::matrix< double > localCoordinates

6.139.2. Constructor & Destructor Documentation

6.139.2.1. csound::ScoreNode::ScoreNode ()

6.139.2.2. virtual csound::ScoreNode::~ScoreNode () [virtual]

6.139.3. Member Function Documentation

6.139.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.139.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.139.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.139.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.139.3.5. virtual Score& csound::ScoreNode::getScore () [virtual]

6.139.3.6. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.139.3.7. virtual void csound::ScoreNode::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual]

The default implementation does nothing.

Reimplemented from csound::Node.

Reimplemented in csound::Cell, csound::Hocket, csound::MCRM, and csound::Rescale.

6.139.3.8. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.139.3.9. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.139.4. Member Data Documentation

6.139.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

Definition at line 57 of file Node.hpp.

6.139.4.2. std::string csound::ScoreNode::importFilename

Definition at line 49 of file ScoreNode.hpp.

6.139.4.3. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

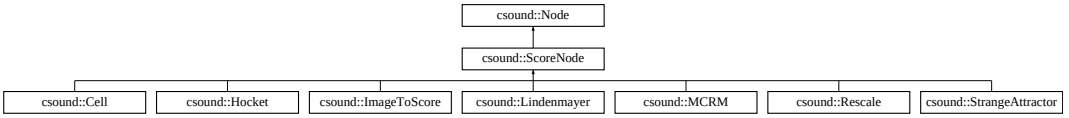
Definition at line 52 of file Node.hpp.

6.139.4.4. Score csound::ScoreNode::score [protected]

Definition at line 47 of file ScoreNode.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/ScoreNode.hpp


csound::Node
csound::ScoreNode
csound::Cell
csound::Hocket
csound::ImageToScore
csound::Lindenmayer
csound::MCRM
csound::Rescale
csound::StrangeAttractor

6.140. Loris::SdifFile Class Reference

`#include <SdifFile.h>`

Public Types

- typedef std::vector< Marker > markers_type
- typedef PartialList partials_type

Public Member Functions

- SdifFile (const std::string &filename)
- template<typename Iter> SdifFile (Iter begin_partials, Iter end_partials)
- SdifFile (void)
- markers_type & markers (void)
- const markers_type & markers (void) const
- partials_type & partials (void)
- const partials_type & partials (void) const
- void addPartial (const Loris::Partial &p)
- template<typename Iter> void addPartials (Iter begin_partials, Iter end_partials)
- void write (const std::string &path)
- void write1TRC (const std::string &path)

Static Public Member Functions

- static void Export (const std::string &filename, const PartialList &plist, const bool enhanced=true)

Private Attributes

- partials_type partials_
- markers_type markers_

6.140.1. Member Typedef Documentation

6.140.1.1. typedef std::vector< Marker > Loris::SdifFile::markers_type

Definition at line 103 of file SdifFile.h.

6.140.1.2. typedef PartialList Loris::SdifFile::partials_type

Definition at line 104 of file SdifFile.h.

6.140.2. Constructor & Destructor Documentation

6.140.2.1. Loris::SdifFile::SdifFile (const std::string & *filename*) [explicit]

6.140.2.2. template<typename Iter> Loris::SdifFile::SdifFile (Iter *begin_partials*, Iter *end_partials*)

Definition at line 208 of file SdifFile.h.

6.140.2.3. Loris::SdifFile::SdifFile (void)

6.140.3. Member Function Documentation

6.140.3.1. void Loris::SdifFile::addPartial (const Loris::Partial & *p*)

6.140.3.2. template<typename Iter> void Loris::SdifFile::addPartials (Iter *begin_partials*, Iter *end_partials*)

Definition at line 228 of file SdifFile.h.

6.140.3.3. static void Loris::SdifFile::Export (const std::string & *filename*, const PartialList & *plist*, const bool *enhanced* = `true`) `[static]`

6.140.3.4. const markers_type& Loris::SdifFile::markers (void) const

6.140.3.5. markers_type& Loris::SdifFile::markers (void)

6.140.3.6. const partials_type& Loris::SdifFile::partials (void) const

6.140.3.7. partials_type& Loris::SdifFile::partials (void)

6.140.3.8. void Loris::SdifFile::write (const std::string & *path*)

6.140.3.9. void Loris::SdifFile::write1TRC (const std::string & *path*)

6.140.4. Member Data Documentation

6.140.4.1. markers_type Loris::SdifFile::markers_ `[private]`

Definition at line 191 of file SdifFile.h.

6.140.4.2. partials_type Loris::SdifFile::partials_ `[private]`

Definition at line 190 of file SdifFile.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/SdifFile.h

6.141. csound::Sequence Class Reference

`#include <Sequence.hpp>`

Inheritance diagram for csound::Sequence::

6.141.1. Detailed Description

Node that creates a temporal sequence of child nodes.

Definition at line 40 of file Sequence.hpp.

Public Member Functions

- Sequence ()
- virtual ~Sequence ()
- virtual ublas::matrix< double > Sequence::traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::vector< Node * > children

Protected Attributes

- ublas::matrix< double > localCoordinates

6.141.2. Constructor & Destructor Documentation

6.141.2.1. csound::Sequence::Sequence ()

6.141.2.2. virtual csound::Sequence::~Sequence () [virtual]

6.141.3. Member Function Documentation

6.141.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.141.3.2. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.141.3.3. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.141.3.4. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.141.3.5. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.141.3.6. virtual void csound::Node::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual, inherited]

The default implementation does nothing.

Reimplemented in csound::Cell, csound::CounterpointNode, csound::Hocket, csound::MCRM, csound::Random, csound::Rescale, and csound::ScoreNode.

6.141.3.7. virtual ublas::matrix<double> csound::Sequence::Sequence::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual]

The notes produced by the first child node have a total duration. The notes produced by the second child node are shifted forward in time by that duration, and so on, to create a strict temporal sequence of child nodes.

6.141.3.8. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.141.3.9. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.141.4. Member Data Documentation

6.141.4.1. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.

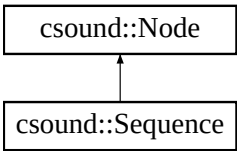
Definition at line 57 of file Node.hpp.

6.141.4.2. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Sequence.hpp


csound::Node
csound::Sequence

6.142. csound::Shell Class Reference

`#include <Shell.hpp>`

Inheritance diagram for csound::Shell::

6.142.1. Detailed Description

Provide a shell in which Python scripts can be loaded, saved, and executed.

Definition at line 41 of file Shell.hpp.

Public Member Functions

- Shell ()
- virtual ~Shell ()
- virtual void open ()
- virtual void close ()
- virtual void main (int argc, char **argv)
- virtual void initialize ()
- virtual void clear ()
- virtual void setFilename (std::string filename)
- virtual std::string getFilename () const
- virtual std::string getOutputSoundfileName () const
- virtual std::string getMidiFilename () const
- virtual std::string getScript () const
- virtual void setScript (std::string text)
- virtual void load (std::string filename)
- virtual void loadAppend (std::string filename)
- virtual void save (std::string filename) const
- virtual void save () const
- virtual int runScript ()
- virtual int runScript (std::string script)
- virtual void stop ()

Static Public Member Functions

- static std::string generateFilename ()

Protected Attributes

- std::string filename
- std::string script

6.142.2. Constructor & Destructor Documentation

6.142.2.1. csound::Shell::Shell ()

6.142.2.2. virtual csound::Shell::~Shell () [virtual]

6.142.3. Member Function Documentation

6.142.3.1. virtual void csound::Shell::clear () [virtual]

6.142.3.2. virtual void csound::Shell::close () [virtual]

6.142.3.3. static std::string csound::Shell::generateFilename () [static]

6.142.3.4. virtual std::string csound::Shell::getFilename () const [virtual]

6.142.3.5. virtual std::string csound::Shell::getMidiFilename () const [virtual]

6.142.3.6. virtual std::string csound::Shell::getOutputSoundfileName () const [virtual]

6.142.3.7. virtual std::string csound::Shell::getScript () const [virtual]

6.142.3.8. virtual void csound::Shell::initialize () [virtual]

6.142.3.9. virtual void csound::Shell::load (std::string *filename*) [virtual]

6.142.3.10. virtual void csound::Shell::loadAppend (std::string *filename*) [virtual]

6.142.3.11. virtual void csound::Shell::main (int *argc*, char ** *argv*) [virtual]

6.142.3.12. virtual void csound::Shell::open () [virtual]

Reimplemented in CsoundVST.

6.142.3.13. virtual int csound::Shell::runScript (std::string *script*) [virtual]

6.142.3.14. virtual int csound::Shell::runScript () [virtual]

6.142.3.15. virtual void csound::Shell::save () const [virtual]

6.142.3.16. virtual void csound::Shell::save (std::string *filename*) const [virtual]

6.142.3.17. virtual void csound::Shell::setFilename (std::string *filename*) [virtual]

6.142.3.18. virtual void csound::Shell::setScript (std::string *text*) [virtual]

6.142.3.19. virtual void csound::Shell::stop () [virtual]

6.142.4. Member Data Documentation

6.142.4.1. std::string csound::Shell::filename [protected]

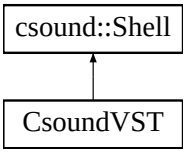
Definition at line 44 of file Shell.hpp.

6.142.4.2. std::string csound::Shell::script [protected]

Definition at line 45 of file Shell.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Shell.hpp


csound::Shell
CsoundVST

6.143. Loris::Sieve Class Reference

`#include <Sieve.h>`

6.143.1. Detailed Description

Class Sieve represents an algorithm for identifying channelized (see Channelizer) Partials that overlap in time, and selecting the longer one to represent the channel. The identification of overlap includes the time needed for Partials to fade to and from zero amplitude in synthesis (

See also:
Synthesizer) or distillation. (
Distiller)

In some cases, the energy redistribution effected by the distiller (see Distiller) is undesirable. In such cases, the partials can be sifted before distillation. The sifting process in Loris identifies all the partials that would be rejected (and converted to noise energy) by the distiller and assigns them a label of 0. These sifted partials can then be identified and treated sepearately or removed altogether, or they can be passed through the distiller unlabeled, and crossfaded in the morphing process (

See also:
Morpher).

Definition at line 62 of file Sieve.h.

Public Member Functions

- Sieve (double partialFadeTime=0.001)
- template<typename Iter> void sift (Iter sift_begin, Iter sift_end)

Static Public Member Functions

- template<typename Iter> static void sift (Iter sift_begin, Iter sift_end, double partialFadeTime)

Private Member Functions

- void sift_ptrs (PartialPtrs &ptrs)

Private Attributes

- double _fadeTime

6.143.2. Constructor & Destructor Documentation

6.143.2.1. Loris::Sieve::Sieve (double *partialFadeTime* = 0.001) [explicit]

Construct a new Sieve using the specified partial fade time. If unspecified, the fade time defaults to one millisecond (0.001 s).

Parameters:

partialFadeTime is the extra time (in seconds) added to each end of a Partial to accomodate the fade to and from zero amplitude. Default is 0.001 (one millisecond). The Partial fade time must be non-negative.

Exceptions:

InvalidArgument if partialFadeTime is negative.

6.143.3. Member Function Documentation

6.143.3.1. template<typename Iter> void Loris::Sieve::sift (Iter *sift_begin*, Iter *sift_end*, double *partialFadeTime*) [static]

Static member that constructs an instance and applies it to a sequence of Partials. Construct a Sieve using the specified Partial fade time (in seconds), and use it to sift a sequence of Partials.

Parameters:

sift_begin is the beginning of the range of Partials to sift

sift_end is (one-past) the end of the range of Partials to sift

partialFadeTime is the extra time (in seconds) added to each end of a Partial to accomodate the fade to and from zero amplitude. The Partial fade time must be non-negative.

Exceptions:

InvalidArgument if partialFadeTime is negative.

If compiled with NO_TEMPLATE_MEMBERS defined, then begin and end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 206 of file Sieve.h.

References sift().

6.143.3.2. template<typename Iter> void Loris::Sieve::sift (Iter *sift_begin*, Iter *sift_end*)

Parameters:

sift_begin is the beginning of the range of Partials to sift

sift_end is (one-past) the end of the range of Partials to sift

If compiled with NO_TEMPLATE_MEMBERS defined, then sift_begin and sift_end must be PartialList::iterators, otherwise they can be any type of iterators over a sequence of Partials.

Definition at line 173 of file Sieve.h.

References Loris::fillPartialPtrs().

Referenced by sift().

6.143.3.3. void Loris::Sieve::sift_ptrs (PartialPtrs & *ptrs*) [private]

Sift labeled Partials. If any two Partials having the same (non-zero) label overlap in time (where overlap includes the fade time at both ends of each Partial), then set the label of the Partial having the shorter duration to zero. Sifting is performed on a collection of pointers to Partials so that the it can be performed without changing the order of the Partials in the sequence.

Parameters:

ptrs is a collection of pointers to the Partials in the sequence to be sifted.

6.143.4. Member Data Documentation

6.143.4.1. double Loris::Sieve::_fadeTime [private]

Definition at line 66 of file Sieve.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Sieve.h

6.144. SNDMEMFILE_ Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char * name
- SNDMEMFILE_ * nxt
- char * fullName
- size_t nFrames
- double sampleRate
- int nChannels
- int sampleFormat
- int fileType
- int loopMode
- double startOffs
- double loopStart
- double loopEnd
- double baseFreq
- double scaleFac
- float data [1]

6.144.1. Member Data Documentation

6.144.1.1. double SNDMEMFILE_::baseFreq

Definition at line 521 of file csoundCore.h.

6.144.1.2. float SNDMEMFILE_::data[1]

Definition at line 523 of file csoundCore.h.

6.144.1.3. int SNDMEMFILE_::fileType

Definition at line 511 of file csoundCore.h.

6.144.1.4. char* SNDMEMFILE_::fullName

Definition at line 506 of file csoundCore.h.

6.144.1.5. double SNDMEMFILE_::loopEnd

Definition at line 520 of file csoundCore.h.

6.144.1.6. int SNDMEMFILE_::loopMode

Definition at line 512 of file csoundCore.h.

6.144.1.7. double SNDMEMFILE_::loopStart

Definition at line 519 of file csoundCore.h.

6.144.1.8. char* SNDMEMFILE_::name

Definition at line 504 of file csoundCore.h.

6.144.1.9. int SNDMEMFILE_::nChannels

Definition at line 509 of file csoundCore.h.

6.144.1.10. size_t SNDMEMFILE_::nFrames

Definition at line 507 of file csoundCore.h.

6.144.1.11. struct SNDMEMFILE_* SNDMEMFILE_::nxt

Definition at line 505 of file csoundCore.h.

6.144.1.12. int SNDMEMFILE_::sampleFormat

Definition at line 510 of file csoundCore.h.

6.144.1.13. double SNDMEMFILE_::sampleRate

Definition at line 508 of file csoundCore.h.

6.144.1.14. double SNDMEMFILE_::scaleFac

Definition at line 522 of file csoundCore.h.

6.144.1.15. double SNDMEMFILE_::startOffs

Definition at line 518 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.145. Loris::SosEnvelopesCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- Int_32 signature
- Int_32 enhanced
- Int_32 validPartials
- Int_32 resolution
- Int_32 quasiHarmonic

6.145.1. Member Data Documentation

6.145.1.1. Int_32 Loris::SosEnvelopesCk::enhanced

Definition at line 174 of file AiffData.h.

6.145.1.2. CkHeader Loris::SosEnvelopesCk::header

Definition at line 172 of file AiffData.h.

6.145.1.3. Int_32 Loris::SosEnvelopesCk::quasiHarmonic

Definition at line 181 of file AiffData.h.

6.145.1.4. Int_32 Loris::SosEnvelopesCk::resolution

Definition at line 180 of file AiffData.h.

6.145.1.5. Int_32 Loris::SosEnvelopesCk::signature

Definition at line 173 of file AiffData.h.

6.145.1.6. Int_32 Loris::SosEnvelopesCk::validPartials

Definition at line 175 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.146. Loris::SoundDataCk Struct Reference

`#include <AiffData.h>`

Public Attributes

- CkHeader header
- Uint_32 offset
- Uint_32 blockSize
- std::vector< Byte > sampleBytes

6.146.1. Member Data Documentation

6.146.1.1. Uint_32 Loris::SoundDataCk::blockSize

Definition at line 125 of file AiffData.h.

6.146.1.2. CkHeader Loris::SoundDataCk::header

Definition at line 123 of file AiffData.h.

6.146.1.3. Uint_32 Loris::SoundDataCk::offset

Definition at line 124 of file AiffData.h.

6.146.1.4. std::vector< Byte > Loris::SoundDataCk::sampleBytes

Definition at line 128 of file AiffData.h.

The documentation for this struct was generated from the following file:

- Opcodes/Loris/src/AiffData.h

6.147. Loris::SpcFile Class Reference

`#include <SpcFile.h>`

Public Types

- typedef std::vector< Marker > markers_type
- typedef std::vector< Partial > partials_type

Public Member Functions

- SpcFile (const std::string &filename)
- template<typename Iter> SpcFile (Iter begin_partials, Iter end_partials, double midiNoteNum=60)
- SpcFile (double midiNoteNum=60)
- markers_type & markers (void)
- const markers_type & markers (void) const
- double midiNoteNumber (void) const
- const partials_type & partials (void) const
- double sampleRate (void) const
- void addPartial (const Loris::Partial &p)
- void addPartial (const Loris::Partial &p, int label)
- template<typename Iter> void addPartials (Iter begin_partials, Iter end_partials)
- void setMidiNoteNumber (double nn)
- void setSampleRate (double rate)
- void write (const std::string &filename, double endApproachTime=0)
- void writeSinusoidal (const std::string &filename, double endApproachTime=0)
- void write (const std::string &filename, bool enhanced, double endApproachTime=0)

Private Member Functions

- void readSpcData (const std::string &filename)
- void growPartials (partials_type::size_type sz)

Private Attributes

- partials_type partials_
- markers_type markers_
- double notenum_
- double rate_

Static Private Attributes

- static const int MinNumPartials
- static const double DefaultRate

6.147.1. Member Typedef Documentation

6.147.1.1. typedef std::vector< Marker > Loris::SpcFile::markers_type

Definition at line 70 of file SpcFile.h.

6.147.1.2. typedef std::vector< Partial > Loris::SpcFile::partials_type

Definition at line 71 of file SpcFile.h.

6.147.2. Constructor & Destructor Documentation

6.147.2.1. Loris::SpcFile::SpcFile (const std::string & *filename*) [explicit]

6.147.2.2. template<typename Iter> Loris::SpcFile::SpcFile (Iter *begin_partials*, Iter *end_partials*, double *midiNoteNum* = 60)

Definition at line 265 of file SpcFile.h.

References addPartials(), growPartials(), and MinNumPartials.

6.147.2.3. Loris::SpcFile::SpcFile (double *midiNoteNum* = 60) [explicit]

6.147.3. Member Function Documentation

6.147.3.1. void Loris::SpcFile::addPartial (const Loris::Partial & *p*, int *label*)

6.147.3.2. void Loris::SpcFile::addPartial (const Loris::Partial & *p*)

6.147.3.3. template<typename Iter> void Loris::SpcFile::addPartials (Iter *begin_partials*, Iter *end_partials*)

Definition at line 298 of file SpcFile.h.

Referenced by SpcFile().

6.147.3.4. void Loris::SpcFile::growPartials (partials_type::size_type *sz*) [private]

Referenced by SpcFile().

6.147.3.5. const markers_type& Loris::SpcFile::markers (void) const

6.147.3.6. markers_type& Loris::SpcFile::markers (void)

6.147.3.7. double Loris::SpcFile::midiNoteNumber (void) const

6.147.3.8. const partials_type& Loris::SpcFile::partials (void) const

6.147.3.9. void Loris::SpcFile::readSpcData (const std::string & *filename*) [private]

6.147.3.10. double Loris::SpcFile::sampleRate (void) const

6.147.3.11. void Loris::SpcFile::setMidiNoteNumber (double *nn*)

6.147.3.12. void Loris::SpcFile::setSampleRate (double *rate*)

6.147.3.13. void Loris::SpcFile::write (const std::string & *filename*, bool *enhanced*, double *endApproachTime* = 0)

6.147.3.14. void Loris::SpcFile::write (const std::string & *filename*, double *endApproachTime* = 0)

6.147.3.15. void Loris::SpcFile::writeSinusoidal (const std::string & *filename*, double *endApproachTime* = 0)

6.147.4. Member Data Documentation

6.147.4.1. const double Loris::SpcFile::DefaultRate [static, private]

Definition at line 244 of file SpcFile.h.

6.147.4.2. markers_type Loris::SpcFile::markers_ [private]

Definition at line 239 of file SpcFile.h.

6.147.4.3. const int Loris::SpcFile::MinNumPartials [static, private]

Definition at line 243 of file SpcFile.h.

Referenced by SpcFile().

6.147.4.4. double Loris::SpcFile::notenum_ [private]

Definition at line 241 of file SpcFile.h.

6.147.4.5. partials_type Loris::SpcFile::partials_ [private]

Definition at line 238 of file SpcFile.h.

6.147.4.6. double Loris::SpcFile::rate_ [private]

Definition at line 241 of file SpcFile.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/SpcFile.h

6.148. SPECDAT Struct Reference

`#include <csoundCore.h>`

Public Attributes

- long ktimstamp
- long ktimprd
- long npts
- long nfreqs
- long dbout
- DOWNDAT * downsrcp
- AUXCH auxch

6.148.1. Member Data Documentation

6.148.1.1. AUXCH SPECDAT::auxch

Definition at line 356 of file csoundCore.h.

6.148.1.2. long SPECDAT::dbout

Definition at line 354 of file csoundCore.h.

6.148.1.3. DOWNDAT* SPECDAT::downsrcp

Definition at line 355 of file csoundCore.h.

6.148.1.4. long SPECDAT::ktimprd

Definition at line 353 of file csoundCore.h.

6.148.1.5. long SPECDAT::ktimstamp

Definition at line 353 of file csoundCore.h.

6.148.1.6. long SPECDAT::nfreqs

Definition at line 354 of file csoundCore.h.

6.148.1.7. long SPECDAT::npts

Definition at line 354 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.149. Loris::SpectralPeakSelector Class Reference

`#include <SpectralPeakSelector.h>`

Public Member Functions

- SpectralPeakSelector (double srate, double res)
- Peaks & extractPeaks (ReassignedSpectrum &spectrum, double minFrequency, double maxTimeOffset)
- Peaks::iterator thinPeaks (double ampFloordB, double fadeRangedB, double frameTime)
- Peaks & peaks (void)

Private Attributes

- Peaks peaks_
- double freqResolution
- double sampleRate

6.149.1. Constructor & Destructor Documentation

6.149.1.1. Loris::SpectralPeakSelector::SpectralPeakSelector (double *srate*, double *res*)

6.149.2. Member Function Documentation

6.149.2.1. Peaks& Loris::SpectralPeakSelector::extractPeaks (ReassignedSpectrum & *spectrum*, double *minFrequency*, double *maxTimeOffset*)

6.149.2.2. Peaks& Loris::SpectralPeakSelector::peaks (void) [inline]

Definition at line 75 of file SpectralPeakSelector.h.

References peaks_.

6.149.2.3. Peaks::iterator Loris::SpectralPeakSelector::thinPeaks (double *ampFloordB*, double *fadeRangedB*, double *frameTime*)

6.149.3. Member Data Documentation

6.149.3.1. double Loris::SpectralPeakSelector::freqResolution [private]

Definition at line 81 of file SpectralPeakSelector.h.

6.149.3.2. Peaks Loris::SpectralPeakSelector::peaks_ [private]

Definition at line 79 of file SpectralPeakSelector.h.

Referenced by peaks().

6.149.3.3. double Loris::SpectralPeakSelector::sampleRate [private]

Definition at line 82 of file SpectralPeakSelector.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/SpectralPeakSelector.h

6.150. csound::StrangeAttractor Class Reference

`#include <StrangeAttractor.hpp>`

Inheritance diagram for csound::StrangeAttractor::

6.150.1. Detailed Description

Generates notes by searching for a chaotic dynamical system defined by a polynomial equation or partial differential equation using Julien C. Sprott's Lyupanov exponent search, or by translating a known chaotic dynamical system into music, by interpreting each iteration of the system as a note. The time of the note can be represented either as the order of iteration, or as a dimension of the attractor. See Julien C. Sprott's book "Strange Attractors".

Definition at line 56 of file StrangeAttractor.hpp.

Public Member Functions

- StrangeAttractor (void)
- virtual ~StrangeAttractor (void)
- virtual void setCode (std::string code)
- virtual std::string getCode () const
- virtual void setIterationCount (size_t iterationCount)
- virtual size_t getIterationCount () const
- virtual void setIteration (size_t iteration)
- virtual size_t getIteration () const
- virtual void setAttractorType (int attractorType)
- virtual int getAttractorType () const
- virtual void setScoreType (int attractorType)
- virtual int getScoreType () const
- virtual void initialize ()
- virtual void reinitialize ()
- virtual void reset ()
- virtual void codeRandomize ()
- virtual void specialFunctions ()
- virtual void getDimensionAndOrder ()
- virtual void getCoefficients ()
- virtual void shuffleRandomNumbers ()
- virtual void calculateLyupanovExponent ()
- virtual void calculateFractalDimension ()
- virtual double getFractalDimension () const
- virtual double getLyupanovExponent () const
- virtual void setX (double X)
- virtual double getX () const
- virtual void setY (double X)
- virtual double getY () const
- virtual void setZ (double X)
- virtual double getZ () const
- virtual void setW (double X)
- virtual double getW () const
- virtual bool searchForAttractor ()
- virtual bool evaluateAttractor ()
- virtual void iterate ()

- virtual void generate ()
- virtual void render (int N, double X, double Y, double Z, double W)
- virtual void setDimensionCount (int D)
- virtual int getDimensionCount () const
- virtual void produceOrTransform (Score &score, size_t beginAt, size_t endAt, const ublas::matrix< double > &coordinates)
- virtual Score & getScore ()
- virtual ublas::matrix< double > getLocalCoordinates () const
- virtual ublas::matrix< double > traverse (const ublas::matrix< double > &global-Coordinates, Score &score)
- virtual ublas::matrix< double > Node::createTransform ()
- virtual void clear ()
- virtual double & element (size_t row, size_t column)
- virtual void setElement (size_t row, size_t column, double value)
- virtual void addChild (Node *node)

Public Attributes

- std::string importFilename
- std::vector< Node * > children

Protected Attributes

- std::string code
- Random random
- std::string filename
- int scoreType
- int NMAX
- std::vector< double > A
- double AL
- double COSAL
- int D
- int DD
- double D2
- double D2MAX
- double decibels
- double DF
- double DL2
- double DLW
- double DLX
- double DLY
- double DLZ
- double DUM
- double DW
- double DX
- double DY
- double DZ
- double EPS
- double F
- int I
- double instrument
- int I1

- int I2
- int I3
- int I4
- int I5
- int J
- double L
- double duration
- double LSUM
- int M
- double MX
- double MY
- int N
- double N1
- double N2
- double NL
- int O
- double octave
- int ODE
- int OMAX
- int P
- double x
- double pitchClassSet
- int PREV
- double PT
- double RAN
- double RS
- double SH
- double SINAL
- double time
- double SW
- int T
- double TIA
- double TT
- int TWOD
- std::vector< double > V
- double W
- double WE
- double WMAX
- double WMIN
- double WNEW
- double WP
- std::vector< double > WS
- double WSAVE
- double X
- double XA
- double XE
- double XH
- double XL
- double XMAX
- double XMIN
- std::vector< double > XN
- double XNEW

- double XP
- std::vector< double > XS
- double XSAVE
- double XW
- std::vector< double > XY
- double XZ
- double Y
- double YA
- double YE
- double YH
- double YL
- double YMAX
- double YMIN
- double YNEW
- double YP
- std::vector< double > YS
- double YSAVE
- double YW
- double YZ
- double Z
- double ZA
- double ZE
- double ZMAX
- double ZMIN
- double ZNEW
- double ZP
- std::vector< double > ZS
- double ZSAVE
- Score score
- ublas::matrix< double > localCoordinates

6.150.2. Constructor & Destructor Documentation

6.150.2.1. csound::StrangeAttractor::StrangeAttractor (void)

6.150.2.2. virtual csound::StrangeAttractor::~StrangeAttractor (void) [virtual]

6.150.3. Member Function Documentation

6.150.3.1. virtual void csound::Node::addChild (Node * *node*) [virtual, inherited]

6.150.3.2. virtual void csound::StrangeAttractor::calculateFractalDimension () [virtual]

6.150.3.3. virtual void csound::StrangeAttractor::calculateLyupanovExponent () [virtual]

6.150.3.4. virtual void csound::Node::clear () [virtual, inherited]

Reimplemented in csound::Lindenmayer, and csound::MusicModel.

6.150.3.5. virtual void csound::StrangeAttractor::codeRandomize () [virtual]

6.150.3.6. virtual double& csound::Node::element (size_t *row*, size_t *column*) [virtual, inherited]

6.150.3.7. virtual bool csound::StrangeAttractor::evaluateAttractor () [virtual]

6.150.3.8. virtual void csound::StrangeAttractor::generate () [virtual]

6.150.3.9. virtual int csound::StrangeAttractor::getAttractorType () const [virtual]

6.150.3.10. virtual std::string csound::StrangeAttractor::getCode () const [virtual]

6.150.3.11. virtual void csound::StrangeAttractor::getCoefficients () [virtual]

6.150.3.12. virtual void csound::StrangeAttractor::getDimensionAndOrder () [virtual]

6.150.3.13. virtual int csound::StrangeAttractor::getDimensionCount () const [virtual]

6.150.3.14. virtual double csound::StrangeAttractor::getFractalDimension () const [virtual]

6.150.3.15. virtual size_t csound::StrangeAttractor::getIteration () const [virtual]

6.150.3.16. virtual size_t csound::StrangeAttractor::getIterationCount () const [virtual]

6.150.3.17. virtual ublas::matrix<double> csound::Node::getLocalCoordinates () const [virtual, inherited]

Returns the local transformation of coordinate system.

Reimplemented in csound::Random.

6.150.3.18. virtual double csound::StrangeAttractor::getLyupanovExponent () const [virtual]

6.150.3.19. virtual Score& csound::ScoreNode::getScore () [virtual, inherited]

6.150.3.20. virtual int csound::StrangeAttractor::getScoreType () const [virtual]

6.150.3.21. virtual double csound::StrangeAttractor::getW () const [virtual]

6.150.3.22. virtual double csound::StrangeAttractor::getX () const [virtual]

6.150.3.23. virtual double csound::StrangeAttractor::getY () const [virtual]

6.150.3.24. virtual double csound::StrangeAttractor::getZ () const [virtual]

6.150.3.25. virtual void csound::StrangeAttractor::initialize () [virtual]

6.150.3.26. virtual void csound::StrangeAttractor::iterate () [virtual]

6.150.3.27. virtual ublas::matrix<double> csound::Node::Node::createTransform () [virtual, inherited]

6.150.3.28. virtual void csound::ScoreNode::produceOrTransform (Score & *score*, size_t *beginAt*, size_t *endAt*, const ublas::matrix< double > & *coordinates*) [virtual, inherited]

The default implementation does nothing.

Reimplemented from csound::Node.

Reimplemented in csound::Cell, csound::Hocket, csound::MCRM, and csound::Rescale.

6.150.3.29. virtual void csound::StrangeAttractor::reinitialize () [virtual]

6.150.3.30. virtual void csound::StrangeAttractor::render (int *N*, double *X*, double *Y*, double *Z*, double *W*) [virtual]

6.150.3.31. virtual void csound::StrangeAttractor::reset () [virtual]

6.150.3.32. virtual bool csound::StrangeAttractor::searchForAttractor () [virtual]

6.150.3.33. virtual void csound::StrangeAttractor::setAttractorType (int *attractorType*) [virtual]

6.150.3.34. virtual void csound::StrangeAttractor::setCode (std::string *code*) [virtual]

6.150.3.35. virtual void csound::StrangeAttractor::setDimensionCount (int *D*) [virtual]

6.150.3.36. virtual void csound::Node::setElement (size_t *row*, size_t *column*, double *value*) [virtual, inherited]

6.150.3.37. virtual void csound::StrangeAttractor::setIteration (size_t *iteration*) [virtual]

6.150.3.38. virtual void csound::StrangeAttractor::setIterationCount (size_t *iterationCount*) [virtual]

6.150.3.39. virtual void csound::StrangeAttractor::setScoreType (int *attractorType*) [virtual]

6.150.3.40. virtual void csound::StrangeAttractor::setW (double *X*) [virtual]

6.150.3.41. virtual void csound::StrangeAttractor::setX (double *X*) [virtual]

6.150.3.42. virtual void csound::StrangeAttractor::setY (double *X*) [virtual]

6.150.3.43. virtual void csound::StrangeAttractor::setZ (double *X*) [virtual]

6.150.3.44. virtual void csound::StrangeAttractor::shuffleRandomNumbers () [virtual]

6.150.3.45. virtual void csound::StrangeAttractor::specialFunctions () [virtual]

6.150.3.46. virtual ublas::matrix<double> csound::Node::traverse (const ublas::matrix< double > & *globalCoordinates*, Score & *score*) [virtual, inherited]

The default implementation postconcatenates its own local coordinate system with the global coordinates, then passes the score and the product of coordinate systems to each child, thus performing a depth-first traversal of the music graph.

Reimplemented in csound::Hocket.

6.150.4. Member Data Documentation

6.150.4.1. std::vector<double> csound::StrangeAttractor::A [protected]

Definition at line 65 of file StrangeAttractor.hpp.

6.150.4.2. double csound::StrangeAttractor::AL [protected]

Definition at line 66 of file StrangeAttractor.hpp.

6.150.4.3. std::vector<Node *> csound::Node::children [inherited]

Child Nodes, if any.
Definition at line 57 of file Node.hpp.

6.150.4.4. std::string csound::StrangeAttractor::code [protected]

Definition at line 60 of file StrangeAttractor.hpp.

6.150.4.5. double csound::StrangeAttractor::COSAL [protected]

Definition at line 67 of file StrangeAttractor.hpp.

6.150.4.6. int csound::StrangeAttractor::D [protected]

Definition at line 68 of file StrangeAttractor.hpp.

6.150.4.7. double csound::StrangeAttractor::D2 [protected]

Definition at line 70 of file StrangeAttractor.hpp.

6.150.4.8. double csound::StrangeAttractor::D2MAX [protected]

Definition at line 71 of file StrangeAttractor.hpp.

6.150.4.9. int csound::StrangeAttractor::DD [protected]

Definition at line 69 of file StrangeAttractor.hpp.

6.150.4.10. double csound::StrangeAttractor::decibels [protected]

Definition at line 72 of file StrangeAttractor.hpp.

6.150.4.11. double csound::StrangeAttractor::DF [protected]

Definition at line 73 of file StrangeAttractor.hpp.

6.150.4.12. double csound::StrangeAttractor::DL2 [protected]

Definition at line 74 of file StrangeAttractor.hpp.

6.150.4.13. double csound::StrangeAttractor::DLW [protected]

Definition at line 75 of file StrangeAttractor.hpp.

6.150.4.14. double csound::StrangeAttractor::DLX [protected]

Definition at line 76 of file StrangeAttractor.hpp.

6.150.4.15. double csound::StrangeAttractor::DLY [protected]

Definition at line 77 of file StrangeAttractor.hpp.

6.150.4.16. double csound::StrangeAttractor::DLZ [protected]

Definition at line 78 of file StrangeAttractor.hpp.

6.150.4.17. double csound::StrangeAttractor::DUM [protected]

Definition at line 79 of file StrangeAttractor.hpp.

6.150.4.18. double csound::StrangeAttractor::duration [protected]

Definition at line 95 of file StrangeAttractor.hpp.

6.150.4.19. double csound::StrangeAttractor::DW [protected]

Definition at line 80 of file StrangeAttractor.hpp.

6.150.4.20. double csound::StrangeAttractor::DX [protected]

Definition at line 81 of file StrangeAttractor.hpp.

6.150.4.21. double csound::StrangeAttractor::DY [protected]

Definition at line 82 of file StrangeAttractor.hpp.

6.150.4.22. double csound::StrangeAttractor::DZ [protected]

Definition at line 83 of file StrangeAttractor.hpp.

6.150.4.23. double csound::StrangeAttractor::EPS [protected]

Definition at line 84 of file StrangeAttractor.hpp.

6.150.4.24. double csound::StrangeAttractor::F [protected]

Definition at line 85 of file StrangeAttractor.hpp.

6.150.4.25. std::string csound::StrangeAttractor::filename [protected]

Definition at line 62 of file StrangeAttractor.hpp.

6.150.4.26. int csound::StrangeAttractor::l [protected]

Definition at line 86 of file StrangeAttractor.hpp.

6.150.4.27. int csound::StrangeAttractor::l1 [protected]

Definition at line 88 of file StrangeAttractor.hpp.

6.150.4.28. int csound::StrangeAttractor::l2 [protected]

Definition at line 89 of file StrangeAttractor.hpp.

6.150.4.29. int csound::StrangeAttractor::l3 [protected]

Definition at line 90 of file StrangeAttractor.hpp.

6.150.4.30. int csound::StrangeAttractor::l4 [protected]

Definition at line 91 of file StrangeAttractor.hpp.

6.150.4.31. int csound::StrangeAttractor::l5 [protected]

Definition at line 92 of file StrangeAttractor.hpp.

6.150.4.32. std::string csound::ScoreNode::importFilename [inherited]

Definition at line 49 of file ScoreNode.hpp.

6.150.4.33. double csound::StrangeAttractor::instrument [protected]

Definition at line 87 of file StrangeAttractor.hpp.

6.150.4.34. int csound::StrangeAttractor::J [protected]

Definition at line 93 of file StrangeAttractor.hpp.

6.150.4.35. double csound::StrangeAttractor::L [protected]

Definition at line 94 of file StrangeAttractor.hpp.

6.150.4.36. ublas::matrix<double> csound::Node::localCoordinates [protected, inherited]

Definition at line 52 of file Node.hpp.

6.150.4.37. double csound::StrangeAttractor::LSUM [protected]

Definition at line 96 of file StrangeAttractor.hpp.

6.150.4.38. int csound::StrangeAttractor::M [protected]

Definition at line 97 of file StrangeAttractor.hpp.

6.150.4.39. double csound::StrangeAttractor::MX [protected]

Definition at line 98 of file StrangeAttractor.hpp.

6.150.4.40. double csound::StrangeAttractor::MY [protected]

Definition at line 99 of file StrangeAttractor.hpp.

6.150.4.41. int csound::StrangeAttractor::N [protected]

Definition at line 100 of file StrangeAttractor.hpp.

6.150.4.42. double csound::StrangeAttractor::N1 [protected]

Definition at line 101 of file StrangeAttractor.hpp.

6.150.4.43. double csound::StrangeAttractor::N2 [protected]

Definition at line 102 of file StrangeAttractor.hpp.

6.150.4.44. double csound::StrangeAttractor::NL [protected]

Definition at line 103 of file StrangeAttractor.hpp.

6.150.4.45. int csound::StrangeAttractor::NMAX [protected]

Definition at line 64 of file StrangeAttractor.hpp.

6.150.4.46. int csound::StrangeAttractor::O [protected]

Definition at line 104 of file StrangeAttractor.hpp.

6.150.4.47. double csound::StrangeAttractor::octave [protected]

Definition at line 105 of file StrangeAttractor.hpp.

6.150.4.48. int csound::StrangeAttractor::ODE [protected]

Definition at line 106 of file StrangeAttractor.hpp.

6.150.4.49. int csound::StrangeAttractor::OMAX [protected]

Definition at line 107 of file StrangeAttractor.hpp.

6.150.4.50. int csound::StrangeAttractor::P [protected]

Definition at line 108 of file StrangeAttractor.hpp.

6.150.4.51. double csound::StrangeAttractor::pitchClassSet [protected]

Definition at line 110 of file StrangeAttractor.hpp.

6.150.4.52. int csound::StrangeAttractor::PREV [protected]

Definition at line 111 of file StrangeAttractor.hpp.

6.150.4.53. double csound::StrangeAttractor::PT [protected]

Definition at line 112 of file StrangeAttractor.hpp.

6.150.4.54. double csound::StrangeAttractor::RAN [protected]

Definition at line 113 of file StrangeAttractor.hpp.

6.150.4.55. Random csound::StrangeAttractor::random [protected]

Definition at line 61 of file StrangeAttractor.hpp.

6.150.4.56. double csound::StrangeAttractor::RS [protected]

Definition at line 114 of file StrangeAttractor.hpp.

6.150.4.57. Score csound::ScoreNode::score [protected, inherited]

Definition at line 47 of file ScoreNode.hpp.

6.150.4.58. int csound::StrangeAttractor::scoreType [protected]

Definition at line 63 of file StrangeAttractor.hpp.

6.150.4.59. double csound::StrangeAttractor::SH [protected]

Definition at line 115 of file StrangeAttractor.hpp.

6.150.4.60. double csound::StrangeAttractor::SINAL [protected]

Definition at line 116 of file StrangeAttractor.hpp.

6.150.4.61. double csound::StrangeAttractor::SW [protected]

Definition at line 118 of file StrangeAttractor.hpp.

6.150.4.62. int csound::StrangeAttractor::T [protected]

Definition at line 119 of file StrangeAttractor.hpp.

6.150.4.63. double csound::StrangeAttractor::TIA [protected]

Definition at line 120 of file StrangeAttractor.hpp.

6.150.4.64. double csound::StrangeAttractor::time [protected]

Definition at line 117 of file StrangeAttractor.hpp.

6.150.4.65. double csound::StrangeAttractor::TT [protected]

Definition at line 121 of file StrangeAttractor.hpp.

6.150.4.66. int csound::StrangeAttractor::TWOD [protected]

Definition at line 122 of file StrangeAttractor.hpp.

6.150.4.67. std::vector<double> csound::StrangeAttractor::V [protected]

Definition at line 124 of file StrangeAttractor.hpp.

6.150.4.68. double csound::StrangeAttractor::W [protected]

Definition at line 125 of file StrangeAttractor.hpp.

6.150.4.69. double csound::StrangeAttractor::WE [protected]

Definition at line 126 of file StrangeAttractor.hpp.

6.150.4.70. double csound::StrangeAttractor::WMAX [protected]

Definition at line 127 of file StrangeAttractor.hpp.

6.150.4.71. double csound::StrangeAttractor::WMIN [protected]

Definition at line 128 of file StrangeAttractor.hpp.

6.150.4.72. double csound::StrangeAttractor::WNEW [protected]

Definition at line 129 of file StrangeAttractor.hpp.

6.150.4.73. double csound::StrangeAttractor::WP [protected]

Definition at line 130 of file StrangeAttractor.hpp.

6.150.4.74. std::vector<double> csound::StrangeAttractor::WS [protected]

Definition at line 131 of file StrangeAttractor.hpp.

6.150.4.75. double csound::StrangeAttractor::WSAVE [protected]

Definition at line 132 of file StrangeAttractor.hpp.

6.150.4.76. double csound::StrangeAttractor::X [protected]

Definition at line 133 of file StrangeAttractor.hpp.

6.150.4.77. double csound::StrangeAttractor::x [protected]

Definition at line 109 of file StrangeAttractor.hpp.

6.150.4.78. double csound::StrangeAttractor::XA [protected]

Definition at line 134 of file StrangeAttractor.hpp.

6.150.4.79. double csound::StrangeAttractor::XE [protected]

Definition at line 135 of file StrangeAttractor.hpp.

6.150.4.80. double csound::StrangeAttractor::XH [protected]

Definition at line 136 of file StrangeAttractor.hpp.

6.150.4.81. double csound::StrangeAttractor::XL [protected]

Definition at line 137 of file StrangeAttractor.hpp.

6.150.4.82. double csound::StrangeAttractor::XMAX [protected]

Definition at line 138 of file StrangeAttractor.hpp.

6.150.4.83. double csound::StrangeAttractor::XMIN [protected]

Definition at line 139 of file StrangeAttractor.hpp.

6.150.4.84. std::vector<double> csound::StrangeAttractor::XN [protected]

Definition at line 140 of file StrangeAttractor.hpp.

6.150.4.85. double csound::StrangeAttractor::XNEW [protected]

Definition at line 141 of file StrangeAttractor.hpp.

6.150.4.86. double csound::StrangeAttractor::XP [protected]

Definition at line 142 of file StrangeAttractor.hpp.

6.150.4.87. std::vector<double> csound::StrangeAttractor::XS [protected]

Definition at line 143 of file StrangeAttractor.hpp.

6.150.4.88. double csound::StrangeAttractor::XSAVE [protected]

Definition at line 144 of file StrangeAttractor.hpp.

6.150.4.89. double csound::StrangeAttractor::XW [protected]

Definition at line 145 of file StrangeAttractor.hpp.

6.150.4.90. std::vector<double> csound::StrangeAttractor::XY [protected]

Definition at line 146 of file StrangeAttractor.hpp.

6.150.4.91. double csound::StrangeAttractor::XZ [protected]

Definition at line 147 of file StrangeAttractor.hpp.

6.150.4.92. double csound::StrangeAttractor::Y [protected]

Definition at line 148 of file StrangeAttractor.hpp.

6.150.4.93. double csound::StrangeAttractor::YA [protected]

Definition at line 149 of file StrangeAttractor.hpp.

6.150.4.94. double csound::StrangeAttractor::YE [protected]

Definition at line 150 of file StrangeAttractor.hpp.

6.150.4.95. double csound::StrangeAttractor::YH [protected]

Definition at line 151 of file StrangeAttractor.hpp.

6.150.4.96. double csound::StrangeAttractor::YL [protected]

Definition at line 152 of file StrangeAttractor.hpp.

6.150.4.97. double csound::StrangeAttractor::YMAX [protected]

Definition at line 153 of file StrangeAttractor.hpp.

6.150.4.98. double csound::StrangeAttractor::YMIN [protected]

Definition at line 154 of file StrangeAttractor.hpp.

6.150.4.99. double csound::StrangeAttractor::YNEW [protected]

Definition at line 155 of file StrangeAttractor.hpp.

6.150.4.100. double csound::StrangeAttractor::YP [protected]

Definition at line 156 of file StrangeAttractor.hpp.

6.150.4.101. std::vector<double> csound::StrangeAttractor::YS [protected]

Definition at line 157 of file StrangeAttractor.hpp.

6.150.4.102. double csound::StrangeAttractor::YSAVE [protected]

Definition at line 158 of file StrangeAttractor.hpp.

6.150.4.103. double csound::StrangeAttractor::YW [protected]

Definition at line 159 of file StrangeAttractor.hpp.

6.150.4.104. double csound::StrangeAttractor::YZ [protected]

Definition at line 160 of file StrangeAttractor.hpp.

6.150.4.105. double csound::StrangeAttractor::Z [protected]

Definition at line 161 of file StrangeAttractor.hpp.

6.150.4.106. double csound::StrangeAttractor::ZA [protected]

Definition at line 162 of file StrangeAttractor.hpp.

6.150.4.107. double csound::StrangeAttractor::ZE [protected]

Definition at line 163 of file StrangeAttractor.hpp.

6.150.4.108. double csound::StrangeAttractor::ZMAX [protected]

Definition at line 164 of file StrangeAttractor.hpp.

6.150.4.109. double csound::StrangeAttractor::ZMIN [protected]

Definition at line 165 of file StrangeAttractor.hpp.

6.150.4.110. double csound::StrangeAttractor::ZNEW [protected]

Definition at line 166 of file StrangeAttractor.hpp.

6.150.4.111. double csound::StrangeAttractor::ZP [protected]

Definition at line 167 of file StrangeAttractor.hpp.

6.150.4.112. std::vector<double> csound::StrangeAttractor::ZS [protected]

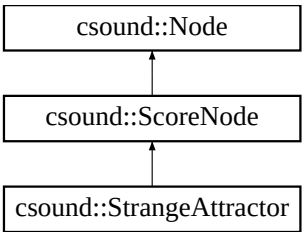
Definition at line 168 of file StrangeAttractor.hpp.

6.150.4.113. double csound::StrangeAttractor::ZSAVE [protected]

Definition at line 169 of file StrangeAttractor.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/StrangeAttractor.hpp


csound::Node
csound::ScoreNode
csound::StrangeAttractor

6.151. Loris::Synthesizer Class Reference

`#include <Synthesizer.h>`

Public Member Functions

- Synthesizer (double srate, std::vector< double > &buffer, double fadeTime=.001)
- void synthesize (const Partial &p)
- void operator() (const Partial &p)
- template<typename Iter> void synthesize (Iter begin_partials, Iter end_partials)
- template<typename Iter> void operator() (Iter begin_partials, Iter end_partials)
- double fadeTime (void) const
- double sampleRate (void) const
- const std::vector< double > samples (void) const
- std::vector< double > samples (void)
- void setFadeTime (double t)

Private Attributes

- Oscillator osc
- std::vector< double > * sampleBuffer
- double tfade
- double srate

6.151.1. Constructor & Destructor Documentation

6.151.1.1. Loris::Synthesizer::Synthesizer (double *srate*, std::vector< double > & *buffer*, double *fadeTime* = .001)

Construct a Synthesizer using the specified sampling rate, sample buffer (a standard library vector), and Partial fade time (in seconds). Since Partials generated by the Loris Analyzer generally begin and end at non-zero amplitude, zero-amplitude Breakpoints are inserted at either end of the Partial, at a temporal distance equal to the fade time, to reduce turn-on and turn-off artifacts. If the fade time is unspecified, the default value of one millisecond (0.001 seconds) is used.

Parameters:

samplerate The rate (Hz) at which to synthesize samples (must be positive).

buffer The vector (of doubles) into which rendered samples should be accumulated.

fade The Partial fade time in seconds (must be non-negative).

Exceptions:

InvalidArgument if the specfied sample rate is non-positive.

InvalidArgument if the specified fade time is negative.

6.151.2. Member Function Documentation

6.151.2.1. double Loris::Synthesizer::fadeTime (void) const

Return this Synthesizer's Partial fade time, in seconds.

6.151.2.2. template<typename Iter> void Loris::Synthesizer::operator() (Iter *begin_partials*, Iter *end_partials*)

Function call operator: same as synthesize(begin_partials, end_partials).

Definition at line 234 of file Synthesizer.h.

6.151.2.3. void Loris::Synthesizer::operator() (const Partial & *p*) [inline]

Function call operator: same as synthesize(p).

Definition at line 106 of file Synthesizer.h.

References synthesize().

6.151.2.4. double Loris::Synthesizer::sampleRate (void) const

Return the sampling rate (in Hz) for this Synthesizer.

6.151.2.5. std::vector<double> Loris::Synthesizer::samples (void)

Return a reference to the sample buffer used (not owned) by this Synthesizer.

6.151.2.6. const std::vector<double> Loris::Synthesizer::samples (void) const

Return a const reference to the sample buffer used (not owned) by this Synthesizer.

6.151.2.7. void Loris::Synthesizer::setFadeTime (double *t*)

Set this Synthesizer's fade time to the specified value (in seconds, must be non-negative).

Parameters:

t The new Partial fade time.

Exceptions:

InvalidArgument if the specified fade time is negative.

6.151.2.8. template<typename Iter> void Loris::Synthesizer::synthesize (Iter *begin_partials*, Iter *end_partials*)

Synthesize all Partials on the specified half-open (STL-style) range. Null Breakpoints are inserted at either end of the Partial to reduce turn-on and turn-off artifacts, as described above. The synthesizer will resize the buffer as necessary to accommodate all the samples, including the fade outs. Previous contents of the buffer are not overwritten. Partials with start times earlier than the Partial fade time will have shorter onset fades. Partials are not rendered at frequencies above the half-sample rate.

Parameters:

begin_partials The beginning of the range of Partials to synthesize.

end_partials The end of the range of Partials to synthesize.

Returns:

Nothing.

Precondition:
The partials must have non-negative start times.

Postcondition:
This Synthesizer's sample buffer (vector) has been resized to accommodate the entire duration of all the Partials including fade out at the ends.

Exceptions:
InvalidPartial if any Partial has negative start time.

Definition at line 207 of file Synthesizer.h.

6.151.2.9. void Loris::Synthesizer::synthesize (const Partial & *p*)

Synthesize a bandwidth-enhanced sinusoidal Partial. Zero-amplitude Breakpoints are inserted at either end of the Partial to reduce turn-on and turn-off artifacts, as described above. The synthesizer will resize the buffer as necessary to accommodate all the samples, including the fade out. Previous contents of the buffer are not overwritten. Partials with start times earlier than the Partial fade time will have shorter onset fades. Partials are not rendered at frequencies above the half-sample rate.

Parameters:
p The Partial to synthesize.

Returns:
Nothing.

Precondition:
The partial must have non-negative start time.

Postcondition:
This Synthesizer's sample buffer (vector) has been resized to accommodate the entire duration of the Partial, p, including fade out at the end.

Exceptions:
InvalidPartial if the Partial has negative start time.

Referenced by operator()().

6.151.3. Member Data Documentation

6.151.3.1. Oscillator Loris::Synthesizer::osc [private]

Definition at line 174 of file Synthesizer.h.

6.151.3.2. std::vector< double >* Loris::Synthesizer::sampleBuffer [private]

Definition at line 175 of file Synthesizer.h.

6.151.3.3. double Loris::Synthesizer::srate [private]

Definition at line 177 of file Synthesizer.h.

6.151.3.4. double Loris::Synthesizer::tfade [private]

Definition at line 176 of file Synthesizer.h.

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/Synthesizer.h

6.152. Synthesizer Class Reference

`#include <synthesizer.hpp>`

Public Member Functions

- Synthesizer ()
- ~Synthesizer ()
- void set_yield_callback (void *, yield_callback_t)
- void set_message_callback (void *, message_callback_t)
- void set_throw_message_callback (void *, message_callback_t)
- void set_draw_graph_callback (void *, draw_graph_callback_t)
- int perform (int, char **)
- int perform (std::vector< std::string >)

Static Public Member Functions

- static std::string get_version ()
- static void s_draw_graph (void *, Curve *)

Private Member Functions

- int yield ()
- void message (const char *, va_list)
- void throw_message (const char *, va_list)
- void draw_graph (Curve *)

Static Private Member Functions

- static int s_yield (void *)
- static void s_message (void *, const char *, va_list)
- static void s_throw_message (void *, const char *, va_list)

Private Attributes

- void * m_csound
- void * m_yield_data
- yield_callback_t m_yield_callback
- void * m_message_data
- message_callback_t m_message_callback
- void * m_throw_message_data
- message_callback_t m_throw_message_callback
- void * m_draw_graph_data
- draw_graph_callback_t m_draw_graph_callback

6.152.1. Constructor & Destructor Documentation

6.152.1.1. Synthesizer::Synthesizer ()

6.152.1.2. Synthesizer::~Synthesizer ()

6.152.2. Member Function Documentation

6.152.2.1. void Synthesizer::draw_graph (Curve *) [private]

6.152.2.2. static std::string Synthesizer::get_version () [static]

6.152.2.3. void Synthesizer::message (const char *, va_list) [private]

6.152.2.4. int Synthesizer::perform (std::vector< std::string >)

6.152.2.5. int Synthesizer::perform (int, char **)

6.152.2.6. static void Synthesizer::s_draw_graph (void *, Curve *) [static]

6.152.2.7. static void Synthesizer::s_message (void *, const char *, va_list) [static, private]

6.152.2.8. static void Synthesizer::s_throw_message (void *, const char *, va_list) [static, private]

6.152.2.9. static int Synthesizer::s_yield (void *) [static, private]

6.152.2.10. void Synthesizer::set_draw_graph_callback (void *, draw_graph_callback_t)

6.152.2.11. void Synthesizer::set_message_callback (void *, message_callback_t)

6.152.2.12. void Synthesizer::set_throw_message_callback (void *, message_callback_t)

6.152.2.13. void Synthesizer::set_yield_callback (void *, yield_callback_t)

6.152.2.14. void Synthesizer::throw_message (const char *, va_list) [private]

6.152.2.15. int Synthesizer::yield () [private]

6.152.3. Member Data Documentation

6.152.3.1. void* Synthesizer::m_csound [private]

Definition at line 44 of file synthesizer.hpp.

6.152.3.2. draw_graph_callback_t Synthesizer::m_draw_graph_callback [private]

Definition at line 58 of file synthesizer.hpp.

6.152.3.3. void* Synthesizer::m_draw_graph_data [private]

Definition at line 57 of file synthesizer.hpp.

6.152.3.4. message_callback_t Synthesizer::m_message_callback [private]

Definition at line 50 of file synthesizer.hpp.

6.152.3.5. void* Synthesizer::m_message_data [private]

Definition at line 49 of file synthesizer.hpp.

6.152.3.6. message_callback_t Synthesizer::m_throw_message_callback [private]

Definition at line 54 of file synthesizer.hpp.

6.152.3.7. void* Synthesizer::m_throw_message_data [private]

Definition at line 53 of file synthesizer.hpp.

6.152.3.8. yield_callback_t Synthesizer::m_yield_callback [private]

Definition at line 46 of file synthesizer.hpp.

6.152.3.9. void* Synthesizer::m_yield_data [private]

Definition at line 45 of file synthesizer.hpp.

The documentation for this class was generated from the following file:

- frontends/flcsound/synthesizer.hpp

6.153. csound::System Class Reference

`#include <System.hpp>`

6.153.1. Detailed Description

Abstraction layer for a minimal set of system services.

Definition at line 55 of file System.hpp.

Public Types

- enum Level { ERROR_LEVEL = 1, WARNING_LEVEL = 2, INFORMATION_LEVEL = 4, DEBUGGING_LEVEL = 8 }

Static Public Member Functions

- static void parsePathname (const std::string pathname, std::string &drive, std::string &base, std::string &file, std::string &extension)
- static void * openLibrary (std::string filename)
- static void * getSymbol (void *library, std::string name)
- static void closeLibrary (void *library)
- static std::vector< std::string > getFilenames (std::string directoryName)
- static std::vector< std::string > getDirectoryNames (std::string directoryName)
- static void * createThread (void(*threadRoutine)(void *threadData), void *data, int priority)
- static void * createThreadLock ()
- static void waitThreadLock (void *lock, size_t timeoutMilliseconds=0)
- static void notifyThreadLock (void *lock)
- static void destroyThreadLock (void *lock)
- static int setMessageLevel (int messageLevel)
- static void yieldThread ()
- static int getMessageLevel ()
- static void setUserdata (void *userdata)
- static void * getUserdata ()
- static void error (void *userdata, const char *format,...)
- static void error (const char *format,...)
- static void warn (void *userdata, const char *format,...)
- static void warn (const char *format,...)
- static void inform (void *userdata, const char *format,...)
- static void inform (const char *format,...)
- static void debug (void *userdata, const char *format,...)
- static void debug (const char *format,...)
- static void message (void *userdata, const char *format,...)
- static void message (const char *format,...)
- static void message (void *userdata, const char *format, va_list valist)
- static void message (const char *format, va_list valist)
- static void message (void *userdata, int level, const char *format,...)
- static void message (void *userdata, int attribute, const char *format, va_list valist)
- static void setMessageCallback (void(*messageCallback_)(void *userdata, int attribute, const char *format, va_list marker))
- static MessageCallbackType getMessageCallback ()
- static int execute (const char *command)

- static int shellOpen (const char *filename, const char *command="open")
- static std::string getSharedLibraryExtension ()
- static clock_t startTiming ()
- static double stopTiming (clock_t startedAt)
- static void sleep (double milliseconds)
- static void beep ()

Static Private Attributes

- static void * userdata_
- static int messageLevel
- static void(* messageCallback)(void *userdata, int attribute, const char *format, va_list valist)

6.153.2. Member Enumeration Documentation

6.153.2.1. enum csound::System::Level

Enumeration values:

ERROR_LEVEL

WARNING_LEVEL

INFORMATION_LEVEL

DEBUGGING_LEVEL

Definition at line 61 of file System.hpp.

6.153.3. Member Function Documentation

6.153.3.1. static void csound::System::beep () [static]

Make some sort of noticeable sound.

6.153.3.2. static void csound::System::closeLibrary (void * *library*) [static]

Closes a shared library.

6.153.3.3. static void* csound::System::createThread (void(*)(void *threadData) *threadRoutine*, void * *data*, int *priority*) [static]

Creates a new thread.

6.153.3.4. static void* csound::System::createThreadLock () [static]

Creates a thread lock.

6.153.3.5. static void csound::System::debug (const char * *format*, ...) [static]

Prints a message if the DEBUGGING_LEVEL flag is set.

6.153.3.6. static void csound::System::debug (void * *userdata*, const char * *format*, ...) [static]

Prints a message if the DEBUGGING_LEVEL flag is set.

6.153.3.7. static void csound::System::destroyThreadLock (void * *lock*) [static]

Destroys a thread lock.

6.153.3.8. static void csound::System::error (const char * *format*, ...) [static]

Prints a message if the ERROR_LEVEL flag is set.

6.153.3.9. static void csound::System::error (void * *userdata*, const char * *format*, ...) [static]

Prints a message if the ERROR_LEVEL flag is set.

6.153.3.10. static int csound::System::execute (const char * *command*) [static]

Execute a system command or program.

6.153.3.11. static std::vector<std::string> csound::System::getDirectoryNames (std::string *directoryName*) [static]

Lists directory names in a directory; useful for locating plugins.

6.153.3.12. static std::vector<std::string> csound::System::getFilenames (std::string *directoryName*) [static]

Lists filenames in a directory; useful for locating plugins.

6.153.3.13. static MessageCallbackType csound::System::getMessageCallback () [static]

Return the message callback, or null if none.

6.153.3.14. static int csound::System::getMessageLevel () [static]

Returns current system message level.

6.153.3.15. static std::string csound::System::getSharedLibraryExtension () [static]

Returns the standard filename extension for a shared library, such as "dll" or "so".

6.153.3.16. static void* csound::System::getSymbol (void * *library*, std::string *name*) [static]

Returns the address of a symbol (function or object) in a shared library; useful for loading plugin functions.

6.153.3.17. static void* csound::System::getUserdata () [static]

Returns userdata for message printing.

6.153.3.18. static void csound::System::inform (const char * *format*, ...) [static]

Prints a message if the INFORMATION_LEVEL flag is set.

6.153.3.19. static void csound::System::inform (void * *userdata*, const char * *format*, ...) [static]

Prints a message if the INFORMATION_LEVEL flag is set.

6.153.3.20. static void csound::System::message (void * *userdata*, int *attribute*, const char * *format*, va_list *valist*) [static]

Prints a message.

6.153.3.21. static void csound::System::message (void * *userdata*, int *level*, const char * *format*, ...) [static]

Prints a message.

6.153.3.22. static void csound::System::message (const char * *format*, va_list *valist*) [static]

Prints a message.

6.153.3.23. static void csound::System::message (void * *userdata*, const char * *format*, va_list *valist*) [static]

Prints a message.

6.153.3.24. static void csound::System::message (const char * *format*, ...) [static]

Prints a message.

6.153.3.25. static void csound::System::message (void * *userdata*, const char * *format*, ...) [static]

Prints a message.

6.153.3.26. static void csound::System::notifyThreadLock (void * *lock*) [static]

Releases a thread lock.

6.153.3.27. static void* csound::System::openLibrary (std::string *filename*) [static]

Opens a shared library; useful for loading plugins.

6.153.3.28. static void csound::System::parsePathname (const std::string *pathname*, std::string & *drive*, std::string & *base*, std::string & *file*, std::string & *extension*) [static]

Parses a filename into its component parts, which are returned in the arguments. On Unix and Linux, "drive" is always empty.

6.153.3.29. static void csound::System::setMessageCallback (void(*)(void *userdata, int attribute, const char *format, va_list marker) *messageCallback_*) [static]

Sets message callback.

6.153.3.30. static int csound::System::setMessageLevel (int *messageLevel*) [static]

Sets message level, returns old message level.

6.153.3.31. static void csound::System::setUserdata (void * *userdata*) [static]

Sets userdata for message printing.

6.153.3.32. static int csound::System::shellOpen (const char * *filename*, const char * *command* = "open") [static]

Open a file using the operating system shell.

6.153.3.33. static void csound::System::sleep (double *milliseconds*) [static]

Sleep the indicated number of milliseconds.

6.153.3.34. static clock_t csound::System::startTiming () [static]

Starts timing.

6.153.3.35. static double csound::System::stopTiming (clock_t *startedAt*) [static]

Stop timing, and return elapsed seonds.

6.153.3.36. static void csound::System::waitThreadLock (void * *lock*, size_t *timeoutMilliseconds* = 0) [static]

Waits on a thread lock. Zero timeout means infinite timeout.

6.153.3.37. static void csound::System::warn (const char * *format*, ...) [static]

Prints a message if the WARNNING_LEVEL flag is set.

6.153.3.38. static void csound::System::warn (void * *userdata*, const char * *format*, ...) [static]

Prints a message if the WARNNING_LEVEL flag is set.

6.153.3.39. static void csound::System::yieldThread () [static]

Yields to the next waiting thread.

6.153.4. Member Data Documentation

6.153.4.1. void(* csound::System::messageCallback)(void *userdata, int attribute, const char *format, va_list valist) [static, private]

6.153.4.2. int csound::System::messageLevel [static, private]

Definition at line 58 of file System.hpp.

6.153.4.3. void* csound::System::userdata_ [static, private]

Definition at line 57 of file System.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/System.hpp

6.154. TEMPO Struct Reference

`#include <csoundCore.h>`

Public Attributes

- OPDS h
- MYFLT * ktempo
- MYFLT * istartempo
- MYFLT prvtempo

6.154.1. Member Data Documentation

6.154.1.1. OPDS TEMPO::h

Definition at line 431 of file csoundCore.h.

6.154.1.2. MYFLT * TEMPO::istartempo

Definition at line 432 of file csoundCore.h.

6.154.1.3. MYFLT* TEMPO::ktempo

Definition at line 432 of file csoundCore.h.

6.154.1.4. MYFLT TEMPO::prvtempo

Definition at line 433 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.155. csound::TempoMap Class Reference

`#include <Midifile.hpp>`

Public Member Functions

- double getCurrentSecondsPerTick (int tick)

6.155.1. Member Function Documentation

6.155.1.1. double csound::TempoMap::getCurrentSecondsPerTick (int *tick*)

The documentation for this class was generated from the following file:

- frontends/CsoundVST/Midifile.hpp

6.156. text Struct Reference

`#include <csoundCore.h>`

Public Attributes

- int linenum
- int opnum
- char * opcod
- ARGLST * inlist
- ARGLST * outlist
- ARGOFFS * inoffs
- ARGOFFS * outoffs
- int xincod
- int xoutcod
- int xincod_str
- int xoutcod_str
- char intype
- char pftype

6.156.1. Member Data Documentation

6.156.1.1. ARGLST* text::inlist

Definition at line 169 of file csoundCore.h.

6.156.1.2. ARGOFFS* text::inoffs

Definition at line 171 of file csoundCore.h.

6.156.1.3. char text::intype

Definition at line 177 of file csoundCore.h.

6.156.1.4. int text::linenum

Definition at line 166 of file csoundCore.h.

6.156.1.5. char* text::opcod

Definition at line 168 of file csoundCore.h.

6.156.1.6. int text::opnum

Definition at line 167 of file csoundCore.h.

6.156.1.7. ARGLST* text::outlist

Definition at line 170 of file csoundCore.h.

6.156.1.8. ARGOFFS* text::outoffs

Definition at line 172 of file csoundCore.h.

6.156.1.9. char text::pftype

Definition at line 178 of file csoundCore.h.

6.156.1.10. int text::xincod

Definition at line 173 of file csoundCore.h.

6.156.1.11. int text::xincod_str

Definition at line 175 of file csoundCore.h.

6.156.1.12. int text::xoutcod

Definition at line 174 of file csoundCore.h.

6.156.1.13. int text::xoutcod_str

Definition at line 176 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.157. csound::ThreadLock Class Reference

`#include <System.hpp>`

6.157.1. Detailed Description

Encapsulates a thread monitor, such as a Windows event handle.

Definition at line 237 of file System.hpp.

Public Member Functions

- ThreadLock ()
- virtual ~ThreadLock ()
- virtual void open ()
- virtual void close ()
- virtual bool isOpen ()
- virtual void startWait (size_t timeoutMilliseconds=0)
- virtual void endWait ()

Private Attributes

- void * lock

6.157.2. Constructor & Destructor Documentation

6.157.2.1. csound::ThreadLock::ThreadLock ()

6.157.2.2. virtual csound::ThreadLock::~ThreadLock () [virtual]

6.157.3. Member Function Documentation

6.157.3.1. virtual void csound::ThreadLock::close () [virtual]

Destroys the monitor.

6.157.3.2. virtual void csound::ThreadLock::endWait () [virtual]

Releases one thread that is waiting on the monitor.

6.157.3.3. virtual bool csound::ThreadLock::isOpen () [virtual]

Returns whether the monitor is open.

6.157.3.4. virtual void csound::ThreadLock::open () [virtual]

Creates and initializes the monitor. The monitor is in a non-notified or unsignaled state.

6.157.3.5. virtual void csound::ThreadLock::startWait (size_t *timeoutMilliseconds* = 0) [virtual]

Waits until the monitor is notified by another thread. Zero timeout means infinite timeout.

6.157.4. Member Data Documentation

6.157.4.1. void* csound::ThreadLock::lock [private]

Definition at line 239 of file System.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/System.hpp

6.158. Loris::PartialUtils::TimeShifter Class Reference

`#include <PartialUtils.h>`

Public Member Functions

- TimeShifter (double x)
- void operator() (Partial &p) const

Private Attributes

- double offset

6.158.1. Constructor & Destructor Documentation

6.158.1.1. Loris::PartialUtils::TimeShifter::TimeShifter (double *x*) [inline]

Definition at line 363 of file PartialUtils.h.

References offset.

6.158.2. Member Function Documentation

6.158.2.1. void Loris::PartialUtils::TimeShifter::operator() (Partial & *p*) const

6.158.3. Member Data Documentation

6.158.3.1. double Loris::PartialUtils::TimeShifter::offset [private]

Definition at line 369 of file PartialUtils.h.

Referenced by TimeShifter().

The documentation for this class was generated from the following file:

- Opcodes/Loris/src/PartialUtils.h

6.159. token Struct Reference

`#include <csoundCore.h>`

Public Attributes

- char * str
- short prec

6.159.1. Member Data Documentation

6.159.1.1. short token::prec

Definition at line 495 of file csoundCore.h.

6.159.1.2. char* token::str

Definition at line 494 of file csoundCore.h.

The documentation for this struct was generated from the following file:

- H/csoundCore.h

6.160. WaitCursor Class Reference

`#include <CsoundVstFltk.hpp>`

6.160.1. Detailed Description

C S O U N D V S T

A VST plugin version of Csound, with Python scripting.

L I C E N S E

This software is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Definition at line 51 of file CsoundVstFltk.hpp.

Public Member Functions

- WaitCursor ()
- virtual ~WaitCursor ()

Private Attributes

- void * cursor

6.160.2. Constructor & Destructor Documentation

6.160.2.1. WaitCursor::WaitCursor ()

6.160.2.2. virtual WaitCursor::~WaitCursor () [virtual]

6.160.3. Member Data Documentation

6.160.3.1. void* WaitCursor::cursor [private]

Definition at line 53 of file CsoundVstFltk.hpp.

The documentation for this class was generated from the following file:

- frontends/CsoundVST/CsoundVstFltk.hpp

6.161. ZFILTER Struct Reference

`#include <filter.h>`

Public Attributes

- OPDS h
- MYFLT * out
- MYFLT * in
- MYFLT * kmagf
- MYFLT * kphsf
- MYFLT * nb
- MYFLT * na
- MYFLT * coeffs [MAXPOLES+MAXZEROS+1]
- int numa
- int numb
- double dcoeffs [MAXPOLES+MAXZEROS+1]
- AUXCH delay
- double * currPos
- int ndelay
- AUXCH roots

6.161.1. Member Data Documentation

6.161.1.1. MYFLT* ZFILTER::coeffs[MAXPOLES+MAXZEROS+1]

Definition at line 65 of file filter.h.

6.161.1.2. double* ZFILTER::currPos

Definition at line 72 of file filter.h.

6.161.1.3. double ZFILTER::dcoeffs[MAXPOLES+MAXZEROS+1]

Definition at line 70 of file filter.h.

6.161.1.4. AUXCH ZFILTER::delay

Definition at line 71 of file filter.h.

6.161.1.5. OPDS ZFILTER::h

Definition at line 59 of file filter.h.

6.161.1.6. MYFLT* ZFILTER::in

Definition at line 62 of file filter.h.

6.161.1.7. MYFLT* ZFILTER::kmagf

Definition at line 63 of file filter.h.

6.161.1.8. MYFLT * ZFILTER::kphsf

Definition at line 63 of file filter.h.

6.161.1.9. MYFLT * ZFILTER::na

Definition at line 64 of file filter.h.

6.161.1.10. MYFLT* ZFILTER::nb

Definition at line 64 of file filter.h.

6.161.1.11. int ZFILTER::ndelay

Definition at line 73 of file filter.h.

6.161.1.12. int ZFILTER::numa

Definition at line 67 of file filter.h.

6.161.1.13. int ZFILTER::numb

Definition at line 68 of file filter.h.

6.161.1.14. MYFLT* ZFILTER::out

Definition at line 61 of file filter.h.

6.161.1.15. AUXCH ZFILTER::roots

Definition at line 74 of file filter.h.

The documentation for this struct was generated from the following file:

- Opcodes/filter.h

7. Csound and CsoundVST File Documentation

7.1. frontends/CsoundVST/Cell.hpp File Reference

```
#include "ScoreNode.hpp"
```

Namespaces

- namespace csound
- namespace boost::numeric

7.2. frontends/CsoundVST/Composition.hpp File Reference

```
#include "CppSound.hpp"
#include "Score.hpp"
```

Namespaces

- namespace csound

7.3. frontends/CsoundVST/Conversions.hpp File Reference

```
#include <cmath>
#include <string>
#include <cstdio>
#include <map>
```

Namespaces

- namespace csound

7.4. frontends/CsoundVST/Counterpoint.hpp File Reference

```
#include <string>
#include <cstdarg>
#include <stdio.h>
#include <malloc.h>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/random.hpp>
#include <boost/random/variate_generator.hpp>
#include "Random.hpp"
```

7.5. frontends/CsoundVST/CounterpointNode.hpp File Reference

```
#include "Node.hpp"
#include "Counterpoint.hpp"
#include <cmath>
```

Namespaces

- namespace csound

7.6. frontends/CsoundVST/CppSound.hpp File Reference

```
#include "CsoundFile.hpp"
#include <string>
#include <vector>
#include <csound.h>
#include <csoundCore.h>
```

7.7. frontends/CsoundVST/CsoundFile.hpp File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <fstream>
#include <sstream>
#include <stdlib.h>
```

Functions

- void gatherArgs (int argc, const char **argv, std::string &commandLine)
- void scatterArgs (const std::string commandLine, int *argc, char ***argv)
- void deleteArgs (int argc, char **argv)
- std::string & trim (std::string &value)
- std::string & trimQuotes (std::string &value)
- bool parseInstrument (const std::string &definition, std::string &preNumber, std::string &id, std::string &name, std::string &postNumber)

7.7.1. Function Documentation

7.7.1.1. void deleteArgs (int *argc*, char ** *argv*) [inline]

Definition at line 83 of file CsoundFile.hpp.

7.7.1.2. void gatherArgs (int *argc*, const char ** *argv*, std::string & *commandLine*) [inline]

C S O U N D V S T

A VST plugin version of Csound, with Python scripting.

L I C E N S E

This software is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Definition at line 49 of file CsoundFile.hpp.

7.7.1.3. bool parseInstrument (const std::string & *definition*, std::string & *preNumber*, std::string & *id*, std::string & *name*, std::string & *postNumber*)

Returns true if definition is a valid Csound instrument definition block. Also returns the part before the instr number, the instr number, the name (all text after the first comment on the same

line as the instr number), and the part after the instr number, all by reference.

7.7.1.4. void scatterArgs (const std::string *commandLine*, int * *argc*, char *** *argv*) [inline]

Definition at line 65 of file CsoundFile.hpp.

7.7.1.5. std::string& trim (std::string & *value*) [inline]

Definition at line 99 of file CsoundFile.hpp.

7.7.1.6. std::string& trimQuotes (std::string & *value*) [inline]

Definition at line 119 of file CsoundFile.hpp.

7.8. frontends/CsoundVST/CsoundVST.hpp File Reference

```
#include "audioeffectx.h"
#include "CppSound.hpp"
#include "Shell.hpp"
#include <list>
```

Functions

- PUBLIC CsoundVST * CreateCsoundVST ()

7.8.1. Function Documentation

7.8.1.1. PUBLIC CsoundVST* CreateCsoundVST ()

7.9. frontends/CsoundVST/csoundvst_api.h File Reference

Defines

- #define PUBLIC
- #define CSVST_CLASSIC_MODE 0
- #define CSVST_PYTHON_MODE 1

Functions

- PUBLIC void * csvstCreate ()
- PUBLIC void csvstDestroy (void *csvst)
- PUBLIC void csvstSetMode (void *csvst, int mode)
- PUBLIC int csvstGetMode (void *csvst)
- PUBLIC void csvstLoad (void *csvst, const char *filename)
- PUBLIC void csvstImport (void *csvst, const char *filename)
- PUBLIC void csvstSave (void *csvst, const char *filename)
- PUBLIC void csvstExport (void *csvst, const char *filename)
- PUBLIC void csvstPerform (void *csvst)
- PUBLIC void csvstInputScoreLine (void *csvst, const char *scoreline)
- PUBLIC void csvstStopPerforming (void *csvst)
- PUBLIC void csvstOpenWindow (void *csvst)
- PUBLIC void csvstCloseWindow (void *csvst)
- PUBLIC void csvstClearScript (void *csvst)
- PUBLIC void csvstClearCsd (void *csvst)
- PUBLIC void csvstClearCommand (void *csvst)
- PUBLIC void csvstClearOrchestra (void *csvst)
- PUBLIC void csvstClearArrangement (void *csvst)
- PUBLIC void csvstClearScore (void *csvst)
- PUBLIC const char * csvstGetScript (void *csvst)
- PUBLIC const char * csvstGetCsd (void *csvst)
- PUBLIC const char * csvstGetCommand (void *csvst)
- PUBLIC const char * csvstGetOrchestra (void *csvst)
- PUBLIC int csvstGetArrangementCount (void *csvst)
- PUBLIC const char * csvstGetArrangement (void *csvst, int index)
- PUBLIC const char * csvstGetScore (void *csvst)
- PUBLIC void csvstSetScript (void *csvst, const char *data)
- PUBLIC void csvstSetCsd (void *csvst, const char *data)
- PUBLIC void csvstSetCommand (void *csvst, const char *data)
- PUBLIC void csvstSetOrchestra (void *csvst, const char *data)
- PUBLIC void csvstAddArrangement (void *csvst, const char *instrumentName)
- PUBLIC void csvstSetScore (void *csvst, const char *data)
- PUBLIC void csvstAddScoreLine (void *csvst, const char *scoreline)
- PUBLIC void csvstAddNote3 (void *csvst, double p1_instrument, double p2_time, double p3_duration)
- PUBLIC void csvstAddNote4 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key)
- PUBLIC void csvstAddNote5 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity)
- PUBLIC void csvstAddNote6 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity, double p6_phase)

- PUBLIC void csvstAddNote7 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity, double p6_phase, double p7_x)

- PUBLIC void csvstAddNote8 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity, double p6_phase, double p7_x, double p8_y)

- PUBLIC void csvstAddNote9 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity, double p6_phase, double p7_x, double p8_y, double p9_z)

- PUBLIC void csvstAddNote10 (void *csvst, double p1_instrument, double p2_time, double p3_duration, double p4_key, double p5_velocity, double p6_phase, double p7_x, double p8_y, double p9_z, double p10_pitchClassSet)

7.9.1. Define Documentation

7.9.1.1. #define CSVST_CLASSIC_MODE 0

This is a high-level "C" API for CsoundVST, for Mathematica and other hosts.

Definition at line 40 of file csoundvst_api.h.

7.9.1.2. #define CSVST_PYTHON_MODE 1

Definition at line 41 of file csoundvst_api.h.

7.9.1.3. #define PUBLIC

Definition at line 28 of file csoundvst_api.h.

7.9.2. Function Documentation

7.9.2.1. PUBLIC void csvstAddArrangement (void * *csvst*, const char * *instrumentName*)

7.9.2.2. PUBLIC void csvstAddNote10 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*, double *p6_phase*, double *p7_x*, double *p8_y*, double *p9_z*, double *p10_pitchClassSet*)

7.9.2.3. PUBLIC void csvstAddNote3 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*)

7.9.2.4. PUBLIC void csvstAddNote4 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*)

7.9.2.5. PUBLIC void csvstAddNote5 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*)

7.9.2.6. PUBLIC void csvstAddNote6 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*, double *p6_phase*)

7.9.2.7. PUBLIC void csvstAddNote7 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*, double *p6_phase*, double *p7_x*)

7.9.2.8. PUBLIC void csvstAddNote8 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*, double *p6_phase*, double *p7_x*, double *p8_y*)

7.9.2.9. PUBLIC void csvstAddNote9 (void * *csvst*, double *p1_instrument*, double *p2_time*, double *p3_duration*, double *p4_key*, double *p5_velocity*, double *p6_phase*, double *p7_x*, double *p8_y*, double *p9_z*)

7.9.2.10. PUBLIC void csvstAddScoreLine (void * *csvst*, const char * *scoreline*)

7.9.2.11. PUBLIC void csvstClearArrangement (void * *csvst*)

7.9.2.12. PUBLIC void csvstClearCommand (void * *csvst*)

7.9.2.13. PUBLIC void csvstClearCsd (void * *csvst*)

7.9.2.14. PUBLIC void csvstClearOrchestra (void * *csvst*)

7.9.2.15. PUBLIC void csvstClearScore (void * *csvst*)

7.9.2.16. PUBLIC void csvstClearScript (void * *csvst*)

7.9.2.17. PUBLIC void csvstCloseWindow (void * *csvst*)

7.9.2.18. PUBLIC void* csvstCreate ()

7.9.2.19. PUBLIC void csvstDestroy (void * *csvst*)

7.9.2.20. PUBLIC void csvstExport (void * *csvst*, const char * *filename*)

7.9.2.21. PUBLIC const char* csvstGetArrangement (void * *csvst*, int *index*)

7.9.2.22. PUBLIC int csvstGetArrangementCount (void * *csvst*)

7.9.2.23. PUBLIC const char* csvstGetCommand (void * *csvst*)

7.9.2.24. PUBLIC const char* csvstGetCsd (void * *csvst*)

7.10. frontends/CsoundVST/CsoundVstFltk.hpp File Reference

```
#include <AEffEditor.hpp>
#include <FL/Fl_Help_View.H>
#include <FL/Fl_Pack.H>
#include <FL/Fl_Tabs.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Preferences.H>
#include <FL/Fl_Browser.H>
#include <FL/Fl_Text_Editor.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Text_Buffer.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Check_Button.H>
#include <FL/Fl_Group.H>
#include <list>
#include "CsoundVST.hpp"
#include "CsoundVstUi.h"
```

7.11. frontends/CsoundVST/Event.hpp File Reference

```
#include "Conversions.hpp"
#include "Midifile.hpp"
#include <map>
#include <string>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <utility>
#include <boost/numeric/ublas/vector.hpp>
```

Namespaces

- namespace csound

Functions

- bool operator< (const Event &a, const Event &b)

7.11.1. Function Documentation

7.11.1.1. bool operator< (const Event & *a*, const Event & *b*)

7.12. frontends/CsoundVST/Exception.hpp File Reference

```
#include <string>
```

Namespaces

- namespace csound

7.13. frontends/CsoundVST/Hocket.hpp File Reference

```
#include "ScoreNode.hpp"
```

Namespaces

- namespace csound

7.14. frontends/CsoundVST/ImageToScore.hpp File Reference

```
#include "Silence.hpp"
```

Namespaces

- namespace csound

7.15. frontends/CsoundVST/Lindenmayer.hpp File Reference

```
#include "Silence.hpp"
#include <stack>
#include <string>
#include <map>
#include <vector>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
```

Namespaces

- namespace csound

7.16. frontends/CsoundVST/MCRM.hpp File Reference

```
#include "Silence.hpp"
```

Namespaces

- namespace csound

7.17. frontends/CsoundVST/Midifile.hpp File Reference

```
#include <algorithm>
#include <utility>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <vector>
```

Namespaces

- namespace csound

Typedefs

- typedef unsigned char csound_u_char

Functions

- bool operator< (const MidiEvent &a, MidiEvent &b)

7.17.1. Typedef Documentation

7.17.1.1. typedef unsigned char csound::csound_u_char

Definition at line 50 of file Midifile.hpp.

7.17.2. Function Documentation

7.17.2.1. bool operator< (const MidiEvent & *a*, MidiEvent & *b*)

7.18. frontends/CsoundVST/MusicModel.hpp File Reference

```
#include "Composition.hpp"
#include "Node.hpp"
#include "Score.hpp"
```

Namespaces

- namespace csound

7.19. frontends/CsoundVST/Node.hpp File Reference

```
#include "Score.hpp"
#include <vector>
#include <boost/numeric/ublas/matrix.hpp>
```

Namespaces

- namespace csound

Typedefs

- typedef Node * NodePtr

7.19.1. Typedef Documentation

7.19.1.1. typedef Node* csound::NodePtr

Definition at line 84 of file Node.hpp.

7.20. frontends/CsoundVST/Random.hpp File Reference

```
#include "Node.hpp"
#include <boost/random.hpp>
#include <boost/random/variate_generator.hpp>
#include <cmath>
```

Namespaces

- namespace csound

7.21. frontends/CsoundVST/Rescale.hpp File Reference

```
#include "ScoreNode.hpp"
```

Namespaces

- namespace csound

7.22. frontends/CsoundVST/Score.hpp File Reference

```
#include "Event.hpp"
#include "Midifile.hpp"
#include <iostream>
#include <vector>
```

Namespaces

- namespace csound

7.23. frontends/CsoundVST/ScoreNode.hpp File Reference

```
#include "Node.hpp"
#include "Score.hpp"
```

Namespaces

- namespace csound

7.24. frontends/CsoundVST/Sequence.hpp File Reference

```
#include "ScoreNode.hpp"
```

Namespaces

- namespace csound

7.25. frontends/CsoundVST/Shell.hpp File Reference

```
#include <string>
```

Namespaces

- namespace csound

7.26. frontends/CsoundVST/Silence.hpp File Reference

```
#include <string>
#include <vector>
#include <map>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include "Conversions.hpp"
#include "System.hpp"
#include "CsoundFile.hpp"
#include "CppSound.hpp"
#include "Event.hpp"
#include "Midifile.hpp"
#include "Score.hpp"
#include "Composition.hpp"
#include "Node.hpp"
#include "Counterpoint.hpp"
#include "CounterpointNode.hpp"
#include "ScoreNode.hpp"
#include "Cell.hpp"
#include "Hocket.hpp"
#include "Rescale.hpp"
#include "MusicModel.hpp"
#include "Sequence.hpp"
#include "Random.hpp"
#include "ImageToScore.hpp"
#include "StrangeAttractor.hpp"
#include "Lindenmayer.hpp"
#include "MCRM.hpp"
```

7.27. frontends/CsoundVST/StrangeAttractor.hpp File Reference

```
#include "Silence.hpp"
#include <string>
#include <vector>
#include <boost/random.hpp>
#include <boost/numeric/ublas/matrix.hpp>
```

Namespaces

- namespace csound

7.28. frontends/CsoundVST/System.hpp File Reference

```
#include <string>
#include <vector>
#include <cstdarg>
#include <ctime>
```

Namespaces

- namespace csound

Typedefs

- typedef void(* MessageCallbackType)(void *userdata, int attribute, const char *format, va_list valist)

7.28.1. Typedef Documentation

7.28.1.1. typedef void(* csound::MessageCallbackType)(void *userdata, int attribute, const char *format, va_list valist)

Definition at line 42 of file System.hpp.

7.29. frontends/flcsound/canvas.hpp File Reference

7.30. frontends/flcsound/curve.hpp File Reference

Enumerations

- enum Polarity { POLARITY_NOPOL, POLARITY_NEGPOL, POLARITY_POSPOL, POLARITY_BIPOL }

7.30.1. Enumeration Type Documentation

7.30.1.1. enum Polarity

Enumeration values:

POLARITY_NOPOL

POLARITY_NEGPOL

POLARITY_POSPOL

POLARITY_BIPOL

Definition at line 24 of file curve.hpp.

7.31. frontends/flcsound/plots.hpp File Reference

7.32. frontends/flcsound/synthesizer.hpp File Reference

Typedefs

- typedef int(∗ yield_callback_t)(void ∗)
- typedef void(∗ message_callback_t)(void ∗, const char ∗, va_list)
- typedef void(∗ draw_graph_callback_t)(void ∗, Curve ∗)

7.32.1. Typedef Documentation

7.32.1.1. typedef void(∗ draw_graph_callback_t)(void ∗, Curve ∗)

Definition at line 27 of file synthesizer.hpp.

7.32.1.2. typedef void(∗ message_callback_t)(void ∗, const char ∗, va_list)

Definition at line 26 of file synthesizer.hpp.

7.32.1.3. typedef int(∗ yield_callback_t)(void ∗)

Definition at line 25 of file synthesizer.hpp.

7.33. H/csdl.h File Reference

```
#include "csoundCore.h"
```

```
#include <limits.h>
```

Defines

- #define Str(x) (((ENVIRON*) csound) → LocalizeString(x))
- #define LINKAGE
- #define FLINKAGE

7.33.1. Define Documentation

7.33.1.1. #define FLINKAGE

Value:

```
PUBLIC long opcode_size(void)           \
{   if (localops == NULL) return LONG_MIN;              \
    else return ((long) sizeof(localops) | LONG_MIN); } \
                                        \
PUBLIC OENTRY *opcode_init(ENVIRON *xx) \
{   return localops;    }               \
                                        \
PUBLIC NGFENS *fgen_init(ENVIRON *xx)   \
{   return localfgens;  }
```

Definition at line 54 of file csdl.h.

7.33.1.2. #define LINKAGE

Value:

```
PUBLIC long opcode_size(void)           \
{                                       \
    return (long) sizeof(localops);     \
}                                       \
                                        \
PUBLIC OENTRY *opcode_init(ENVIRON *xx) \
{                                       \
    return localops;                    \
}
```

Definition at line 43 of file csdl.h.

7.33.1.3. #define Str(x) (((ENVIRON*) csound) → LocalizeString(x))

Definition at line 41 of file csdl.h.

7.34. H/csound.h File Reference

7.34.1. Detailed Description

Author:
John P. ffitch, Michael Gogins, Matt Ingalls, John D. Ramsdell, and Istvan Varga

Purposes

The purposes of the Csound API are as follows:

- Declare a stable public application programming interface (API) for Csound in csound.h. This is the only header file that needs to be #included by users of the Csound API.
- Hide the internal implementation details of Csound from users of the API, so that development of Csound can proceed without affecting code that uses the API.

Users

Users of the Csound API fall into two main categories: hosts, and plugins.

- Hosts are applications that use Csound as a software synthesis engine. Hosts can link with the Csound API either statically or dynamically.
- Plugins are shared libraries loaded by Csound at run time to implement external opcodes and/or drivers for audio or MIDI input and output.

Hosts using the Csound API must #include <csound.h>, and link with the Csound API library.

Hosts must first create an instance of Csound using the `csoundCreate` API function. When hosts are finished using Csound, they must destroy the instance of csound using the `csoundDestroy` API function. Most of the other Csound API functions take the Csound instance as their first argument. Hosts can call either the standalone API functions defined in csound.h, e.g. `csoundGetSr(csound)`, or the function pointers in the Csound instance structure, e.g. csound->GetSr(csound). Each function in the Csound API has a corresponding function pointer in the Csound instance structure.

Here is the complete code for the simplest possible Csound API host, a command-line Csound application:

```
#include <csound.h>

int main(int argc, char **argv)
{
    void *csound = csoundCreate(0);
    int result = csoundPerform(csound, argc, argv);
    csoundDestroy(csound);
    return result;
}
```

All opcodes, including plugins, receive a pointer to their host instance of Csound as the first argument. Therefore, plugins MUST NOT create an instance of Csound, and MUST call the Csound API function pointers off the Csound instance pointer.

```
MYFLT sr = csound->GetSr(csound);
```

In general, plugins should ONLY access Csound functionality through the API function pointers.

Definition in file csound.h.

```
#include "sysdep.h"
#include "cwindow.h"
#include "opcode.h"
#include "text.h"
#include <stdarg.h>
#include "cfgvar.h"
#include "envvar.h"
#include "fftlib.h"
#include "msg_attr.h"
```

Defines

- #define PUBLIC
- #define LIBRARY_CALL
- #define CSOUNDINIT_NO_SIGNAL_HANDLER 1
- #define CSFILE_FD_R 1
- #define CSFILE_FD_W 2
- #define CSFILE_STD 3
- #define CSFILE_SND_R 4
- #define CSFILE_SND_W 5

Typedefs

- typedef oentry OENTRY
- typedef PUBLIC int(* CsoundRegisterExternalType)(void *csound)
- typedef RTCLOCK_S RTCLOCK

Enumerations

- enum CSOUND_STATUS {
  CSOUND_SUCCESS = 0, CSOUND_ERROR = -1, CSOUND_INITIALIZATION = -2, CSOUND_PERFORMANCE = -3,
  CSOUND_MEMORY = -4 }

Functions

- PUBLIC int csoundInitialize (int *argc, char ***argv, int flags)
- PUBLIC void * csoundCreate (void *hostData)
- PUBLIC int csoundPreCompile (void *csound)
- PUBLIC int csoundQueryInterface (const char *name, void **iface, int *version)
- PUBLIC void csoundDestroy (void *csound)
- PUBLIC int csoundGetVersion (void)
- PUBLIC void * csoundGetHostData (void *csound)
- PUBLIC void csoundSetHostData (void *csound, void *hostData)
- PUBLIC int csoundPerform (void *csound, int argc, char **argv)
- PUBLIC int csoundCompile (void *csound, int argc, char **argv)

- PUBLIC int csoundPerformKsmps (void *csound)
- PUBLIC int csoundPerformKsmpsAbsolute (void *csound)
- PUBLIC int csoundPerformBuffer (void *csound)
- PUBLIC int csoundCleanup (void *csound)
- PUBLIC void csoundReset (void *csound)
- PUBLIC MYFLT csoundGetSr (void *csound)
- PUBLIC MYFLT csoundGetKr (void *csound)
- PUBLIC int csoundGetKsmps (void *csound)
- PUBLIC int csoundGetNchnls (void *csound)
- PUBLIC int csoundGetSampleFormat (void *csound)
- PUBLIC int csoundGetSampleSize (void *csound)
- PUBLIC long csoundGetInputBufferSize (void *csound)
- PUBLIC long csoundGetOutputBufferSize (void *csound)
- PUBLIC void * csoundGetInputBuffer (void *csound)
- PUBLIC void * csoundGetOutputBuffer (void *csound)
- PUBLIC MYFLT * csoundGetSpin (void *csound)
- PUBLIC MYFLT * csoundGetSpout (void *csound)
- PUBLIC MYFLT csoundGetScoreTime (void *csound)
- PUBLIC MYFLT csoundGetProgress (void *csound)
- PUBLIC MYFLT csoundGetProfile (void *csound)
- PUBLIC MYFLT csoundGetCpuUsage (void *csound)
- PUBLIC int csoundIsScorePending (void *csound)
- PUBLIC void csoundSetScorePending (void *csound, int pending)
- PUBLIC MYFLT csoundGetScoreOffsetSeconds (void *csound)
- PUBLIC void csoundSetScoreOffsetSeconds (void *csound, MYFLT time)
- PUBLIC void csoundRewindScore (void *csound)
- PUBLIC void csoundMessage (void *csound, const char *format,...)
- PUBLIC void csoundMessageS (void *csound, int attr, const char *format,...)
- PUBLIC void csoundMessageV (void *csound, int attr, const char *format, va_list args)
- PUBLIC void csoundThrowMessage (void *csound, const char *format,...)
- PUBLIC void csoundThrowMessageV (void *csound, const char *format, va_list args)
- PUBLIC void csoundSetMessageCallback (void *csound, void(*csoundMessage-Callback)(void *csound, int attr, const char *format, va_list valist))
- PUBLIC void csoundSetThrowMessageCallback (void *csound, void(*throwMessage-Callback)(void *csound, const char *format, va_list valist))
- PUBLIC int csoundGetMessageLevel (void *csound)
- PUBLIC void csoundSetMessageLevel (void *csound, int messageLevel)
- PUBLIC void csoundInputMessage (void *csound, const char *message)
- PUBLIC void csoundKeyPress (void *csound, char c)
- PUBLIC void csoundSetInputValueCallback (void *csound, void(*inputValueCalback)(void *csound, char *channelName, MYFLT *value))
- PUBLIC void csoundSetOutputValueCallback (void *csound, void(*outputValue-Calback)(void *csound, char *channelName, MYFLT value))
- PUBLIC int csoundScoreEvent (void *csound, char type, MYFLT *pFields, long numFields)
- int csoundExternalMidiInOpen (void *csound, void **userData, const char *devName)
- int csoundExternalMidiRead (void *csound, void *userData, unsigned char *buf, int nbytes)
- int csoundExternalMidiInClose (void *csound, void *userData)
- int csoundExternalMidiOutOpen (void *csound, void **userData, const char *devName)
- int csoundExternalMidiWrite (void *csound, void *userData, unsigned char *buf, int nbytes)
- int csoundExternalMidiOutClose (void *csound, void *userData)
- char * csoundExternalMidiErrorString (void *csound, int errcode)

- PUBLIC void csoundSetExternalMidiInOpenCallback (void *csound, int(*func)(void *, void **, const char *))
- PUBLIC void csoundSetExternalMidiReadCallback (void *csound, int(*func)(void *, void *, unsigned char *, int))
- PUBLIC void csoundSetExternalMidiInCloseCallback (void *csound, int(*func)(void *, void *))
- PUBLIC void csoundSetExternalMidiOutOpenCallback (void *csound, int(*func)(void *, void **, const char *))
- PUBLIC void csoundSetExternalMidiWriteCallback (void *csound, int(*func)(void *, void *, unsigned char *, int))
- PUBLIC void csoundSetExternalMidiOutCloseCallback (void *csound, int(*func)(void *, void *))
- PUBLIC void csoundSetExternalMidiErrorStringCallback (void *csound, char *(*func)(int))
- PUBLIC void csoundSetIsGraphable (void *csound, int isGraphable)
- PUBLIC void csoundSetMakeGraphCallback (void *csound, void(*makeGraphCallback)(void *csound, WINDAT *windat, char *name))
- PUBLIC void csoundSetDrawGraphCallback (void *csound, void(*drawGraphCallback)(void *csound, WINDAT *windat))
- PUBLIC void csoundSetKillGraphCallback (void *csound, void(*killGraphCallback)(void *csound, WINDAT *windat))
- PUBLIC void csoundSetExitGraphCallback (void *csound, int(*exitGraphCallback)(void *csound))
- PUBLIC opcodelist * csoundNewOpcodeList (void)
- PUBLIC void csoundDisposeOpcodeList (opcodelist *opcodelist_)
- PUBLIC int csoundAppendOpcode (void *csound, char *opname, int dsblksiz, int thread, char *outypes, char *intypes, int(*iopadr)(void *, void *), int(*kopadr)(void *, void *), int(*aopadr)(void *, void *))
- PUBLIC int csoundAppendOpcodes (void *csound, const OENTRY *opcodeList, int n)
- PUBLIC int csoundLoadExternal (void *csound, const char *libraryPath)
- PUBLIC int csoundLoadExternals (void *csound)
- PUBLIC void * csoundOpenLibrary (const char *libraryPath)
- PUBLIC int csoundCloseLibrary (void *library)
- PUBLIC void * csoundGetLibrarySymbol (void *library, const char *symbolName)
- PUBLIC int csoundYield (void *)
- PUBLIC void csoundSetYieldCallback (void *csound, int(*yieldCallback)(void *csound))
- PUBLIC void csoundSetPlayopenCallback (void *csound, int(*playopen__)(void *csound, csRtAudioParams *parm))
- PUBLIC void csoundSetRtplayCallback (void *csound, void(*rtplay__)(void *csound, void *outBuf, int nbytes))
- PUBLIC void csoundSetRecopenCallback (void *csound, int(*recopen_)(void *csound, csRtAudioParams *parm))
- PUBLIC void csoundSetRtrecordCallback (void *csound, int(*rtrecord__)(void *csound, void *inBuf, int nbytes))
- PUBLIC void csoundSetRtcloseCallback (void *csound, void(*rtclose__)(void *csound))
- PUBLIC int csoundGetDebug (void *csound)
- PUBLIC void csoundSetDebug (void *csound, int debug)
- PUBLIC int csoundTableLength (void *csound, int table)
- PUBLIC MYFLT csoundTableGet (void *csound, int table, int index)
- PUBLIC void csoundTableSet (void *csound, int table, int index, MYFLT value)
- PUBLIC void * csoundCreateThread (void *csound, int(*threadRoutine)(void *userdata), void *userdata)
- PUBLIC int csoundJoinThread (void *csound, void *thread)
- PUBLIC void * csoundCreateThreadLock (void *csound)

- PUBLIC void csoundWaitThreadLock (void *csound, void *lock, size_t milliseconds)
- PUBLIC void csoundNotifyThreadLock (void *csound, void *lock)
- PUBLIC void csoundDestroyThreadLock (void *csound, void *lock)
- PUBLIC void csoundSetFLTKThreadLocking (void *csound, int isLocking)
- PUBLIC int csoundGetFLTKThreadLocking (void *csound)
- PUBLIC void timers_struct_init (RTCLOCK *)
- PUBLIC double timers_get_real_time (RTCLOCK *)
- PUBLIC double timers_get_CPU_time (RTCLOCK *)
- PUBLIC unsigned long timers_random_seed (void)
- PUBLIC void csoundSetLanguage (cslanguage_t lang_code)
- PUBLIC char * csoundLocalizeString (const char *s)
- PUBLIC int csoundCreateGlobalVariable (void *csound, const char *name, size_t nbytes)
- PUBLIC void * csoundQueryGlobalVariable (void *csound, const char *name)
- PUBLIC void * csoundQueryGlobalVariableNoCheck (void *csound, const char *name)
- PUBLIC int csoundDestroyGlobalVariable (void *csound, const char *name)
- PUBLIC int csoundGetSizeOfMYFLT (void)
- PUBLIC void ** csoundGetRtRecordUserData (void *csound)
- PUBLIC void ** csoundGetRtPlayUserData (void *csound)
- PUBLIC int csoundRegisterSenseEventCallback (void *csound_, void(*func)(void *, void *), void *userData)
- PUBLIC void * csoundFileOpen (void *csound, void *fd, int type, const char *name, void *param, const char *env)
- PUBLIC void * csoundCreateFileHandle (void *csound, void *fd, int type, const char *fullName)
- PUBLIC char * csoundGetFileName (void *fd)
- PUBLIC int csoundFileClose (void *csound, void *fd)
- PUBLIC int csoundRegisterDeinitCallback (void *csound, void *p, int(*func)(void *, void *))
- PUBLIC int csoundRegisterResetCallback (void *csound, void *userData, int(*func)(void *, void *))

7.34.2. Define Documentation

7.34.2.1. #define CSFILE_FD_R 1

Definition at line 1009 of file csound.h.

7.34.2.2. #define CSFILE_FD_W 2

Definition at line 1010 of file csound.h.

7.34.2.3. #define CSFILE_SND_R 4

Definition at line 1012 of file csound.h.

7.34.2.4. #define CSFILE_SND_W 5

Definition at line 1013 of file csound.h.

7.34.2.5. #define CSFILE_STD 3

Definition at line 1011 of file csound.h.

7.34.2.6. #define CSOUNDINIT_NO_SIGNAL_HANDLER 1

INSTANTIATION

Definition at line 166 of file csound.h.

7.34.2.7. #define LIBRARY_CALL

Definition at line 121 of file csound.h.

7.34.2.8. #define PUBLIC

Platform-dependent definitions and declarations.

Definition at line 120 of file csound.h.

7.34.3. Typedef Documentation

7.34.3.1. typedef PUBLIC int(* CsoundRegisterExternalType)(void *csound)

Signature for external registration function, such as for plugin opcodes or scripting languages. The external has complete access to the Csound API, so a plugin opcode can just call csound->Append-Opcode() for each of its opcodes.

Definition at line 717 of file csound.h.

7.34.3.2. typedef struct oentry OENTRY

7.34.3.3. typedef struct RTCLOCK_S RTCLOCK

7.34.4. Enumeration Type Documentation

7.34.4.1. enum CSOUND_STATUS

ERROR DEFINITIONS

Enumeration values:

CSOUND_SUCCESS
CSOUND_ERROR
CSOUND_INITIALIZATION
CSOUND_PERFORMANCE
CSOUND_MEMORY

Definition at line 147 of file csound.h.

7.34.5. Function Documentation

7.34.5.1. PUBLIC int csoundAppendOpcode (void * *csound*, char * *opname*, int *dsblksiz*, int *thread*, char * *outypes*, char * *intypes*, int(*)(void *, void *) *iopadr*, int(*)(void *, void *) *kopadr*, int(*)(void *, void *) *aopadr*)

Appends an opcode implemented by external software to Csound's internal opcode list. The opcode list is extended by one slot, and the parameters are copied into the new slot. Returns zero on success.

7.34.5.2. PUBLIC int csoundAppendOpcodes (void * *csound*, const OENTRY * *opcodeList*, int *n*)

Appends a list of opcodes implemented by external software to Csound's internal opcode list. The list should either be terminated with an entry that has a NULL opname, or the number of entries (> 0) should be specified in 'n'. Returns zero on success.

7.34.5.3. PUBLIC int csoundCleanup (void * *csound*)

Prints information about the end of a performance. After calling csoundCleanup(), the operation of the perform functions is undefined.

7.34.5.4. PUBLIC int csoundCloseLibrary (void * *library*)

Platform-independent function to unload a shared library.

7.34.5.5. PUBLIC int csoundCompile (void * *csound*, int *argc*, char ** *argv*)

Compiles Csound input files (such as an orchestra and score) as directed by the supplied command-line arguments, but does not perform them. Returns a non-zero error code on failure. In this (host-driven) mode, the sequence of calls should be as follows: /code

csoundCompile(csound, argc, argv, thisObj); while(!csoundPerformBuffer(csound)); csound-Cleanup(csound); csoundReset(csound); /endcode

7.34.5.6. PUBLIC void* csoundCreate (void * *hostData*)

Creates an instance of Csound. Returns an opaque pointer that must be passed to most Csound API functions. The hostData parameter can be null, or it can be a pointer to any sort of data; this pointer can be accessed from the Csound instance that is passed to callback routines.

7.34.5.7. PUBLIC void* csoundCreateFileHandle (void * *csound*, void * *fd*, int *type*, const char * *fullName*)

Allocate a file handle for an existing file already opened with open(), fopen(), or sf_open(), for later use with csoundFileClose() or csoundGetFileName(), or storing in an FDCH structure. Files registered this way (or opened with csoundFileOpen()) are also automatically closed by csound-Reset(). Parameters and return value are similar to csoundFileOpen(), except fullName is the name that will be returned by a later call to csoundGetFileName().

7.34.5.8. PUBLIC int csoundCreateGlobalVariable (void * *csound*, const char * *name*, size_t *nbytes*)

Allocate nbytes bytes of memory that can be accessed later by calling csoundQueryGlobalVariable() with the specified name; the space is cleared to zero. Returns CSOUND_SUCCESS on success, CSOUND_ERROR in case of invalid parameters (zero nbytes, invalid or already used name), or CSOUND_MEMORY if there is not enough memory.

7.34.5.9. PUBLIC void* csoundCreateThread (void * *csound*, int(*)(void *userdata) *threadRoutine*, void * *userdata*)

Creates and starts a new thread of execution. Returns an opaque pointer that represents the thread on success, or null for failure. The userdata pointer is passed to the thread routine.

7.34.5.10. PUBLIC void* csoundCreateThreadLock (void * *csound*)

Creates and returns a monitor object, or null if not successful.

7.34.5.11. PUBLIC void csoundDestroy (void * *csound*)

Destroys an instance of Csound.

7.34.5.12. PUBLIC int csoundDestroyGlobalVariable (void * *csound*, const char * *name*)

Free memory allocated for "name" and remove "name" from the database. Return value is CSOUND_SUCCESS on success, or CSOUND_ERROR if the name is not defined.

7.34.5.13. PUBLIC void csoundDestroyThreadLock (void * *csound*, void * *lock*)

Destroys the indicated monitor object.

7.34.5.14. PUBLIC void csoundDisposeOpcodeList (opcodelist * *opcodelist_*)

Releases an opcode list

7.34.5.15. char* csoundExternalMidiErrorString (void * *csound*, int *errcode*)

Returns pointer to a string constant storing an error massage for error code 'errcode'.

7.34.5.16. int csoundExternalMidiInClose (void * *csound*, void * *userData*)

Close MIDI input device associated with 'userData'. Return value is zero on success, and a non-zero error code on failure.

7.34.5.17. int csoundExternalMidiInOpen (void * *csound*, void ** *userData*, const char * *devName*)

Open MIDI input device 'devName', and store stream specific data pointer in *userData. Return value is zero on success, and a non-zero error code if an error occured.

7.34.5.18. int csoundExternalMidiOutClose (void * *csound*, void * *userData*)

Close MIDI output device associated with '*userData'. Return value is zero on success, and a non-zero error code on failure.

7.34.5.19. int csoundExternalMidiOutOpen (void * *csound*, void ** *userData*, const char * *devName*)

Open MIDI output device 'devName', and store stream specific data pointer in *userData. Return value is zero on success, and a non-zero error code if an error occured.

7.34.5.20. int csoundExternalMidiRead (void * *csound*, void * *userData*, unsigned char * *buf*, int *nbytes*)

Read at most 'nbytes' bytes of MIDI data from input stream 'userData', and store in 'buf'. Returns the actual number of bytes read, which may be zero if there were no events, and negative in case of an error. Note: incomplete messages (such as a note on status without the data bytes) should not be returned.

7.34.5.21. int csoundExternalMidiWrite (void * *csound*, void * *userData*, unsigned char * *buf*, int *nbytes*)

Write 'nbytes' bytes of MIDI data to output stream 'userData' from 'buf' (the buffer will not contain incomplete messages). Returns the actual number of bytes written, or a negative error code.

7.34.5.22. PUBLIC int csoundFileClose (void * *csound*, void * *fd*)

Close a file previously opened with csoundFileOpen().

7.34.5.23. PUBLIC void* csoundFileOpen (void * *csound*, void * *fd*, int *type*, const char * *name*, void * *param*, const char * *env*)

Open a file and return handle.

void *csound: Csound instance pointer void *fd: pointer a variable of type int, FILE*, or SNDFILE*, depending on 'type', for storing handle to be passed to file read/write functions int type: file type, one of the following: CSFILE_FD_R: read file using low level interface (open()) CSFILE_FD_W: write file using low level interface (open()) CSFILE_STD: use ANSI C interface (fopen()) CSFILE_SND_R: read sound file CSFILE_SND_W: write sound file const char *name: file name void *param: parameters, depending on type: CSFILE_FD_R: unused (should be NULL) CSFILE_FD_W: unused (should be NULL) CSFILE_STD: mode parameter (of type char*) to be passed to fopen() CSFILE_SND_R: SF_INFO* parameter for sf_open(), with defaults for raw file; the actual format paramaters of the opened file will be stored in this structure CSFILE_SND_W: SF_INFO* parameter for sf_open(), output file format const char *env: list of environment variables for search path (see csoundFindInputFile() for details); if NULL, the specified name is used as it is, without any conversion or search. return value: opaque handle to the opened file, for use with csoundGetFileName() or csoundFileClose(), or storing in FDCH.fd. On failure, NULL is returned.

7.34.5.24. PUBLIC MYFLT csoundGetCpuUsage (void * *csound*)

Returns the sampsTime vs. calculatedTime ratio (unimplemented).

7.34.5.25. PUBLIC int csoundGetDebug (void * *csound*)

Returns whether Csound is in debug mode.

7.34.5.26. PUBLIC char* csoundGetFileName (void * *fd*)

Get the full name of a file previously opened with csoundFileOpen().

7.34.5.27. PUBLIC int csoundGetFLTKThreadLocking (void * *csound*)

Returns whether or not the FLTK widget thread calls Fl::lock().

7.34.5.28. PUBLIC void* csoundGetHostData (void * *csound*)

Returns host data.

7.34.5.29. PUBLIC void* csoundGetInputBuffer (void * *csound*)

Returns the address of the Csound audio input buffer. Enables external software to write audio into Csound before calling csoundPerformBuffer

7.34.5.30. PUBLIC long csoundGetInputBufferSize (void * *csound*)

Returns the number of samples in Csound's input buffer.

7.34.5.31. PUBLIC MYFLT csoundGetKr (void * *csound*)

Returns the number of control samples per second.

7.34.5.32. PUBLIC int csoundGetKsmps (void * *csound*)

Returns the number of audio sample frames per control sample.

7.34.5.33. PUBLIC void* csoundGetLibrarySymbol (void * *library*, const char * *symbolName*)

Platform-independent function to get a symbol address in a shared library.

7.34.5.34. PUBLIC int csoundGetMessageLevel (void * *csound*)

Returns the Csound message level (from 0 to 7).

7.34.5.35. PUBLIC int csoundGetNchnls (void * *csound*)

Returns the number of audio output channels.

7.34.5.36. PUBLIC void* csoundGetOutputBuffer (void * *csound*)

Returns the address of the Csound audio output buffer. Enables external software to read audio from Csound after calling csoundPerformBuffer.

7.34.5.37. PUBLIC long csoundGetOutputBufferSize (void * *csound*)

Returns the number of samples in Csound's output buffer.

7.34.5.38. PUBLIC MYFLT csoundGetProfile (void * *csound*)

Returns the scoreTime vs. calculatedTime ratio (unimplemented). For real-time performance this value should be always == 1.

7.34.5.39. PUBLIC MYFLT csoundGetProgress (void * *csound*)

Returns the of score completed (unimplemented).

7.34.5.40. PUBLIC void** csoundGetRtPlayUserData (void * *csound*)

Return pointer to user data pointer for real time audio output.

7.34.5.41. PUBLIC void** csoundGetRtRecordUserData (void * *csound*)

Return pointer to user data pointer for real time audio input.

7.34.5.42. PUBLIC int csoundGetSampleFormat (void * *csound*)

Returns the sample format.

7.34.5.43. PUBLIC int csoundGetSampleSize (void * *csound*)

Returns the size in bytes of a single sample.

7.34.5.44. PUBLIC MYFLT csoundGetScoreOffsetSeconds (void * *csound*)

Returns the score time beginning at which score events will actually immediately be performed (see csoundSetScoreOffsetSeconds()).

7.34.5.45. PUBLIC MYFLT csoundGetScoreTime (void * *csound*)

Returns the current score time.

7.34.5.46. PUBLIC int csoundGetSizeOfMYFLT (void)

Return the size of MYFLT in bytes.

7.34.5.47. PUBLIC MYFLT* csoundGetSpin (void * *csound*)

Returns the address of the Csound audio input working buffer (spin). Enables external software to write audio into Csound before calling csoundPerformKsmps.

7.34.5.48. PUBLIC MYFLT* csoundGetSpout (void * *csound*)

Returns the address of the Csound audio output working buffer (spout). Enables external software to read audio from Csound after calling csoundPerformKsmps.

7.34.5.49. PUBLIC MYFLT csoundGetSr (void * *csound*)

Returns the number of audio sample frames per second.

7.34.5.50. PUBLIC int csoundGetVersion (void)

Returns the version number times 100 (4.20 = 420).

7.34.5.51. PUBLIC int csoundInitialize (int * *argc*, char *** *argv*, int *flags*)

Initialise Csound library. Returns zero on success.

7.34.5.52. PUBLIC void csoundInputMessage (void * *csound*, const char * *message*)

Input a NULL-terminated string (as if from a console) usually used for lineevents

7.34.5.53. PUBLIC int csoundIsScorePending (void * *csound*)

Sets whether Csound score events are performed or not, independently of real-time MIDI events (see csoundSetScorePending()).

7.34.5.54. PUBLIC int csoundJoinThread (void * *csound*, void * *thread*)

Waits until the indicated thread's routine has finished. Returns the value returned by the thread routine.

7.34.5.55. PUBLIC void csoundKeyPress (void * *csound*, char *c*)

Set the ASCII code of the most recent key pressed. This value is used by the 'keypress' opcode.

7.34.5.56. PUBLIC int csoundLoadExternal (void * *csound*, const char * *libraryPath*)

Registers all opcodes in the library.

7.34.5.57. PUBLIC int csoundLoadExternals (void * *csound*)

Registers all opcodes in all libraries in the opcodes directory.

7.34.5.58. PUBLIC char* csoundLocalizeString (const char * *s*)

Translate string 's' to the current language, and return pointer to the translated message. This may be the same as 's' if language was set to CSLANGUAGE_DEFAULT.

7.34.5.59. PUBLIC void csoundMessage (void * *csound*, const char * *format*, ...)

Displays an informational message.

7.34.5.60. PUBLIC void csoundMessageS (void * *csound*, int *attr*, const char * *format*, ...)

Print message with special attributes (see msg_attr.h for the list of available attributes). With attr=0, csoundMessageS() is identical to csoundMessage().

7.34.5.61. PUBLIC void csoundMessageV (void * *csound*, int *attr*, const char * *format*, va_list *args*)

7.34.5.62. PUBLIC opcodelist* csoundNewOpcodeList (void)

Gets a list of all opcodes. Make sure to call csoundDisposeOpcodeList() when done with the list.

7.34.5.63. PUBLIC void csoundNotifyThreadLock (void * *csound*, void * *lock*)

Notifies the indicated monitor object.

7.34.5.64. PUBLIC void* csoundOpenLibrary (const char * *libraryPath*)

Platform-independent function to load a shared library.

7.34.5.65. PUBLIC int csoundPerform (void * *csound*, int *argc*, char ** *argv*)

Compiles and renders a Csound performance, as directed by the supplied command-line arguments, in one pass. Returns 0 for success.

7.34.5.66. PUBLIC int csoundPerformBuffer (void * *csound*)

Performs Csound, sensing real-time and score events and processing one buffer's worth (-b frames) of interleaved audio. Returns a pointer to the new output audio in 'outputAudio' Note that csoundCompile must be called first, then call csoundGetOutputBuffer() and csoundGetInputBuffer() to get the pointer to csound's I/O buffers. Returns false during performance, and true when performance is finished.

7.34.5.67. PUBLIC int csoundPerformKsmps (void * *csound*)

Senses input events, and performs one control sample worth (ksmps) of audio output. Note that csoundCompile must be called first. Returns false during performance, and true when performance is finished. If called until it returns true, will perform an entire score. Enables external software to control the execution of Csound, and to synchronize performance with audio input and output.

7.34.5.68. PUBLIC int csoundPerformKsmpsAbsolute (void * *csound*)

Senses input events, and performs one control sample worth (ksmps) of audio output. Note that csoundCompile must be called first. Performs audio whether or not the Csound score has finished. Enables external software to control the execution of Csound, and to synchronize performance with audio input and output.

7.34.5.69. PUBLIC int csoundPreCompile (void * *csound*)

Reset and prepare an instance of Csound for compilation. Returns CSOUND_SUCCESS on success, and CSOUND_ERROR or CSOUND_MEMORY if an error occured.

7.34.5.70. PUBLIC void* csoundQueryGlobalVariable (void * *csound*, const char * *name*)

Get pointer to space allocated with the name "name". Returns NULL if the specified name is not defined.

7.34.5.71. PUBLIC void* csoundQueryGlobalVariableNoCheck (void * *csound*, const char * *name*)

This function is the same as csoundQueryGlobalVariable(), except the variable is assumed to exist and no error checking is done. Faster, but may crash or return an invalid pointer if 'name' is not defined.

7.34.5.72. PUBLIC int csoundQueryInterface (const char * *name*, void ** *iface*, int * *version*)

Returns a pointer to the requested interface, if available, in the interface argument, and its version number, in the version argument. Returns 0 for success and 1 for failure.

7.34.5.73. PUBLIC int csoundRegisterDeinitCallback (void * *csound*, void * *p*, int(*)(void *, void *) *func*)

Register a function to be called at note deactivation. Should be called from the initialisation routine of an opcode. 'p' is a pointer to the OPDS structure of the opcode, and 'func' is the function to be called, with the same arguments and return value as in the case of opcode init/perf functions. The functions are called in reverse order of registration. Returns zero on success.

7.34.5.74. PUBLIC int csoundRegisterResetCallback (void * *csound*, void * *userData*, int(*)(void *, void *) *func*)

Register a function to be called by csoundReset(), in reverse order of registration, before unloading external modules. The function takes the Csound instance pointer as the first argument, and the pointer passed here as 'userData' as the second, and is expected to return zero on success. The return value of csoundRegisterResetCallback() is zero on success.

7.34.5.75. PUBLIC int csoundRegisterSenseEventCallback (void * *csound_*, void(*)(void *, void *) *func*, void * *userData*)

Register a function to be called once in every control period by sensevents(). Any number of functions may be registered. The callback function takes two arguments of type void*, the first is the Csound instance pointer, and the second is the userData pointer as passed to this function. Returns zero on success.

7.34.5.76. PUBLIC void csoundReset (void * *csound*)

Resets all internal memory and state in preparation for a new performance. Enables external software to run successive Csound performances without reloading Csound. Implies csoundCleanup(), unless already called.

7.34.5.77. PUBLIC void csoundRewindScore (void * *csound*)

Rewinds a compiled Csound score to its beginning.

7.34.5.78. PUBLIC int csoundScoreEvent (void * *csound*, char *type*, MYFLT * *pFields*, long *numFields*)

Send a new score event. 'type' is the score event type ('i', 'f', or 'e') 'numFields' is the size of the pFields array. 'pFields' is an array of floats with all the pfields for this event, starting with the p1 value specified in pFields[0].

7.34.5.79. PUBLIC void csoundSetDebug (void * *csound*, int *debug*)

Sets whether Csound is in debug mode.

7.34.5.80. PUBLIC void csoundSetDrawGraphCallback (void * *csound*, void(*)(void *csound, WINDAT *windat) *drawGraphCallback*)

Called by external software to set Csound's DrawGraph function.

7.34.5.81. PUBLIC void csoundSetExitGraphCallback (void * *csound*, int(*)(void *csound) *exitGraphCallback*)

Called by external software to set Csound's ExitGraph function.

7.34.5.82. PUBLIC void csoundSetExternalMidiErrorStringCallback (void * *csound*, char *(*)(int) *func*)

7.34.5.83. PUBLIC void csoundSetExternalMidiInCloseCallback (void * *csound*, int(*)(void *, void *) *func*)

7.34.5.84. PUBLIC void csoundSetExternalMidiInOpenCallback (void * *csound*, int(*)(void *, void **, const char *) *func*)

7.34.5.85. PUBLIC void csoundSetExternalMidiOutCloseCallback (void * *csound*, int(*)(void *, void *) *func*)

7.34.5.86. PUBLIC void csoundSetExternalMidiOutOpenCallback (void * *csound*, int(*)(void *, void **, const char *) *func*)

7.34.5.87. PUBLIC void csoundSetExternalMidiReadCallback (void * *csound*, int(*)(void *, void *, unsigned char *, int) *func*)

7.34.5.88. PUBLIC void csoundSetExternalMidiWriteCallback (void * *csound*, int(*)(void *, void *, unsigned char *, int) *func*)

7.34.5.89. PUBLIC void csoundSetFLTKThreadLocking (void * *csound*, int *isLocking*)

Sets whether or not the FLTK widget thread calls Fl::lock().

7.34.5.90. PUBLIC void csoundSetHostData (void * *csound*, void * *hostData*)

Sets host data.

7.34.5.91. PUBLIC void csoundSetInputValueCallback (void * *csound*, void(*)(void *csound, char *channelName, MYFLT *value) *inputValueCalback*)

Called by external software to set a function for Csound to fetch input control values. The 'invalue' opcodes will directly call this function.

7.34.5.92. PUBLIC void csoundSetIsGraphable (void * *csound*, int *isGraphable*)

Tells Csound supports external graphic table display.

7.34.5.93. PUBLIC void csoundSetKillGraphCallback (void * *csound*, void(*)(void *csound, WINDAT *windat) *killGraphCallback*)

Called by external software to set Csound's KillGraph function.

7.34.5.94. PUBLIC void csoundSetLanguage (cslanguage_t *lang_code*)

Set language to 'lang_code' (lang_code can be for example CSLANGUAGE_ENGLISH_UK or CSLANGUAGE_FRENCH or many others, see n_getstr.h for the list of languages). This affects all Csound instances running in the address space of the current process. The special language code CSLANGUAGE_DEFAULT can be used to disable translation of messages and free all memory allocated by a previous call to csoundSetLanguage(). csoundSetLanguage() loads all files for the selected language from the directory specified by the CSSTRNGS environment variable.

7.34.5.95. PUBLIC void csoundSetMakeGraphCallback (void * *csound*, void(*)(void *csound, WINDAT *windat, char *name) *makeGraphCallback*)

Called by external software to set Csound's MakeGraph function.

7.34.5.96. PUBLIC void csoundSetMessageCallback (void * *csound*, void(*)(void *csound, int attr, const char *format, va_list valist) *csoundMessageCallback*)

Sets a function to be called by Csound to print an informational message.

7.34.5.97. PUBLIC void csoundSetMessageLevel (void * *csound*, int *messageLevel*)

Sets the Csound message level (from 0 to 7).

7.34.5.98. PUBLIC void csoundSetOutputValueCallback (void * *csound*, void(*)(void *csound, char *channelName, MYFLT value) *outputValueCalback*)

Called by external software to set a function for Csound to send output control values. The 'outvalue' opcodes will directly call this function.

7.34.5.99. PUBLIC void csoundSetPlayopenCallback (void * *csound*, int(*)(void *csound, csRtAudioParams *parm) *playopen__*)

Sets a function to be called by Csound for opening real-time audio playback.

7.34.5.100. PUBLIC void csoundSetRecopenCallback (void * *csound*, int(*)(void *csound, csRtAudioParams *parm) *recopen_*)

Sets a function to be called by Csound for opening real-time audio recording.

7.34.5.101. PUBLIC void csoundSetRtcloseCallback (void * *csound*, void(*)(void *csound) *rtclose__*)

Sets a function to be called by Csound for closing real-time audio playback and recording.

7.34.5.102. PUBLIC void csoundSetRtplayCallback (void * *csound*, void(*)(void *csound, void *outBuf, int nbytes) *rtplay__*)

Sets a function to be called by Csound for performing real-time audio playback.

7.34.5.103. PUBLIC void csoundSetRtrecordCallback (void * *csound*, int(*)(void *csound, void *inBuf, int nbytes) *rtrecord__*)

Sets a function to be called by Csound for performing real-time audio recording.

7.34.5.104. PUBLIC void csoundSetScoreOffsetSeconds (void * *csound*, MYFLT *time*)

Csound score events prior to the specified time are not performed, and performance begins immediately at the specified time (real-time events will continue to be performed as they are received). Can be used by external software, such as a VST host, to begin score performance midway through a Csound score, for example to repeat a loop in a sequencer, or to synchronize other events with the Csound score.

7.34.5.105. PUBLIC void csoundSetScorePending (void * *csound*, int *pending*)

Sets whether Csound score events are performed or not (real-time events will continue to be performed). Can be used by external software, such as a VST host, to turn off performance of score events (while continuing to perform real-time events), for example to mute a Csound score while working on other tracks of a piece, or to play the Csound instruments live.

7.34.5.106. PUBLIC void csoundSetThrowMessageCallback (void * *csound*, void(*)(void *csound, const char *format, va_list valist) *throwMessageCallback*)

Sets a function for Csound to stop execution with an error message or exception.

7.34.5.107. PUBLIC void csoundSetYieldCallback (void * *csound*, int(*)(void *csound) *yieldCallback*)

Called by external software to set a function for checking system events, yielding cpu time for coopertative multitasking, etc.. This function is optional. It is often used as a way to 'turn off'

Csound, allowing it to exit gracefully. In addition, some operations like utility analysis routines are not reentrant and you should use this function to do any kind of updating during the operation.

Returns an 'OK to continue' boolean

7.34.5.108. PUBLIC MYFLT csoundTableGet (void * *csound*, int *table*, int *index*)

Returns the value of a slot in a function table.

7.34.5.109. PUBLIC int csoundTableLength (void * *csound*, int *table*)

Returns the length of a function table, or -1 if the table does not exist.

7.34.5.110. PUBLIC void csoundTableSet (void * *csound*, int *table*, int *index*, MYFLT *value*)

Sets the value of a slot in a function table.

7.34.5.111. PUBLIC void csoundThrowMessage (void * *csound*, const char * *format*, ...)

Throws an informational message as a C++ exception.

7.34.5.112. PUBLIC void csoundThrowMessageV (void * *csound*, const char * *format*, va_list *args*)

7.34.5.113. PUBLIC void csoundWaitThreadLock (void * *csound*, void * *lock*, size_t *milliseconds*)

Waits on the indicated monitor object for the indicated period. The function returns either when the monitor object is notified, or when the period has elapsed, whichever is sooner. If the period is 0, the wait is infinite.

7.34.5.114. PUBLIC int csoundYield (void *)

Check system events, yielding cpu time for coopertative multitasking, etc.

7.34.5.115. PUBLIC double timers_get_CPU_time (RTCLOCK *)

Return the elapsed CPU time (in seconds) since the specified timer structure was initialised.

7.34.5.116. PUBLIC double timers_get_real_time (RTCLOCK *)

Return the elapsed real time (in seconds) since the specified timer structure was initialised.

7.34.5.117. PUBLIC unsigned long timers_random_seed (void)

Return a 32-bit unsigned integer to be used as seed from current time.

7.34.5.118. PUBLIC void timers_struct_init (RTCLOCK *)

initialise a timer structure

7.35. H/csoundCore.h File Reference

```
#include "sysdep.h"
#include <stdarg.h>
#include <setjmp.h>
#include "cwindow.h"
#include "opcode.h"
#include "version.h"
#include <sndfile.h>
#include "csound.h"
#include "cs_util.h"
#include "sort.h"
#include "midiops2.h"
#include "text.h"
#include "prototyp.h"
```

Defines

- #define OK (0)
- #define NOTOK (-1)
- #define CSOUND_EXITJMP_SUCCESS (256)
- #define INSTR 1
- #define ENDIN 2
- #define OPCODE 3
- #define ENDOP 4
- #define LABEL 5
- #define SETBEG 6
- #define PSET 6
- #define SETEND 7
- #define MAXINSNO (200)
- #define PMAX (1000)
- #define VARGMAX (1001)
- #define TOKMAX 50L
- #define OPCODENUMOUTS 24
- #define ORTXT h.optext → t
- #define INCOUNT ORTXT.inlist → count
- #define OUTCOUNT ORTXT.outlist → count
- #define INOCOUNT ORTXT.inoffs → count
- #define OUTOCOUNT ORTXT.outoffs → count
- #define XINCODE ORTXT.xincod
- #define XINARG1 (p → XINCODE & 1)
- #define XINARG2 (p → XINCODE & 2)
- #define XINARG3 (p → XINCODE & 4)
- #define XINARG4 (p → XINCODE & 8)
- #define XOUTCODE ORTXT.xoutcod
- #define XSTRCODE ORTXT.xincod_str
- #define XOUTSTRCODE ORTXT.xoutcod_str

- #define MAXLEN 0x1000000L
- #define FMAXLEN ((MYFLT)(MAXLEN))
- #define PHMASK 0x0FFFFFFL
- #define PFRAC(x) ((MYFLT)((x) & ftp → lomask) * ftp → lodiv)
- #define MAXPOS 0x7FFFFFFFL
- #define BYTREVS(n) ((n>>8 & 0xFF) | (n<<8 & 0xFF00))
- #define BYTREVL(n)
- #define OCTRES 8192
- #define CPSOCTL(n) ((MYFLT)(1 << ((int)(n) >> 13)) * cpsocfrc[(int)(n) & 8191])
- #define LOBITS 10
- #define LOFACT 1024
- #define LOSCAL FL(0.0009765625)
- #define LOMASK 1023
- #define SSTRCOD 3945467
- #define SSTRSIZ 200
- #define ALLCHNLS 0x7fff
- #define DFLT_SR FL(44100.0)
- #define DFLT_KR FL(4410.0)
- #define DFLT_KSMPS 10
- #define DFLT_NCHNLS 1
- #define MAXCHNLS 256
- #define MAXNAME (256)
- #define DFLT_DBFS (FL(32768.0))
- #define ONEPT 1.02197486
- #define LOG10D20 0.11512925
- #define DV32768 FL(0.000030517578125)
- #define DKBAS 25
- #define MAXOCTS 8
- #define AIFF_MAXCHAN 8
- #define MAXCHAN 16
- #define MBUFSIZ (4096)
- #define MIDIINBUFMAX (1024)
- #define MIDIINBUFMSK (MIDIINBUFMAX-1)
- #define __cdecl
- #define PI (3.14159265358979323846)
- #define TWOPI (6.28318530717958647692)
- #define PI_F ((MYFLT) PI)
- #define TWOPI_F ((MYFLT) TWOPI)
- #define WARNMSG 04

Typedefs

- typedef polish POLISH
- typedef arglst ARGLST
- typedef argoffs ARGOFFS
- typedef text TEXT
- typedef instr INSTRTXT
- typedef op OPTXT
- typedef fdch FDCH
- typedef auxch AUXCH
- typedef monblk MONPCH
- typedef dklst DKLST

- typedef mchnblk MCHNBLK
- typedef insds INSDS
- typedef int(* SUBR)(void *, void *)
- typedef opds OPDS
- typedef lblblk LBLBLK
- typedef MEMFIL MEMFIL
- typedef event EVTBLK
- typedef eventnode EVTNODE
- typedef opcodinfo OPCODINFO
- typedef void(* GEN)(FUNC *, struct ENVIRON_ *)
- typedef midiglobals MGLOBAL
- typedef token TOKEN
- typedef names NAMES
- typedef SNDMEMFILE_ SNDMEMFILE
- typedef pvx_memfile_ PVOCEX_MEMFILE
- typedef ENVIRON_ ENVIRON

Functions

- void dbfs_init (struct ENVIRON_ *csound, MYFLT dbfs)

7.35.1. Define Documentation

7.35.1.1. #define __cdecl

Definition at line 1012 of file csoundCore.h.

7.35.1.2. #define AIFF_MAXCHAN 8

Definition at line 359 of file csoundCore.h.

7.35.1.3. #define ALLCHNLS 0x7fff

Definition at line 103 of file csoundCore.h.

7.35.1.4. #define BYTREVL(n)

Value:

```
((n>>24 & 0xFF) | (n>>8 & 0xFF00L) | \
                  (n<<8 & 0xFF0000L) | (n<<24 & 0xFF000000L))
```

Definition at line 88 of file csoundCore.h.

7.35.1.5. #define BYTREVS(n) ((n>>8 & 0xFF) | (n<<8 & 0xFF00))

Definition at line 87 of file csoundCore.h.

7.35.1.6. #define CPSOCTL(n) ((MYFLT)(1 << ((int)(n) >> 13)) * cpsocfrc[(int)(n) & 8191])

Definition at line 92 of file csoundCore.h.

7.35.1.7. #define CSOUND_EXITJMP_SUCCESS (256)

Definition at line 49 of file csoundCore.h.

7.35.1.8. #define DFLT_DBFS (FL(32768.0))

Definition at line 112 of file csoundCore.h.

7.35.1.9. #define DFLT_KR FL(4410.0)

Definition at line 105 of file csoundCore.h.

7.35.1.10. #define DFLT_KSMPS 10

Definition at line 106 of file csoundCore.h.

7.35.1.11. #define DFLT_NCHNLS 1

Definition at line 107 of file csoundCore.h.

7.35.1.12. #define DFLT_SR FL(44100.0)

Definition at line 104 of file csoundCore.h.

7.35.1.13. #define DKBAS 25

Definition at line 249 of file csoundCore.h.

7.35.1.14. #define DV32768 FL(0.000030517578125)

Definition at line 146 of file csoundCore.h.

7.35.1.15. #define ENDIN 2

Definition at line 52 of file csoundCore.h.

7.35.1.16. #define ENDOP 4

Definition at line 54 of file csoundCore.h.

7.35.1.17. #define FMAXLEN ((MYFLT)(MAXLEN))

Definition at line 82 of file csoundCore.h.

7.35.1.18. #define INCOUNT ORTXT.inlist → count

Definition at line 68 of file csoundCore.h.

7.35.1.19. #define INOCOUNT ORTXT.inoffs → count

Definition at line 70 of file csoundCore.h.

7.35.1.20. #define INSTR 1

Definition at line 51 of file csoundCore.h.

7.35.1.21. #define LABEL 5

Definition at line 55 of file csoundCore.h.

7.35.1.22. #define LOBITS 10

Definition at line 94 of file csoundCore.h.

7.35.1.23. #define LOFACT 1024

Definition at line 95 of file csoundCore.h.

7.35.1.24. #define LOG10D20 0.11512925

Definition at line 145 of file csoundCore.h.

7.35.1.25. #define LOMASK 1023

Definition at line 99 of file csoundCore.h.

7.35.1.26. #define LOSCAL FL(0.0009765625)

Definition at line 97 of file csoundCore.h.

7.35.1.27. #define MAXCHAN 16

Definition at line 445 of file csoundCore.h.

7.35.1.28. #define MAXCHNLS 256

Definition at line 108 of file csoundCore.h.

7.35.1.29. #define MAXINSNO (200)

Definition at line 60 of file csoundCore.h.

7.35.1.30. #define MAXLEN 0x1000000L

Definition at line 81 of file csoundCore.h.

7.35.1.31. #define MAXNAME (256)

Definition at line 110 of file csoundCore.h.

7.35.1.32. #define MAXOCTS 8

Definition at line 343 of file csoundCore.h.

7.35.1.33. #define MAXPOS 0x7FFFFFFFL

Definition at line 85 of file csoundCore.h.

7.35.1.34. #define MBUFSIZ (4096)

Definition at line 454 of file csoundCore.h.

7.35.1.35. #define MIDIINBUFMAX (1024)

Definition at line 455 of file csoundCore.h.

7.35.1.36. #define MIDIINBUFMSK (MIDIINBUFMAX-1)

Definition at line 456 of file csoundCore.h.

7.35.1.37. #define NOTOK (-1)

Definition at line 45 of file csoundCore.h.

Referenced by OpcodeBase< T >::audio(), OpcodeBase< T >::init(), and OpcodeBase< T >::kontrol().

7.35.1.38. #define OCTRES 8192

Definition at line 91 of file csoundCore.h.

7.35.1.39. #define OK (0)

Definition at line 44 of file csoundCore.h.

Referenced by OpcodeBase< T >::noteoff().

7.35.1.40. #define ONEPT 1.02197486

Definition at line 144 of file csoundCore.h.

7.35.1.41. #define OPCODE 3

Definition at line 53 of file csoundCore.h.

7.35.1.42. #define OPCODENUMOUTS 24

Definition at line 65 of file csoundCore.h.

7.35.1.43. #define ORTXT h.optext → t

Definition at line 67 of file csoundCore.h.

7.35.1.44. #define OUTCOUNT ORTXT.outlist → count

Definition at line 69 of file csoundCore.h.

7.35.1.45. #define OUTOCOUNT ORTXT.outoffs → count

Definition at line 71 of file csoundCore.h.

7.35.1.46. #define PFRAC(x) ((MYFLT)((x) & ftp → lomask) ∗ ftp → lodiv)

Definition at line 84 of file csoundCore.h.

7.35.1.47. #define PHMASK 0x0FFFFFFL

Definition at line 83 of file csoundCore.h.

7.35.1.48. #define PI (3.14159265358979323846)

Definition at line 1018 of file csoundCore.h.

7.35.1.49. #define PI_F ((MYFLT) PI)

Definition at line 1021 of file csoundCore.h.

7.35.1.50. #define PMAX (1000)

Definition at line 61 of file csoundCore.h.

7.35.1.51. #define PSET 6

Definition at line 57 of file csoundCore.h.

7.35.1.52. #define SETBEG 6

Definition at line 56 of file csoundCore.h.

7.35.1.53. #define SETEND 7

Definition at line 58 of file csoundCore.h.

7.35.1.54. #define SSTRCOD 3945467

Definition at line 101 of file csoundCore.h.

7.35.1.55. #define SSTRSIZ 200

Definition at line 102 of file csoundCore.h.

7.35.1.56. #define TOKMAX 50L

Definition at line 63 of file csoundCore.h.

7.35.1.57. #define TWOPI (6.28318530717958647692)

Definition at line 1020 of file csoundCore.h.

7.35.1.58. #define TWOPI_F ((MYFLT) TWOPI)

Definition at line 1022 of file csoundCore.h.

7.35.1.59. #define VARGMAX (1001)

Definition at line 62 of file csoundCore.h.

7.35.1.60. #define WARNMSG 04

Definition at line 1024 of file csoundCore.h.
Referenced by OpcodeBase< T >::warn().

7.35.1.61. #define XINARG1 (p → XINCODE & 1)

Definition at line 73 of file csoundCore.h.

7.35.1.62. #define XINARG2 (p → XINCODE & 2)

Definition at line 74 of file csoundCore.h.

7.35.1.63. #define XINARG3 (p → XINCODE & 4)

Definition at line 75 of file csoundCore.h.

7.35.1.64. #define XINARG4 (p → XINCODE & 8)

Definition at line 76 of file csoundCore.h.

7.35.1.65. #define XINCODE ORTXT.xincod

Definition at line 72 of file csoundCore.h.

7.35.1.66. #define XOUTCODE ORTXT.xoutcod

Definition at line 77 of file csoundCore.h.

7.35.1.67. #define XOUTSTRCODE ORTXT.xoutcod_str

Definition at line 79 of file csoundCore.h.

7.35.1.68. #define XSTRCODE ORTXT.xincod_str

Definition at line 78 of file csoundCore.h.

7.35.2. Typedef Documentation

7.35.2.1. typedef struct arglst ARGLST

7.35.2.2. typedef struct argoffs ARGOFFS

7.35.2.3. typedef struct auxch AUXCH

7.35.2.4. typedef struct dklst DKLST

7.35.2.5. typedef struct ENVIRON_ ENVIRON

7.35.2.6. typedef struct event EVTBLK

7.35.2.7. typedef struct eventnode EVTNODE

7.35.2.8. typedef struct fdch FDCH

7.35.2.9. typedef void(* GEN)(FUNC *, struct ENVIRON_ *)

Definition at line 450 of file csoundCore.h.

7.35.2.10. typedef struct insds INSDS

7.35.2.11. typedef struct instr INSTRTXT

7.35.2.12. typedef struct lblblk LBLBLK

7.35.2.13. typedef struct mchnblk MCHNBLK

7.35.2.14. typedef struct MEMFIL MEMFIL

7.35.2.15. typedef struct midiglobals MGLOBAL

7.35.2.16. typedef struct monblk MONPCH

7.35.2.17. typedef struct names NAMES

7.35.2.18. typedef struct opcodinfo OPCODINFO

7.35.2.19. typedef struct opds OPDS

7.35.2.20. typedef struct op OPTXT

7.35.2.21. typedef struct polish POLISH

7.35.2.22. typedef struct pvx_memfile_ PVOCEX_MEMFILE

7.35.2.23. typedef struct SNDMEMFILE_ SNDMEMFILE

7.35.2.24. typedef int(* SUBR)(void *, void *)

Definition at line 314 of file csoundCore.h.

7.35.2.25. typedef struct text TEXT

7.35.2.26. typedef struct token TOKEN

7.35.3. Function Documentation

7.35.3.1. struct void dbfs_init (struct ENVIRON_ * *csound*, MYFLT *dbfs*)

7.36. H/OpcodeBase.hpp File Reference

```
#include "csoundCore.h"
#include <cstdarg>
```

7.37. Opcodes/filter.h File Reference

Defines

- #define MAXZEROS 50
- #define MAXPOLES 50

7.37.1. Define Documentation

7.37.1.1. #define MAXPOLES 50

Definition at line 36 of file filter.h.

7.37.1.2. #define MAXZEROS 50

Definition at line 35 of file filter.h.

7.38. Opcodes/fluidOpcodes/fluidOpcodes.hpp File Reference

```
#include <fluidsynth.h>
#include "csoundCore.h"
```

7.39. Opcodes/Loris/src/AiffData.h File Reference

```
#include "Marker.h"
#include <string>
#include <vector>
```

Namespaces

- namespace Loris

Defines

- #define SIZEOF_SHORT 2
- #define SIZEOF_INT 4
- #define SIZEOF_LONG 4

Typedefs

- typedef short Int_16
- typedef unsigned short Uint_16
- typedef int Int_32
- typedef unsigned int Uint_32
- typedef unsigned long ID
- typedef char Byte

Enumerations

- enum {

  ContainerId = 0x464f524d, AiffType = 0x41494646, CommonId = 0x434f4d4d, ApplicationSpecificId = 0x4150504c,

  SosEnvelopesId = 0x534f5365, SoundDataId = 0x53534e44, InstrumentId = 0x494e5354, MarkerId = 0x4d41524b }

Functions

- std::istream & readChunkHeader (std::istream &s, CkHeader &h)
- std::istream & readApplicationSpecifcData (std::istream &s, SosEnvelopesCk &ck, unsigned long chunkSize)
- std::istream & readCommonData (std::istream &s, CommonCk &ck, unsigned long chunkSize)
- std::istream & readContainer (std::istream &s, ContainerCk &ck, unsigned long chunkSize)
- std::istream & readInstrumentData (std::istream &s, InstrumentCk &ck, unsigned long chunkSize)
- std::istream & readMarkerData (std::istream &s, MarkerCk &ck, unsigned long chunkSize)
- std::istream & readSampleData (std::istream &s, SoundDataCk &ck, unsigned long chunkSize)
- void configureCommonCk (CommonCk &ck, unsigned long nFrames, unsigned int nChans, unsigned int bps, double srate)
- void configureContainer (ContainerCk &ck, unsigned long dataSize)

- void configureInstrumentCk (InstrumentCk &ck, double midiNoteNum)
- void configureMarkerCk (MarkerCk &ck, const std::vector< Marker > &markers, double srate)
- void configureSoundDataCk (SoundDataCk &ck, const std::vector< double > &samples, unsigned int bps)
- std::ostream & writeCommonData (std::ostream &s, const CommonCk &ck)
- std::ostream & writeContainer (std::ostream &s, const ContainerCk &ck)
- std::ostream & writeInstrumentData (std::ostream &s, const InstrumentCk &ck)
- std::ostream & writeMarkerData (std::ostream &s, const MarkerCk &ck)
- std::ostream & writeSampleData (std::ostream &s, const SoundDataCk &ck)
- void convertBytesToSamples (const std::vector< Byte > &bytes, std::vector< double > &samples, unsigned int bps)
- void convertSamplesToBytes (const std::vector< double > &samples, std::vector< Byte > &bytes, unsigned int bps)

7.39.1. Define Documentation

7.39.1.1. #define SIZEOF_INT 4

Definition at line 47 of file AiffData.h.

7.39.1.2. #define SIZEOF_LONG 4

Definition at line 51 of file AiffData.h.

7.39.1.3. #define SIZEOF_SHORT 2

Definition at line 43 of file AiffData.h.

7.39.2. Typedef Documentation

7.39.2.1. typedef char Loris::Byte

Definition at line 93 of file AiffData.h.

7.39.2.2. typedef unsigned long Loris::ID

Definition at line 92 of file AiffData.h.

7.39.2.3. typedef short Int_16

Definition at line 56 of file AiffData.h.

7.39.2.4. typedef int Int_32

Definition at line 66 of file AiffData.h.

7.39.2.5. typedef unsigned short Uint_16

Definition at line 57 of file AiffData.h.

7.39.2.6. typedef unsigned int Uint_32

Definition at line 67 of file AiffData.h.

7.39.3. Enumeration Type Documentation

7.39.3.1. anonymous enum

Enumeration values:

ContainerId

AiffType

CommonId

ApplicationSpecificId

SosEnvelopesId

SoundDataId

InstrumentId

MarkerId

Definition at line 80 of file AiffData.h.

7.39.4. Function Documentation

7.39.4.1. void configureCommonCk (CommonCk & *ck*, unsigned long *nFrames*, unsigned int *nChans*, unsigned int *bps*, double *srate*)

7.39.4.2. void configureContainer (ContainerCk & *ck*, unsigned long *dataSize*)

7.39.4.3. void configureInstrumentCk (InstrumentCk & *ck*, double *midiNoteNum*)

7.39.4.4. void configureMarkerCk (MarkerCk & *ck*, const std::vector< Marker > & *markers*, double *srate*)

7.39.4.5. void Loris::configureSoundDataCk (SoundDataCk & *ck*, const std::vector< double > & *samples*, unsigned int *bps*)

7.39.4.6. void convertBytesToSamples (const std::vector< Byte > & *bytes*, std::vector< double > & *samples*, unsigned int *bps*)

7.39.4.7. void convertSamplesToBytes (const std::vector< double > & *samples*, std::vector< Byte > & *bytes*, unsigned int *bps*)

7.39.4.8. std::istream& readApplicationSpecifcData (std::istream & *s*, SosEnvelopesCk & *ck*, unsigned long *chunkSize*)

7.39.4.9. std::istream& readChunkHeader (std::istream & *s*, CkHeader & *h*)

7.39.4.10. std::istream& readCommonData (std::istream & *s*, CommonCk & *ck*, unsigned long *chunkSize*)

7.39.4.11. std::istream& readContainer (std::istream & *s*, ContainerCk & *ck*, unsigned long *chunkSize*)

7.39.4.12. std::istream& readInstrumentData (std::istream & *s*, InstrumentCk & *ck*, unsigned long *chunkSize*)

7.39.4.13. std::istream& readMarkerData (std::istream & *s*, MarkerCk & *ck*, unsigned long *chunkSize*)

7.39.4.14. std::istream& readSampleData (std::istream & *s*, SoundDataCk & *ck*, unsigned long *chunkSize*)

7.39.4.15. std::ostream& writeCommonData (std::ostream & *s*, const CommonCk & *ck*)

7.39.4.16. std::ostream& writeContainer (std::ostream & *s*, const ContainerCk & *ck*)

7.39.4.17. std::ostream& writeInstrumentData (std::ostream & *s*, const InstrumentCk & *ck*)

7.39.4.18. std::ostream& writeMarkerData (std::ostream & *s*, const MarkerCk & *ck*)

7.39.4.19. std::ostream& writeSampleData (std::ostream & *s*, const SoundDataCk & *ck*)

7.40. Opcodes/Loris/src/AiffFile.h File Reference

```
#include "Marker.h"
#include "Synthesizer.h"
#include <memory>
#include <string>
#include <vector>
```

Namespaces

- namespace Loris

7.41. Opcodes/Loris/src/Analyzer.h File Reference

```
#include <memory>
#include <vector>
#include "Partial.h"
#include "PartialList.h"
```

Namespaces

- namespace Loris

7.42. Opcodes/Loris/src/AssociateBandwidth.h File Reference

```
#include "SpectralPeaks.h"
#include <vector>
```

Namespaces

- namespace Loris

7.43. Opcodes/Loris/src/Breakpoint.h File Reference

Namespaces

- namespace Loris

7.44. Opcodes/Loris/src/BreakpointEnvelope.h File Reference

```
#include "Envelope.h"
#include <map>
```

Namespaces

- namespace Loris

7.45. Opcodes/Loris/src/BreakpointUtils.h File Reference

```
#include "Breakpoint.h"
#include <algorithm>
#include <functional>
```

Namespaces

- namespace Loris
- namespace Loris::BreakpointUtils

Functions

- void addNoiseEnergy (Breakpoint &bp, double enoise)
- Breakpoint makeNullBefore (const Breakpoint &bp, double fadeTime)
- Breakpoint makeNullAfter (const Breakpoint &bp, double fadeTime)

7.45.1. Function Documentation

7.45.1.1. void addNoiseEnergy (Breakpoint & *bp*, double *enoise*) [inline]

Definition at line 78 of file BreakpointUtils.h.

7.45.1.2. Breakpoint makeNullAfter (const Breakpoint & *bp*, double *fadeTime*)

7.45.1.3. Breakpoint makeNullBefore (const Breakpoint & *bp*, double *fadeTime*)

7.46. Opcodes/Loris/src/Channelizer.h File Reference

```
#include <memory>
```

Namespaces

- namespace Loris

7.47. Opcodes/Loris/src/Dilator.h File Reference

```
#include <vector>
```

Namespaces

- namespace Loris

7.48. Opcodes/Loris/src/Distiller.h File Reference

```
#include "Partial.h"
#include "PartialList.h"
#include "PartialUtils.h"
#include <algorithm>
```

Namespaces

- namespace Loris

7.49. Opcodes/Loris/src/Endian.h File Reference

`#include <iosfwd>`

Namespaces

- namespace Loris

7.50. Opcodes/Loris/src/Envelope.h File Reference

Namespaces

- namespace Loris

7.51. Opcodes/stk/include/Envelope.h File Reference

```
#include "Generator.h"
```

7.52. Opcodes/Loris/src/Exception.h File Reference

```
#include <stdexcept>
```

```
#include <string>
```

Namespaces

- namespace Loris

Defines

- #define __STR(x) __VAL(x)
- #define __VAL(x) #x
- #define Throw(exType, report) throw exType(report, " (" __FILE__ " line: " __STR(__LINE__) ")")
- #define Assert(test)

7.52.1. Define Documentation

7.52.1.1. #define __STR(x) __VAL(x)

Definition at line 291 of file Exception.h.

7.52.1.2. #define __VAL(x) #x

Definition at line 292 of file Exception.h.

7.52.1.3. #define Assert(test)

Value:

```
do {                                                                \
        if (!(test)) Throw( Loris::AssertionFailure, #test );           \
    } while (false)
```

Definition at line 296 of file Exception.h.

7.52.1.4. #define Throw(exType, report) throw exType(report, " (" __FILE__ " line: " __STR(__LINE__) ")")

Definition at line 293 of file Exception.h.

Referenced by Loris::Filter::Filter().

7.53. Opcodes/Loris/src/Filter.h File Reference

```
#include "Exception.h"
#include "Notifier.h"
#include <algorithm>
#include <deque>
#include <functional>
#include <numeric>
#include <vector>
```

Namespaces

- namespace Loris

7.54. Opcodes/stk/include/Filter.h File Reference

```
#include "Stk.h"
#include <vector>
#include <valarray>
```

7.55. Opcodes/Loris/src/FourierTransform.h File Reference

```
#include <complex>
#include <vector>
```

Namespaces

- namespace Loris

7.56. Opcodes/Loris/src/FrequencyReference.h File Reference

```
#include "Envelope.h"
#include "PartialList.h"
#include <memory>
```

Namespaces

- namespace Loris

7.57. Opcodes/Loris/src/ImportLemur.h File Reference

```
#include "PartialList.h"
#include "Exception.h"
#include <string>
```

Namespaces

- namespace Loris

7.58. Opcodes/Loris/src/KaiserWindow.h File Reference

```
#include <vector>
```

Namespaces

- namespace Loris

7.59. Opcodes/Loris/src/Marker.h File Reference

```
#include <functional>
#include <string>
#include <vector>
```

Namespaces

- namespace Loris

7.60. Opcodes/Loris/src/Morpher.h File Reference

```
#include "PartialList.h"
#include "Partial.h"
#include <memory>
```

Namespaces

- namespace Loris

7.61. Opcodes/Loris/src/NoiseGenerator.h File Reference

```
#include "Filter.h"
```

Namespaces

- namespace Loris

7.62. Opcodes/Loris/src/Notifier.h File Reference

Typedefs

- typedef void(* NotificationHandler)(const char *s)

Functions

- NotificationHandler setNotifierHandler (NotificationHandler fn)
- NotificationHandler setDebuggerHandler (NotificationHandler fn)

7.62.1. Typedef Documentation

7.62.1.1. typedef void(* NotificationHandler)(const char *s)

Definition at line 108 of file Notifier.h.

7.62.2. Function Documentation

7.62.2.1. NotificationHandler setDebuggerHandler (NotificationHandler *fn*)

7.62.2.2. NotificationHandler setNotifierHandler (NotificationHandler *fn*)

7.63. Opcodes/Loris/src/Oscillator.h File Reference

```
#include "NoiseGenerator.h"
#include <vector>
```

Namespaces

- namespace Loris

7.64. Opcodes/Loris/src/Partial.h File Reference

```
#include "Breakpoint.h"
#include "Exception.h"
#include <map>
```

Namespaces

- namespace Loris

7.65. Opcodes/Loris/src/PartialBuilder.h File Reference

```
#include "Partial.h"
#include "PartialList.h"
#include "PartialPtrs.h"
#include "SpectralPeaks.h"
```

Namespaces

- namespace Loris

7.66. Opcodes/Loris/src/PartialList.h File Reference

```
#include "Partial.h"
#include <list>
```

Namespaces

- namespace Loris

Typedefs

- typedef std::list< Loris::Partial > PartialList
- typedef std::list< Loris::Partial >::iterator PartialListIterator
- typedef std::list< Loris::Partial >::const_iterator PartialListConstIterator

7.66.1. Typedef Documentation

7.66.1.1. typedef std::list< Loris::Partial > Loris::PartialList

Definition at line 55 of file PartialList.h.

7.66.1.2. typedef std::list< Loris::Partial >::const_iterator Loris::PartialListConstIterator

Definition at line 57 of file PartialList.h.

7.66.1.3. typedef std::list< Loris::Partial >::iterator Loris::PartialListIterator

Definition at line 56 of file PartialList.h.

7.67. Opcodes/Loris/src/PartialPtrs.h File Reference

```
#include "Partial.h"
#include <iterator>
#include <vector>
```

Namespaces

- namespace Loris

Typedefs

- typedef std::vector< Partial * > PartialPtrs
- typedef std::vector< Partial * >::iterator PartialPtrsIterator
- typedef std::vector< Partial * >::const_iterator PartialPtrsConstIterator
- typedef std::vector< const Partial * > ConstPartialPtrs
- typedef std::vector< const Partial * >::iterator ConstPartialPtrsIterator
- typedef std::vector< const Partial * >::const_iterator ConstPartialPtrsConstIterator

Functions

- template<typename Iter> void fillPartialPtrs (Iter begin, Iter end, PartialPtrs &fillme)
- template<typename Iter> void fillPartialPtrs (Iter begin, Iter end, ConstPartialPtrs &fillme)

7.67.1. Typedef Documentation

7.67.1.1. typedef std::vector< const Partial * > Loris::ConstPartialPtrs

Definition at line 72 of file PartialPtrs.h.

7.67.1.2. typedef std::vector< const Partial * >::const_iterator Loris::ConstPartialPtrsConstIterator

Definition at line 74 of file PartialPtrs.h.

7.67.1.3. typedef std::vector< const Partial * >::iterator Loris::ConstPartialPtrsIterator

Definition at line 73 of file PartialPtrs.h.

7.67.1.4. typedef std::vector< Partial * > Loris::PartialPtrs

Definition at line 68 of file PartialPtrs.h.

7.67.1.5. typedef std::vector< Partial * >::const_iterator Loris::PartialPtrsConstIterator

Definition at line 70 of file PartialPtrs.h.

7.67.1.6. typedef std::vector< Partial * >::iterator Loris::PartialPtrsIterator

Definition at line 69 of file PartialPtrs.h.

7.67.2. Function Documentation

7.67.2.1. template<typename Iter> void fillPartialPtrs (Iter *begin*, Iter *end*, ConstPartialPtrs & *fillme*)

Definition at line 96 of file PartialPtrs.h.

7.67.2.2. template<typename Iter> void fillPartialPtrs (Iter *begin*, Iter *end*, PartialPtrs & *fillme*)

Definition at line 87 of file PartialPtrs.h.

Referenced by Loris::Sieve::sift().

7.68. Opcodes/Loris/src/PartialUtils.h File Reference

```
#include "Envelope.h"
#include "Partial.h"
#include <functional>
#include <utility>
```

Namespaces

- namespace Loris
- namespace Loris::PartialUtils

Functions

- template<class Arg> void scaleAmplitude (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleAmplitude (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleBandwidth (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleBandwidth (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleFrequency (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleFrequency (Iter b, Iter e, const Arg &arg)
- template<class Arg> void scaleNoiseRatio (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void scaleNoiseRatio (Iter b, Iter e, const Arg &arg)
- template<class Arg> void shiftPitch (Partial &p, const Arg &arg)
- template<class Iter, class Arg> void shiftPitch (Iter b, Iter e, const Arg &arg)
- void crop (Partial &p, double t1, double t2)
- template<class Iter> void crop (Iter b, Iter e, double t1, double t2)
- void shiftTime (Partial &p, double arg)
- template<class Iter> void shiftTime (Iter b, Iter e, double arg)
- template<typename Iterator> std::pair< double, double > timeSpan (Iterator begin, Iterator end)

7.68.1. Function Documentation

7.68.1.1. template<class Iter> void crop (Iter *b*, Iter *e*, double *t1*, double *t2*)

Definition at line 346 of file PartialUtils.h.

7.68.1.2. void crop (Partial & *p*, double *t1*, double *t2*) [inline]

Definition at line 339 of file PartialUtils.h.

7.68.1.3. template<class Iter, class Arg> void scaleAmplitude (Iter *b*, Iter *e*, const Arg & *arg*)

Scale the amplitude of a sequence of Partials according to an envelope representing a amplitude scale value or envelope.

Parameters:

b is the beginning of a sequence of Partials to mutate.

e is the end of a sequence of Partials to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 148 of file PartialUtils.h.

7.68.1.4. template<class Arg> void scaleAmplitude (Partial & *p*, const Arg & *arg*)

Scale the amplitude of the specified Partial according to an envelope representing a amplitude scale value or envelope.

Parameters:

p is a Partial to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 130 of file PartialUtils.h.

7.68.1.5. template<class Iter, class Arg> void scaleBandwidth (Iter *b*, Iter *e*, const Arg & *arg*)

Scale the bandwidth of a sequence of Partials according to an envelope representing a amplitude scale value or envelope.

Parameters:

b is the beginning of a sequence of Partials to mutate.

e is the end of a sequence of Partials to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 208 of file PartialUtils.h.

7.68.1.6. template<class Arg> void scaleBandwidth (Partial & *p*, const Arg & *arg*)

Scale the bandwidth of the specified Partial according to an envelope representing a amplitude scale value or envelope.

Parameters:

p is a Partial to mutate.

arg is either a constant scale factor or an Envelope describing the time-varying scale factor.

Definition at line 190 of file PartialUtils.h.

7.68.1.7. template<class Iter, class Arg> void scaleFrequency (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 240 of file PartialUtils.h.

7.68.1.8. template<class Arg> void scaleFrequency (Partial & *p*, const Arg & *arg*)

Definition at line 233 of file PartialUtils.h.

7.68.1.9. template<class Iter, class Arg> void scaleNoiseRatio (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 273 of file PartialUtils.h.

7.68.1.10. template<class Arg> void scaleNoiseRatio (Partial & *p*, const Arg & *arg*)

Definition at line 266 of file PartialUtils.h.

7.68.1.11. template<class Iter, class Arg> void shiftPitch (Iter *b*, Iter *e*, const Arg & *arg*)

Definition at line 306 of file PartialUtils.h.

7.68.1.12. template<class Arg> void shiftPitch (Partial & *p*, const Arg & *arg*)

Definition at line 299 of file PartialUtils.h.

7.68.1.13. template<class Iter> void shiftTime (Iter *b*, Iter *e*, double *arg*)

Definition at line 380 of file PartialUtils.h.

7.68.1.14. void shiftTime (Partial & *p*, double *arg*) [inline]

Definition at line 373 of file PartialUtils.h.

7.68.1.15. template<typename Iterator> std::pair< double, double > timeSpan (Iterator *begin*, Iterator *end*)

Definition at line 396 of file PartialUtils.h.

7.69. Opcodes/Loris/src/ReassignedSpectrum.h File Reference

```
#include "FourierTransform.h"
#include <vector>
```

Namespaces

- namespace Loris

7.70. Opcodes/Loris/src/Resampler.h File Reference

```
#include "PartialList.h"
```

Namespaces

- namespace Loris

7.71. Opcodes/Loris/src/SdifFile.h File Reference

```
#include "Marker.h"
#include "Partial.h"
#include "PartialList.h"
#include <string>
#include <vector>
```

Namespaces

- namespace Loris

7.72. Opcodes/Loris/src/Sieve.h File Reference

```
#include "PartialPtrs.h"
```

Namespaces

- namespace Loris

7.73. Opcodes/Loris/src/SpcFile.h File Reference

```
#include "Marker.h"
#include "Partial.h"
#include <string>
#include <vector>
```

Namespaces

- namespace Loris

7.74. Opcodes/Loris/src/SpectralPeaks.h File Reference

```
#include <utility>
#include <vector>
#include "Breakpoint.h"
```

Namespaces

- namespace Loris

Typedefs

- typedef std::vector< std::pair< double, Breakpoint > > Peaks

7.74.1. Typedef Documentation

7.74.1.1. typedef std::vector< std::pair< double, Breakpoint > > Loris::Peaks

Definition at line 45 of file SpectralPeaks.h.

7.75. Opcodes/Loris/src/SpectralPeakSelector.h File Reference

```
#include "Partial.h"
#include "SpectralPeaks.h"
```

Namespaces

- namespace Loris

7.76. Opcodes/Loris/src/Synthesizer.h File Reference

```
#include "Oscillator.h"
#include "PartialList.h"
#include "PartialUtils.h"
#include <vector>
```

Namespaces

- namespace Loris