LASS Reference Manual

1.0

Generated by Doxygen 1.3.4

Mon Apr 25 13:18:11 2005

Contents

Chapter 1

LASS Reference Manual

1.1 Introduction

LASS was written to provide musicians an environment for performing additive sound synthesis. It is unique from other systems in the way that it allows musicians to specify how 'loud' a sound should be heard. LASS then adjusts the sounds to the correct amplitude via a method called critical bands. The three main design goals for the project are expandability, ease of use, and efficiency. LASS is designed with a very modular architecture. No doubt, there will be features that need to be added - and future generations of students must be able to easily expand the system. The system was also designed to be easy for users. The interface to the classes were made as clear as possible and kept consistent across objects. Also, I've made extensive use of references instead of pointers to help ensure good memory management. Finally, LASS must also be efficient, for sound synthesis requires much calculation. This tool may not be quite as efficient as M4C, but tradeoffs in features and ease of use make it worth the while.

1.2 More Documentation

for a brief tutorial on LASS, see tutorial.pdf

Chapter 2

LASS Namespace Index

2.1 LASS Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

LASS Hierarchical Index

3.1 LASS Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

LASS Class Index

4.1 LASS Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

LASS File Index

5.1 LASS File List

Here is a list of all files with brief descriptions:

Chapter 6

LASS Page Index

6.1 LASS Related Pages

Here is a list of all related documentation pages:

Chapter 7

LASS Namespace Documentation

7.1 std Namespace Reference

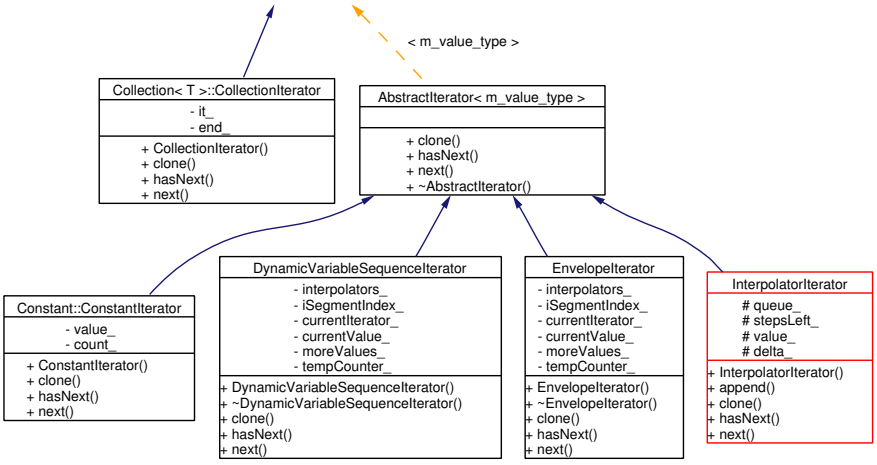
Chapter 8

LASS Class Documentation

8.1 AbstractIterator< T > Class Template Reference

`#include <AbstractIterator.h>`

Inheritance diagram for AbstractIterator< T >:



Public Member Functions

- virtual AbstractIterator< T > * clone ()=0
- virtual bool hasNext ()=0
- virtual T & next ()=0
- virtual ~AbstractIterator ()

8.1.1 Detailed Description

template<class T> class AbstractIterator< T >

This is a templated abstract definition of the most basic iterator.

Author:
Braden Kowitz

Definition at line 33 of file AbstractIterator.h.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 template<class T> virtual AbstractIterator< T >::~AbstractIterator () [inline, virtual]

This is the destructor for the iterator.

Definition at line 56 of file AbstractIterator.h.

8.1.3 Member Function Documentation

8.1.3.1 template<class T> virtual AbstractIterator<T>* AbstractIterator< T >::clone () [pure virtual]

Returns:
An exact copy of this iterator

Implemented in Collection< T >::CollectionIterator, Constant::ConstantIterator, DynamicVariableSequenceIterator, EnvelopeIterator, InterpolatorIterator, LinearInterpolatorIterator, ExponentialInterpolatorIterator, and CubicSplineInterpolatorIterator.

8.1.3.2 template<class T> virtual bool AbstractIterator< T >::hasNext () [pure virtual]

Return values:

true If the iterator has another value

false If the iterator does not have another value

Implemented in Collection< T >::CollectionIterator, Constant::ConstantIterator, DynamicVariableSequenceIterator, EnvelopeIterator, and InterpolatorIterator.

8.1.3.3 template<class T> virtual T& AbstractIterator< T >::next () [pure virtual]

Returns:

The next value in the iterator as a reference

Implemented in Collection< T >::CollectionIterator, Constant::ConstantIterator, DynamicVariableSequenceIterator, EnvelopeIterator, InterpolatorIterator, LinearInterpolatorIterator, ExponentialInterpolatorIterator, and CubicSplineInterpolatorIterator.

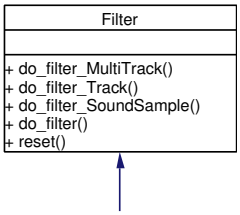
The documentation for this class was generated from the following file:

- AbstractIterator.h

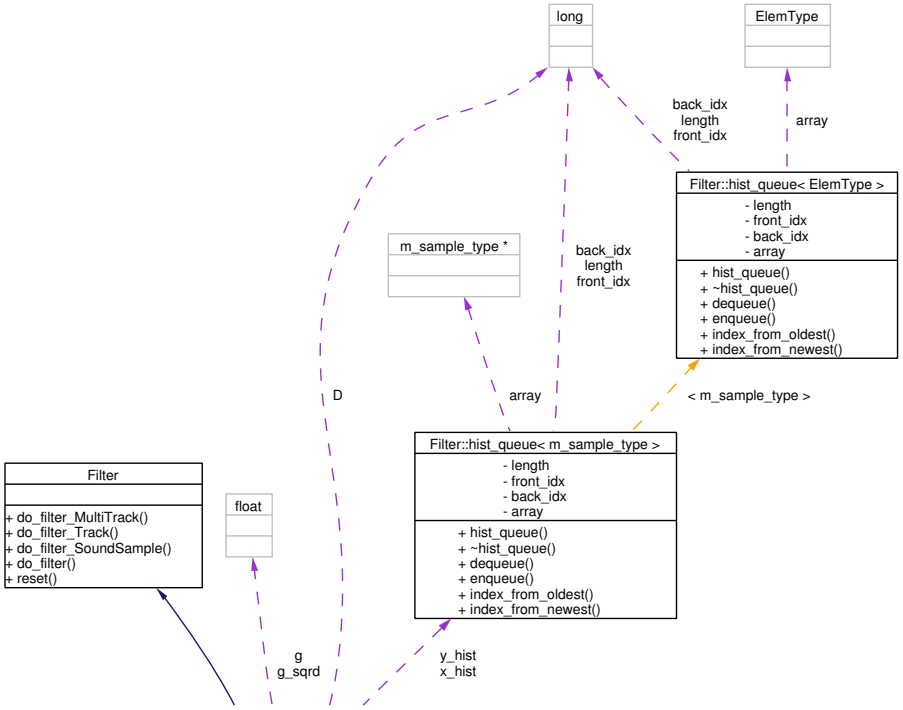
8.2 AllPassFilter Class Reference

`#include <AllPassFilter.h>`

Inheritance diagram for AllPassFilter:



Collaboration diagram for AllPassFilter:



Public Member Functions

- AllPassFilter (float gain, long delay)
- ~AllPassFilter ()
- m_sample_type do_filter (m_sample_type x_t)
- void reset (void)
- void xml_print (ofstream &xmlOutput)
- AllPassFilter ()
- void set_g (float gain)
- void set_D (long delay)
- void xml_read (XmlReader::xmltag *apftag)

Private Attributes

- float g
- float g_sqrd
- long D
- Filter::hist_queue< m_sample_type > * y_hist
- Filter::hist_queue< m_sample_type > * x_hist

8.2.1 Detailed Description

The comb filter class implements an allpass filter as described on page 385 of 'Elements of Computer Music:

y(t) = -gx(t) + (1-g^2)*(x(t-D) + g*y(t-D))

Note:

This filter (and all other 'class Filter's) are stateful machines with internal feedback mechanisms. Thus this filter should be reset() each time you begin a new channel and should not be mixed between channels.

Author:

Andrew Kurtz
Jim Lindstrom

Definition at line 51 of file AllPassFilter.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 AllPassFilter::AllPassFilter (float *gain*, long *delay*)

This is the constructor.

Parameters:

gain The feedback gain (should be in the range 0.0 to 1.0)

delay The allpass delay (in units of samples)

Definition at line 43 of file AllPassFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), g, g_sqrd, x_hist, and y_hist.

8.2.2.2 AllPassFilter::~AllPassFilter ()

This is the destructor

Definition at line 62 of file AllPassFilter.cpp.

References x_hist, and y_hist.

8.2.2.3 AllPassFilter::AllPassFilter ()

This is a constructor.

Definition at line 112 of file AllPassFilter.cpp.

8.2.3 Member Function Documentation

8.2.3.1 m_sample_type AllPassFilter::do_filter (m_sample_type *x_t*) [virtual]

This method applies an allpass filter to a single sample

Parameters:
x_t The input sample

Returns:
The sampled filter

Implements Filter.

Definition at line 70 of file AllPassFilter.cpp.

References Filter::hist_queue< m_sample_type >::dequeue(), Filter::hist_queue< m_sample_type >::enqueue(), g, g_sqrd, m_sample_type, x_hist, and y_hist.

Referenced by Reverb::do_reverb().

8.2.3.2 void AllPassFilter::reset (void) [virtual]

This method should be redefined by each class derived from Filter to reset the filter to an initial state. It should have the same effect as deleting the filter and creating a new one.

Implements Filter.

Definition at line 82 of file AllPassFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), x_hist, and y_hist.

Referenced by Reverb::reset().

8.2.3.3 void AllPassFilter::set_D (long *delay*)

This sets the delay.

Parameters:
delay The delay as a long

Definition at line 121 of file AllPassFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), x_hist, and y_hist.

Referenced by xml_read().

8.2.3.4 void AllPassFilter::set_g (float *gain*)

This sets the gain.

Parameters:
gain The gain as a float

Definition at line 116 of file AllPassFilter.cpp.

References g, and g_sqrd.

Referenced by xml_read().

8.2.3.5 void AllPassFilter::xml_print (ofstream & *xmlOutput*)

Deprecated
This outputs an XML representation of the object to STDOUT.

Parameters:
xmlOutput The output channel

Definition at line 100 of file AllPassFilter.cpp.

References D, and g.

Referenced by Reverb::xml_print().

8.2.3.6 void AllPassFilter::xml_read (XmlReader::xmltag * *apftag*)

Deprecated
Reads some xml and sets the gain and delay.

Parameters:
apftag A pointer to the xml to read

Definition at line 136 of file AllPassFilter.cpp.

References XmlReader::xmltag::findChildParamValue(), set_D(), and set_g().

Referenced by Reverb::xml_read().

8.2.4 Member Data Documentation

8.2.4.1 long AllPassFilter::D [private]

The delay for the comb component of the filter

Definition at line 124 of file AllPassFilter.h.

Referenced by AllPassFilter(), reset(), set_D(), and xml_print().

8.2.4.2 float AllPassFilter::g [private]

The gain for the comb component of the filter

Definition at line 114 of file AllPassFilter.h.

Referenced by AllPassFilter(), do_filter(), set_g(), and xml_print().

8.2.4.3 float AllPassFilter::g_sqrd [private]

The square of the gain

Definition at line 119 of file AllPassFilter.h.

Referenced by AllPassFilter(), do_filter(), and set_g().

8.2.4.4 Filter::hist_queue<m_sample_type>* AllPassFilter::x_hist [private]

This queue holds past samples to implement the delay

Definition at line 134 of file AllPassFilter.h.

Referenced by AllPassFilter(), do_filter(), reset(), set_D(), and ~AllPassFilter().

8.2.4.5 Filter::hist_queue<m_sample_type>* AllPassFilter::y_hist [private]

This queue holds past samples to implement the delay

Definition at line 129 of file AllPassFilter.h.

Referenced by AllPassFilter(), do_filter(), reset(), set_D(), and ~AllPassFilter().

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

- AllPassFilter.h
- AllPassFilter.cpp

8.3 AuWriter Class Reference

`#include <AuWriter.h>`

Static Public Member Functions

- bool write (SoundSample &ss, string filename)
- bool write (Track &t, string filename)
- bool write (MultiTrack &mt, string filename)
- bool write_one_per_track (MultiTrack &mt, char *filename,...)

Static Private Member Functions

- bool write (vector< SoundSample * > &channels, string filename)
- void WriteIntMsb (ostream &out, long l, int size)

8.3.1 Detailed Description

Writes out SoundSample, Track, and MultiTrack objects to an AU file. All files are written as 16 bits.

Author:
Braden Kowitz

Definition at line 46 of file AuWriter.h.

8.3.2 Member Function Documentation

8.3.2.1 bool AuWriter::write (vector< SoundSample * > & *channels*, string *filename*) [static, private]

This write the channels of a SoundSample out as an AU file

Parameters:
channels A vector of SoundSamples to write out

filename The name of the file to write out to

Return values:
true On success

false On failure

Definition at line 89 of file AuWriter.cpp.

References m_rate_type, m_sample_count_type, m_sample_type, m_time_type, and WriteIntMsb().

8.3.2.2 bool AuWriter::write (MultiTrack & *mt*, string *filename*) [static]

This writes a MultiTrack object out as an AU file.

Note:
Some systems may not correctly read AU files with more than 2 tracks. Try converting the file to another format if you have issues (tool: SOX, sound exchange)

Parameters:
mt The MultiTrack to write out

filename The name of the file to write out to

Return values:
true On success

false On failure

Definition at line 52 of file AuWriter.cpp.

References Iterator< T >::hasNext(), Collection< Track * >::iterator(), Iterator< T >::next(), and write().

8.3.2.3 bool AuWriter::write (Track & *t*, string *filename*) [static]

This writes a Track object out as an AU file.

Parameters:
t The Track to write out

filename The name of the file to write out to

Return values:
true On success

false On failure

Definition at line 43 of file AuWriter.cpp.

References Track::getWave(), and write().

8.3.2.4 bool AuWriter::write (SoundSample & *ss*, string *filename*) [static]

This writes a SoundSample object out as an AU file.

Parameters:

ss The SoundSample to write out

filename The name of the file to write out to

Return values:

true On success

false On failure

Definition at line 35 of file AuWriter.cpp.

Referenced by write(), and write_one_per_track().

8.3.2.5 bool AuWriter::write_one_per_track (MultiTrack & *mt*, char * *filename*, ...) [static]

This writes the individual Tracks in a MultiTrack to a set of filenames specified.

Parameters:

mt The MultiTrack to write out

filename A pointer to an array holding the names of the files to write out to

Return values:

true On success

false On failure

Definition at line 66 of file AuWriter.cpp.

References Iterator< T >::hasNext(), Collection< Track * >::iterator(), Iterator< T >::next(), and write().

8.3.2.6 void AuWriter::WriteIntMsb (ostream & *out*, long *l*, int *size*) [inline, static, private]

Todo

Recursive inlining? This is insane. This needs to be reworked.

Definition at line 218 of file AuWriter.cpp.

Referenced by write().

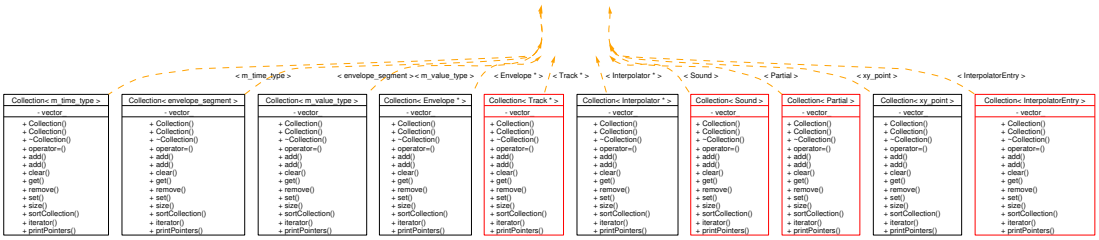
The documentation for this class was generated from the following files:

- AuWriter.h
- AuWriter.cpp

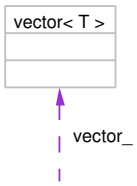
8.4 Collection< T > Class Template Reference

`#include <Collection.h>`

Inheritance diagram for Collection< T >:



Collaboration diagram for Collection< T >:



Public Member Functions

- Collection ()
- Collection (const Collection &c)
- virtual ~Collection ()

- Collection & operator= (const Collection &c)
- void add (const T &element)
- void add (int index, const T &element)
- void clear ()
- T & get (int index)
- T remove (int index)
- T set (int index, const T &element)
- int size ()
- void sortCollection ()
- Iterator< T > iterator ()
- void printPointers (char *collType)

Private Attributes

- vector< T > vector_

8.4.1 Detailed Description

template<class T> class Collection< T >

Represents a collection of objects. They are kept in order, with consecutive index numbers running from 0 to (size-1). Each entry is also accompanied by a string name. Note: This class is heavily modeled from Java Collection and Vector

Author:
Braden Kowitz

Definition at line 49 of file Collection.h.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 template<class T> Collection< T >::Collection ()

This is the default constructor, which creates an empty collection.

Definition at line 42 of file Collection.cpp.

8.4.2.2 template<class T> Collection< T >::Collection (const Collection< T > & *c*)

This is a copy constructor.

Parameters:
c The Collection from which to make a copy

Definition at line 47 of file Collection.cpp.

References Collection< T >::vector_.

8.4.2.3 template<class T> Collection< T >::~Collection () [virtual]

This is the destructor.

Definition at line 55 of file Collection.cpp.

8.4.3 Member Function Documentation

8.4.3.1 template<class T> void Collection< T >::add (int *index*, const T & *element*)

Inserts the given element at the specified index. The element at that index, and subsequent elements are shifted to the right.

Note:
This will make a COPY of the element in the Collection.

Exceptions:
IndexOutOfBoundsException

Parameters:
index An integer specifying where to insert the element

element The element to insert

Definition at line 96 of file Collection.cpp.

References Collection< T >::vector_.

8.4.3.2 template<class T> void Collection< T >::add (const T & *element*)

Appends the given element to the end of the Collection.

Note:
This will make a COPY of the element in the Collection.

Parameters:
element An element to add to the collection

Definition at line 85 of file Collection.cpp.

References Collection< T >::vector_.

Referenced by Envelope::Envelope(), Envelope::getPoints(), EnvelopeLibrary::loadLibrary(), Envelope::multiply(), Collection< T >::operator=(), Score::render(), and Envelope::xml_read().

8.4.3.3 template<class T> void Collection< T >::clear ()

Clears every element from the collection.

Definition at line 113 of file Collection.cpp.

References Collection< T >::vector_.

Referenced by EnvelopeLibrary::loadLibrary(), and Collection< T >::operator=().

8.4.3.4 template<class T> T & Collection< T >::get (int *index*)

Exceptions:
IndexOutOfBoundsException

Parameters:
index An integer specifying which element to retrieve

Returns:
A reference to the element at the specified index

Definition at line 119 of file Collection.cpp.

References Collection< T >::vector_.

Referenced by Envelope::DefineShape(), Envelope::Envelope(), Envelope::multiply(), and Score::render().

8.4.3.5 template<class T> Iterator< T > Collection< T >::iterator ()

Returns an iterator over the elements in this collection.

Returns:
an iteration

Definition at line 182 of file Collection.cpp.

References Collection< T >::vector_.

8.4.3.6 template<class T> Collection< T > & Collection< T >::operator= (const Collection< T > & *c*)

This assigns one Collection to another.

Parameters:
c The Collection to assign

Definition at line 61 of file Collection.cpp.

References Collection< T >::add(), Collection< T >::clear(), and Collection< T >::vector_.

8.4.3.7 template<class T> void Collection< T >::printPointers (char * *collType*)

Definition at line 220 of file Collection.cpp.

References Collection< T >::vector_.

8.4.3.8 template<class T> T Collection< T >::remove (int *index*)

Removes the element at the specified index. All other elements are shifted left to fill the hole.

Exceptions:
IndexOutOfBoundsException

Parameters:
index An integer specifying which element to delete

Returns:
A reference to the element at the specified index

Definition at line 133 of file Collection.cpp.

References Collection< T >::vector_.

8.4.3.9 template<class T> T Collection< T >::set (int *index*, const T & *element*)

Replaces the element at the specified index.

Exceptions:
IndexOutOfBoundsException

Parameters:

index An integer specifying which element to replace

element An element to replace the old one

Returns:

A reference to the element that was replaced

Definition at line 153 of file Collection.cpp.

References Collection< T >::vector_.

Referenced by Envelope::DefineShape().

8.4.3.10 template<class T> int Collection< T >::size ()

Returns the size of this collection. This is a 0-based array so the entries are indexed from 0 to (size-1).

Returns:

The size of the collection

Definition at line 169 of file Collection.cpp.

References Collection< T >::vector_.

Referenced by Envelope::DefineShape(), DynamicVariableSequenceIterator::DynamicVariableSequenceIterator(), Envelope::Envelope(), and EnvelopeIterator::EnvelopeIterator().

8.4.3.11 template<class T> void Collection< T >::sortCollection ()

This sorts the collection.

Note:

only use this function if template type of this collection has well defined operator< , operator== , and operator> . Examples of this are standard types like int and float, as well as custom types like LinearInterpolatorEntry.

Definition at line 176 of file Collection.cpp.

References Collection< T >::vector_.

8.4.4 Member Data Documentation

8.4.4.1 template<class T> vector<T> Collection< T >::vector_ [private]

Definition at line 151 of file Collection.h.

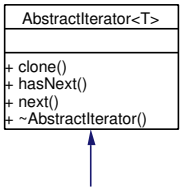
Referenced by Collection< T >::add(), Collection< T >::clear(), Collection< T >::Collection(), Collection< T >::get(), Collection< T >::iterator(), Collection< T >::operator=(), Collection< T >::printPointers(), Collection< T >::remove(), Collection< T >::set(), Collection< T >::size(), and Collection< T >::sortCollection().

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

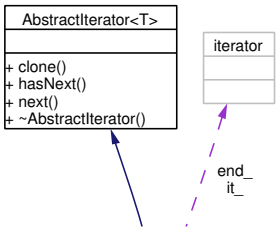
- Collection.h
- Collection.cpp

8.5 Collection< T >::CollectionIterator Class Reference

Inheritance diagram for Collection< T >::CollectionIterator:



Collaboration diagram for Collection< T >::CollectionIterator:



Public Member Functions

- CollectionIterator (typename vector< T >::iterator it, typename vector< T >::iterator end)
- CollectionIterator * clone ()
- bool hasNext ()
- T & next ()

Private Attributes

- vector< T >::iterator it_
- vector< T >::iterator end_

8.5.1 Detailed Description

template<class T> class Collection< T >::CollectionIterator

An abstract iterator that makes it possible to iterate over all values in a collection.

Definition at line 157 of file Collection.h.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 template<class T> Collection< T >::CollectionIterator::CollectionIterator (typename vector< T >::iterator *it*, typename vector< T >::iterator *end*)

Constructor that takes two STL vector iterators over which the iterator will iterate.

Definition at line 194 of file Collection.cpp.

Referenced by Collection< T >::CollectionIterator::clone().

8.5.3 Member Function Documentation

8.5.3.1 template<class T> Collection< T >::CollectionIterator * Collection< T >::CollectionIterator::clone () [virtual]

Creates an exact copy of this iterator.

Returns:
exact copy of this iterator

Implements AbstractIterator< T >.

Definition at line 200 of file Collection.cpp.

References Collection< T >::CollectionIterator::CollectionIterator(), Collection< T >::CollectionIterator::end_, and Collection< T >::CollectionIterator::it_.

8.5.3.2 template<class T> bool Collection< T >::CollectionIterator::hasNext () [virtual]

Indicates whether the iterator has another value.

Returns:
true If iterator has another value
false If iterator does not have another value

Implements AbstractIterator< T >.

Definition at line 206 of file Collection.cpp.

References Collection< T >::CollectionIterator::end_, and Collection< T >::CollectionIterator::it_.

8.5.3.3 template<class T> T & Collection< T >::CollectionIterator::next () [virtual]

Gets the next value for the iterator.

Returns:
the next iterator

Todo
What to do when out of bounds?

Implements AbstractIterator< T >.

Definition at line 212 of file Collection.cpp.

References Collection< T >::CollectionIterator::it_.

8.5.4 Member Data Documentation

8.5.4.1 template<class T> vector<T>::iterator Collection< T >::CollectionIterator::end_ [private]

The end position, so this iterator knows when to stop.

Definition at line 197 of file Collection.h.

Referenced by Collection< T >::CollectionIterator::clone(), and Collection< T >::CollectionIterator::hasNext().

8.5.4.2 template<class T> vector<T>::iterator Collection< T >::CollectionIterator::it_ [private]

Iterator which this class wraps.

Definition at line 192 of file Collection.h.

Referenced by Collection< T >::CollectionIterator::clone(), Collection< T >::CollectionIterator::hasNext(), and Collection< T >::CollectionIterator::next().

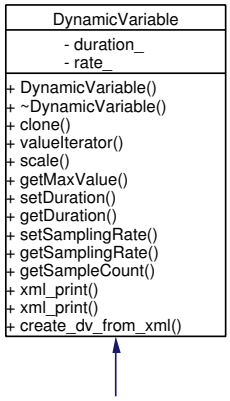
The documentation for this class was generated from the following files:

- Collection.h
- Collection.cpp

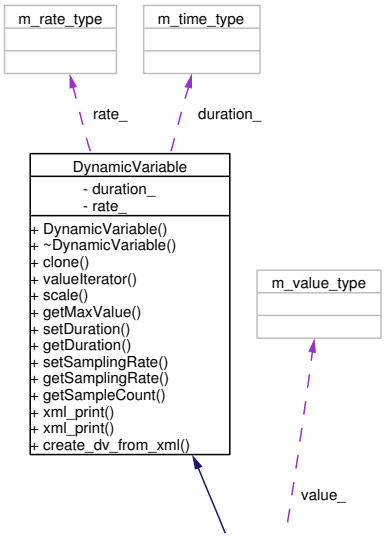
8.6 Constant Class Reference

`#include <Constant.h>`

Inheritance diagram for Constant:



Collaboration diagram for Constant:



Public Member Functions

- Constant (m_value_type value=0)
- Constant * clone ()
- void setValue (m_value_type value)
- m_value_type getValue ()
- Iterator< m_value_type > valueIterator ()
- void scale (m_value_type factor)
- m_value_type getMaxValue ()
- void xml_print (ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)
- void xml_print (ofstream &xmlOutput)
- void xml_read (XmlReader::xmltag *xml)

Private Attributes

- m_value_type value_

8.6.1 Detailed Description

A constant Dynamic variable. Kind of an oxymoron, but it's needed.

Author:
Braden Kowitz

Definition at line 46 of file Constant.h.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Constant::Constant (m_value_type *value* = 0)

Constructor that creates a constant at a specified value.

Definition at line 38 of file Constant.cpp.

References m_value_type.

Referenced by clone().

8.6.3 Member Function Documentation

8.6.3.1 Constant * Constant::clone () [virtual]

Creates an exact copy of this object.

Returns:
pointer to the copy

Implements DynamicVariable.

Definition at line 44 of file Constant.cpp.

References Constant().

8.6.3.2 m_value_type Constant::getMaxValue () [virtual]

Simply return the constant value.

Returns:
an m_value_type

Implements DynamicVariable.

Definition at line 75 of file Constant.cpp.

References m_value_type, and value_.

8.6.3.3 m_value_type Constant::getValue ()

Returns the value of this constant.

Returns:
An m_value_type

Definition at line 56 of file Constant.cpp.

References m_value_type, and value_.

8.6.3.4 void Constant::scale (m_value_type *factor*) [virtual]

Scales the constant by this value.

Parameters:
factor an m_value_type

Implements DynamicVariable.

Definition at line 68 of file Constant.cpp.

References m_value_type, and value_.

8.6.3.5 void Constant::setValue (m_value_type *value*)

Sets the value of this constant.

Parameters:
value An m_value_type

Definition at line 50 of file Constant.cpp.

References m_value_type, and value_.

Referenced by xml_read().

8.6.3.6 Iterator< m_value_type > Constant::valueIterator () [virtual]

Returns a Iterator(ConstantIterator) (private sub-class)

Returns:
an Iterator<m_value_type>

Implements DynamicVariable.

Definition at line 62 of file Constant.cpp.

References DynamicVariable::getSampleCount(), and value_.

8.6.3.7 void Constant::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

Implements DynamicVariable.

Definition at line 120 of file Constant.cpp.

References DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), and value_.

8.6.3.8 void Constant::xml_print (ofstream & *xmlOutput*, list< DynamicVariable * > & *dynObjs*) [virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 148 of file Constant.cpp.

8.6.3.9 void Constant::xml_read (XmlReader::xmltag * *xml*)

Deprecated

This sets the duration, sampling rate, and value if they they are contained in the xml.

Parameters:

xml The xml to read

Definition at line 133 of file Constant.cpp.

References XmlReader::xmltag::findChildParamValue(), DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), and setValue().

Referenced by DynamicVariable::create_dv_from_xml().

8.6.4 Member Data Documentation

8.6.4.1 m_value_type Constant::value_ [private]

Definition at line 110 of file Constant.h.

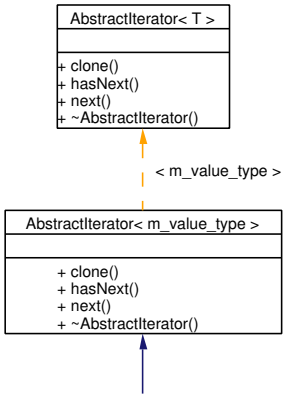
Referenced by getMaxValue(), getValue(), scale(), setValue(), valueIterator(), and xml_print().

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

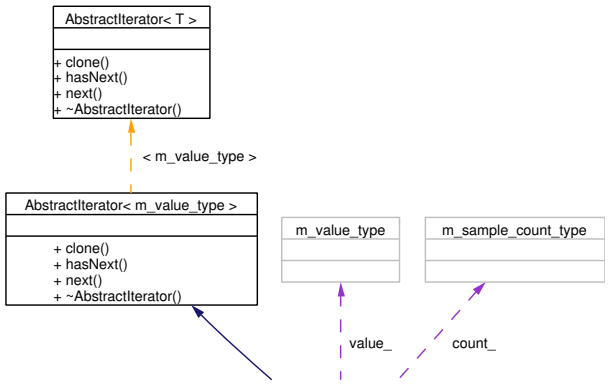
- Constant.h
- Constant.cpp

8.7 Constant::ConstantIterator Class Reference

Inheritance diagram for Constant::ConstantIterator:



Collaboration diagram for Constant::ConstantIterator:



Public Member Functions

- ConstantIterator (m_value_type value, m_sample_count_type count)
- ConstantIterator * clone ()
- bool hasNext ()
- m_value_type & next ()

Private Attributes

- m_value_type value_
- m_sample_count_type count_

8.7.1 Detailed Description

This iterator iterates over a constant value. for a given amount of steps.

Definition at line 116 of file Constant.h.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 Constant::ConstantIterator::ConstantIterator (m_value_type *value*, m_sample_count_type *count*)

This iterator will iterate over 'value' count times.

Parameters:

value An m_value_type that holds the value

count An m_sample_count_type that holds the number of times to iterate

Definition at line 87 of file Constant.cpp.

References m_sample_count_type, and m_value_type.

8.7.3 Member Function Documentation

8.7.3.1 Constant::ConstantIterator * Constant::ConstantIterator::clone () [virtual]

Creates an exact copy of this iterator.

Returns:

Copy of this iterator

Implements AbstractIterator< m_value_type >.

Definition at line 95 of file Constant.cpp.

8.7.3.2 bool Constant::ConstantIterator::hasNext () [inline, virtual]

Indicates whether is another value to iterate over.

Return values:

true If there is another value to iterate over

false If there is not another value to iterate over

Implements AbstractIterator< m_value_type >.

Definition at line 104 of file Constant.cpp.

8.7.3.3 m_value_type & Constant::ConstantIterator::next () [inline, virtual]

Gets the next value in the iteration.

Todo

What to do on an out of bounds error?

Returns:

the next value in the iteration

Implements AbstractIterator< m_value_type >.

Definition at line 112 of file Constant.cpp.

8.7.4 Member Data Documentation

8.7.4.1 m_sample_count_type Constant::ConstantIterator::count_ [private]

Definition at line 149 of file Constant.h.

8.7.4.2 m_value_type Constant::ConstantIterator::value_ [private]

Definition at line 148 of file Constant.h.

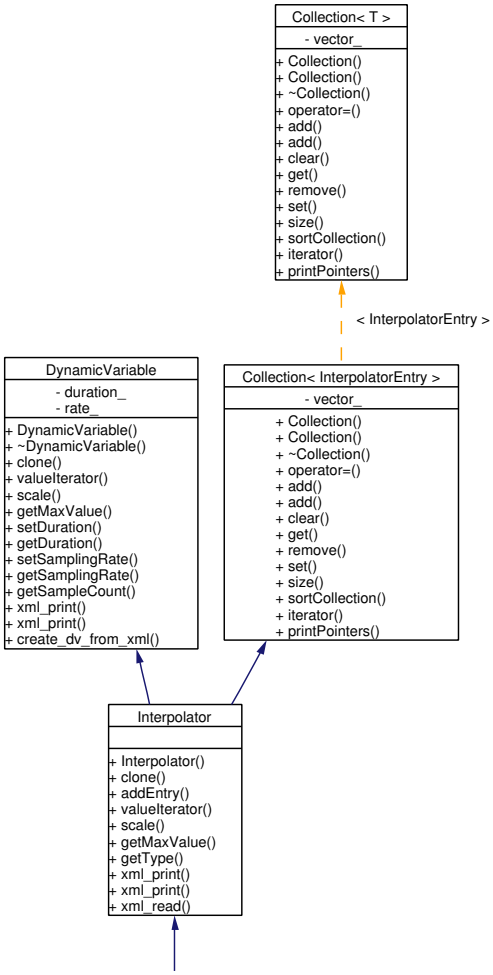
The documentation for this class was generated from the following files:

- Constant.h
- Constant.cpp

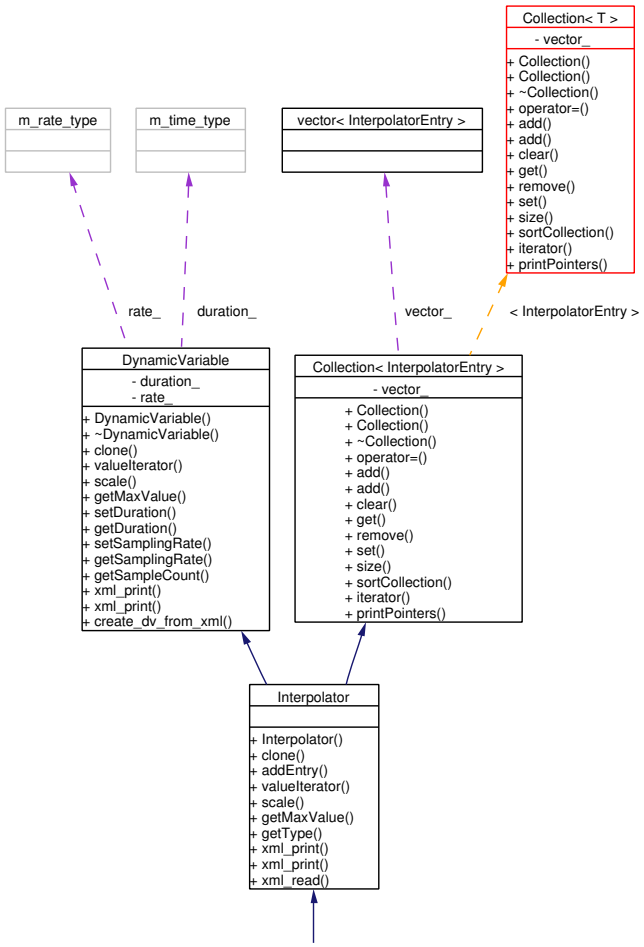
8.8 CubicSplineInterpolator Class Reference

`#include <InterpolatorTypes.h>`

Inheritance diagram for CubicSplineInterpolator:



Collaboration diagram for CubicSplineInterpolator:



Public Member Functions

- CubicSplineInterpolator ()
- CubicSplineInterpolator * clone ()
- Iterator< m_value_type > valueIterator ()
- virtual interpolation_type getType ()

8.8.1 Detailed Description

This is a DynamicVariable that changes over time. This does cubic spline interpolation between a set of points ordered in time.

Author:
Braden Kowitz
Philipp Fraund

Definition at line 117 of file InterpolatorTypes.h.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 CubicSplineInterpolator::CubicSplineInterpolator ()

This is the default constructor.

Definition at line 195 of file InterpolatorTypes.cpp.

Referenced by clone().

8.8.3 Member Function Documentation

8.8.3.1 CubicSplineInterpolator * CubicSplineInterpolator::clone () [virtual]

This makes a clone of a CubicSplineInterpolator.

Returns:
A new CubicSplineInterpolator

Implements Interpolator.

Definition at line 200 of file InterpolatorTypes.cpp.

References CubicSplineInterpolator().

8.8.3.2 interpolation_type CubicSplineInterpolator::getType () [virtual]

This provides an implementation to get the type of interpolator.

Returns:
The interpolation type

Implements Interpolator.

Definition at line 267 of file InterpolatorTypes.cpp.

References CUBIC_SPLINE, and interpolation_type.

8.8.3.3 Iterator< m_value_type > CubicSplineInterpolator::valueIterator () [virtual]

This creates an iterator over CubicSplineInterpolators.

Returns:
An iterator

Implements Interpolator.

Definition at line 209 of file InterpolatorTypes.cpp.

References InterpolatorIterator::append(), Collection< InterpolatorEntry >::get(), DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), m_sample_count_type, m_time_type, m_value_type, Iterator< T >::next(), Collection< InterpolatorEntry >::size(), Collection< InterpolatorEntry >::sortCollection(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

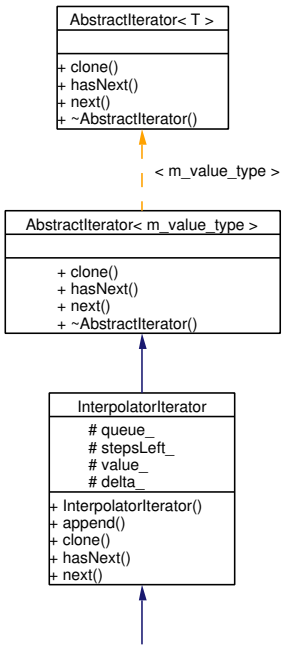
The documentation for this class was generated from the following files:

- InterpolatorTypes.h
- InterpolatorTypes.cpp

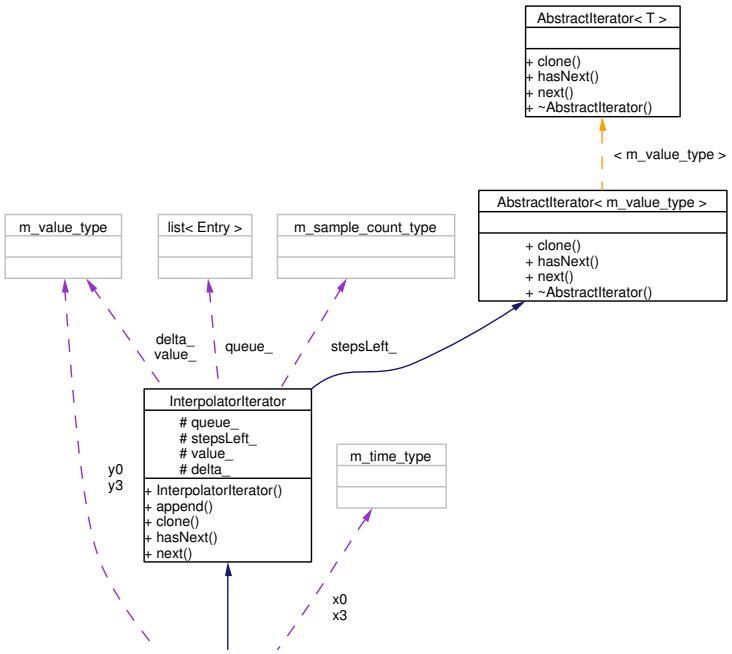
8.9 CubicSplineInterpolatorIterator Class Reference

`#include <InterpolatorIterator.h>`

Inheritance diagram for CubicSplineInterpolatorIterator:



Collaboration diagram for CubicSplineInterpolatorIterator:



Public Member Functions

- CubicSplineInterpolatorIterator ()

  constructor for the iterator

- CubicSplineInterpolatorIterator * clone ()

  make a clone of the iterator

- m_value_type & next ()

  get the next value in the iterator

Private Attributes

- m_time_type x0
- m_value_type y0

- m_time_type x3
- m_value_type y3

8.9.1 Detailed Description

This is an interpolator that will iterate over values in a CubicSplineInterpolator.

Definition at line 188 of file InterpolatorIterator.h.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 CubicSplineInterpolatorIterator::CubicSplineInterpolatorIterator ()

constructor for the iterator

Definition at line 200 of file InterpolatorIterator.cpp.

Referenced by clone().

8.9.3 Member Function Documentation

8.9.3.1 CubicSplineInterpolatorIterator * CubicSplineInterpolatorIterator::clone () [virtual]

make a clone of the iterator

Implements InterpolatorIterator.

Definition at line 205 of file InterpolatorIterator.cpp.

References CubicSplineInterpolatorIterator().

8.9.3.2 m_value_type & CubicSplineInterpolatorIterator::next () [virtual]

get the next value in the iterator

Implements InterpolatorIterator.

Definition at line 230 of file InterpolatorIterator.cpp.

References m_time_type, m_value_type, InterpolatorIterator::queue_, InterpolatorIterator::stepsLeft_, InterpolatorIterator::value_, x0, x3, y0, and y3.

8.9.4 Member Data Documentation

8.9.4.1 m_time_type CubicSplineInterpolatorIterator::x0 [private]

Definition at line 191 of file InterpolatorIterator.h.

Referenced by next().

8.9.4.2 m_time_type CubicSplineInterpolatorIterator::x3 [private]

Definition at line 193 of file InterpolatorIterator.h.

Referenced by next().

8.9.4.3 m_value_type CubicSplineInterpolatorIterator::y0 [private]

Definition at line 192 of file InterpolatorIterator.h.

Referenced by next().

8.9.4.4 m_value_type CubicSplineInterpolatorIterator::y3 [private]

Definition at line 194 of file InterpolatorIterator.h.

Referenced by next().

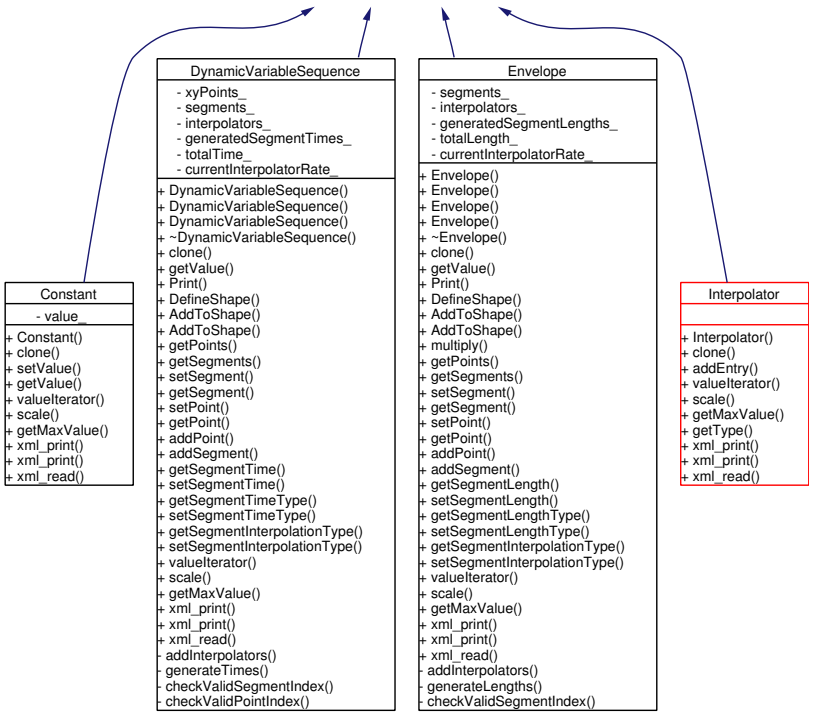
The documentation for this class was generated from the following files:

- InterpolatorIterator.h
- InterpolatorIterator.cpp

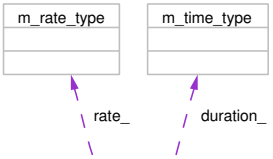
8.10 DynamicVariable Class Reference

`#include <DynamicVariable.h>`

Inheritance diagram for DynamicVariable:



Collaboration diagram for DynamicVariable:



Public Member Functions

- DynamicVariable ()
- virtual ~DynamicVariable ()
- virtual DynamicVariable * clone ()=0
- virtual Iterator< m_value_type > valueIterator ()=0
- virtual void scale (m_value_type factor)=0
- virtual m_value_type getMaxValue ()=0
- void setDuration (m_time_type duration)
- m_time_type getDuration ()
- void setSamplingRate (m_rate_type rate)
- m_rate_type getSamplingRate ()
- m_sample_count_type getSampleCount ()
- virtual void xml_print (ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)=0
- virtual void xml_print (ofstream &xmlOutput)=0

Static Public Member Functions

- DynamicVariable * create_dv_from_xml (XmlReader::xmltag *dvtag)

Private Attributes

- m_time_type duration_
- m_rate_type rate_

8.10.1 Detailed Description

An abstract definition of a simple Dynamic Variable. This simply defines an m_value_-type that changes over time. Any implemention that derives from DynamicVariable can be added.

Author:
Braden Kowitz

Definition at line 55 of file DynamicVariable.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 DynamicVariable::DynamicVariable ()

Default constructor for DynamicVariables. Sets the length to 1 second. Sets the rate to DEFAULT_SAMPLING_RATE

Definition at line 42 of file DynamicVariable.cpp.

8.10.2.2 virtual DynamicVariable::~DynamicVariable () [inline, virtual]

Destructor.

Definition at line 69 of file DynamicVariable.h.

8.10.3 Member Function Documentation

8.10.3.1 virtual DynamicVariable* DynamicVariable::clone () [pure virtual]

Creates an exact duplicate of this variable.

Implemented in Constant, DynamicVariableSequence, Envelope, Interpolator, LinearInterpolator, ExponentialInterpolator, and CubicSplineInterpolator.

Referenced by Partial::render(), Pan::set(), and ParameterLib< StaticT, DynamicT >::setParam().

8.10.3.2 DynamicVariable * DynamicVariable::create_dv_from_xml (XmlReader::xmltag * *dvtag*) [static]

Deprecated
This create a DynamicVariable from xml

Parameters:
dvtag Some xml

Returns:
A DynamicVariable

Definition at line 80 of file DynamicVariable.cpp.

References CUBIC_SPLINE, EXPONENTIAL, XmlReader::xmltag::findChildParamValue(), LINEAR, Interpolator::xml_read(), Envelope::xml_read(), and Constant::xml_read().

Referenced by Partial::auxLoadParam(), and Score::xml_read().

8.10.3.3 m_time_type DynamicVariable::getDuration ()

Returns the Length in seconds of this Dynamic Variablei

Returns:
The duration in seconds

Definition at line 54 of file DynamicVariable.cpp.

References duration_, and m_time_type.

Referenced by CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), Envelope::valueIterator(), DynamicVariableSequence::valueIterator(), Interpolator::xml_print(), Envelope::xml_print(), DynamicVariableSequence::xml_print(), and Constant::xml_print().

8.10.3.4 virtual m_value_type DynamicVariable::getMaxValue () [pure virtual]

Returns the maximum value present in this variable.

Returns:
the max value

Implemented in Constant, DynamicVariableSequence, Envelope, and Interpolator.

Referenced by Loudness::calculate().

8.10.3.5 m_sample_count_type DynamicVariable::getSampleCount ()

Returns the number of samples at the current length and rate.

Returns:
number of samples

Definition at line 73 of file DynamicVariable.cpp.

References duration_, m_sample_count_type, and rate_.

Referenced by Constant::valueIterator().

8.10.3.6 m_rate_type DynamicVariable::getSamplingRate ()

Returns the sampling rate.

Returns:
The sampling rate

Definition at line 67 of file DynamicVariable.cpp.

References m_rate_type, and rate_.

Referenced by CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), Envelope::valueIterator(), DynamicVariableSequence::valueIterator(), Interpolator::xml_print(), Envelope::xml_print(), DynamicVariableSequence::xml_print(), and Constant::xml_print().

8.10.3.7 virtual void DynamicVariable::scale (m_value_type *factor*) [pure virtual]

Scales this DynamicIterator by the given factor.

Parameters:
factor The scaling factor

Implemented in Constant, DynamicVariableSequence, Envelope, and Interpolator.

8.10.3.8 void DynamicVariable::setDuration (m_time_type *duration*)

Sets the length of this dynamic variable. Will also affect the sample count for this variable.

Parameters:
duration The duration

Definition at line 48 of file DynamicVariable.cpp.

References duration_, and m_time_type.

Referenced by Envelope::addInterpolators(), DynamicVariableSequence::addInterpolators(), Loudness::calculate(), Envelope::getValue(), DynamicVariableSequence::getValue(), Partial::render(), Pan::spatialize(), Interpolator::xml_read(), Envelope::xml_read(), and Constant::xml_read().

8.10.3.9 void DynamicVariable::setSamplingRate (m_rate_type *rate*)

Sets the sampling rate for this dynamic variable. Will also affect the sample count for this variable.

Parameters:

rate The sampling rate

Definition at line 61 of file DynamicVariable.cpp.

References m_rate_type, and rate_.

Referenced by Envelope::addInterpolators(), DynamicVariableSequence::addInterpolators(), Loudness::calculate(), Envelope::getValue(), DynamicVariableSequence::getValue(), Partial::render(), Pan::spatialize(), Interpolator::xml_read(), Envelope::xml_read(), and Constant::xml_read().

8.10.3.10 virtual Iterator<m_value_type> DynamicVariable::valueIterator () [pure virtual]

Returns an iterator object that will iterate over all values in this DynamicVariable.

Returns:

An iterator

Implemented in Constant, DynamicVariableSequence, Envelope, Interpolator, LinearInterpolator, ExponentialInterpolator, and CubicSplineInterpolator.

Referenced by Loudness::calculate(), Partial::render(), and Pan::spatialize().

8.10.3.11 virtual void DynamicVariable::xml_print (ofstream & *xmlOutput*) [pure virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implemented in Constant, DynamicVariableSequence, Envelope, and Interpolator.

8.10.3.12 virtual void DynamicVariable::xml_print (ofstream & *xmlOutput*, list< DynamicVariable * > & *dynObjs*) [pure virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implemented in Constant, DynamicVariableSequence, Envelope, and Interpolator.

Referenced by Partial::xml_print().

8.10.4 Member Data Documentation

8.10.4.1 m_time_type DynamicVariable::duration_ [private]

The duration

Definition at line 155 of file DynamicVariable.h.

Referenced by getDuration(), getSampleCount(), and setDuration().

8.10.4.2 m_rate_type DynamicVariable::rate_ [private]

The sampling rate

Definition at line 160 of file DynamicVariable.h.

Referenced by getSampleCount(), getSamplingRate(), and setSamplingRate().

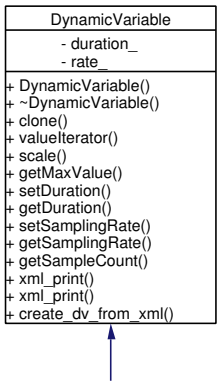
The documentation for this class was generated from the following files:

- DynamicVariable.h
- DynamicVariable.cpp

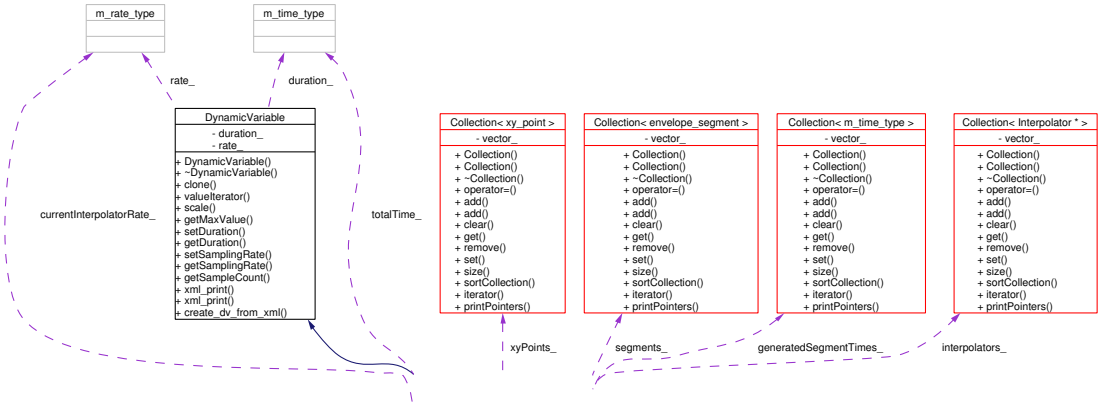
8.11 DynamicVariableSequence Class Reference

`#include <DynamicVariableSequence.h>`

Inheritance diagram for DynamicVariableSequence:



Collaboration diagram for DynamicVariableSequence:



Public Member Functions

- DynamicVariableSequence ()

  Constructor - no parameters...assumes data will be given later.

- DynamicVariableSequence (DynamicVariableSequence &dvs)

  Constructor - no parameters...assumes data will be given later.

- DynamicVariableSequence (Collection< xy_point > xyPoints, Collection< envelope_segment > segments)
- ~DynamicVariableSequence ()
- DynamicVariableSequence * clone ()
- m_value_type getValue (m_time_type time, m_time_type totalTime)
- void Print ()

- void DefineShape (Collection< xy_point > xyPoints, Collection< envelope_-segment > segments)
- void AddToShape (Collection< xy_point > xyPoints, Collection< envelope_-segment > segments)
- void AddToShape (DynamicVariableSequence *shape)
- Collection< xy_point > * getPoints ()
- Collection< envelope_segment > * getSegments ()
- void setSegment (int index, envelope_segment segment)
- envelope_segment getSegment (int index)
- void setPoint (int index, xy_point point)
- xy_point getPoint (int index)
- void addPoint (xy_point point)
- void addSegment (envelope_segment segment)
- m_time_type getSegmentTime (int index)
- void setSegmentTime (int index, m_time_type time)
- stretch_type getSegmentTimeType (int index)
- void setSegmentTimeType (int index, stretch_type timeType)
- interpolation_type getSegmentInterpolationType (int index)
- void setSegmentInterpolationType (int index, interpolation_type interType)
- Iterator< m_value_type > valueIterator ()
- void scale (m_value_type factor)
- m_value_type getMaxValue ()
- void xml_print (ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)
- void xml_print (ofstream &xmlOutput)
- void xml_read (XmlReader *xml, char *tagname)

Private Member Functions

- void addInterpolators (m_rate_type rate)
- void generateTimes (m_time_type totalTime)
- bool checkValidSegmentIndex (int index)
- bool checkValidPointIndex (int index)

Private Attributes

- Collection< xy_point > * xyPoints_
- Collection< envelope_segment > * segments_
- Collection< Interpolator * > * interpolators_
- Collection< m_time_type > * generatedSegmentTimes_
- m_time_type totalTime_
- m_rate_type currentInterpolatorRate_

8.11.1 Detailed Description

This class is designed to "put together" several DynamicVariables that then act as a single DynamicVariable. This allows easy combining of shapes for sounds defined by DynamicVariables to make bigger shapes. It also adds the ability to specify whether a particular DynamicVariable plays for a fixed amount of time or for a percentage of time available. For each DynamicVariableSequenceEntry that is stored in the DynamicVariableSequence, the user specifies how long the Entry will play for. Checking is done to ensure that if invalid input is given, the values are adjusted accordingly. The sampledvs∗.cpp files in the samples directory demonstrate how checking is done.

As demonstrated in the sample files, the values for a fixed amount of time are the amount of time that this DynamicVariableSequenceEntry will play for, and the only reason that it would not play for this much time is if the sound that it is a shape for does not play for that much time. In this case, the amount of time will get scaled to fill up the whole time available.

Also demonstrated in the sample files are the ways that flexible time is assigned. Basically, all values for flexible time are relative to one another. This is easiest explained by examples:

- DynamicVariableSequence A will play for 10 seconds and has 3 Entries:
  - Entry 1 uses flexible time with a value of 0.1 as a "percentage."
  - Entry 2 uses flexible time with a value of 0.4 as a "percentage."
  - Entry 3 uses flexible time with a value of 0.5 as a "percentage."
- DynamicVariableSequence B will play for 10 seconds and has 3 Entries:
  - Entry 1 uses flexible time with a value of 10 as a "percentage."
  - Entry 2 uses flexible time with a value of 40 as a "percentage."
  - Entry 3 uses flexible time with a value of 50 as a "percentage."
- DynamicVariableSequence C will play for 10 seconds and has 3 Entries:
  - Entry 1 uses flexible time with a value of 20 as a "percentage."
  - Entry 2 uses flexible time with a value of 80 as a "percentage."
  - Entry 3 uses flexible time with a value of 100 as a "percentage."

Since all the "percentage" values are relative to one another, these will all define the same shape. Thus, the value of every "percentage" is normalized to whatever value all the flexible times add up to.

Author:
Jon Kishkunas

Definition at line 84 of file DynamicVariableSequence.h.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 DynamicVariableSequence::DynamicVariableSequence ()

Constructor - no parameters...assumes data will be given later.

Definition at line 43 of file DynamicVariableSequence.cpp.

References currentInterpolatorRate_, generatedSegmentTimes_, interpolators_, segments_, totalTime_, and xyPoints_.

Referenced by clone().

8.11.2.2 DynamicVariableSequence::DynamicVariableSequence (DynamicVariableSequence & *dvs*)

Constructor - no parameters...assumes data will be given later.

Definition at line 54 of file DynamicVariableSequence.cpp.

References currentInterpolatorRate_, generatedSegmentTimes_, getPoints(), getSegments(), interpolators_, segments_, totalTime_, and xyPoints_.

8.11.2.3 DynamicVariableSequence::DynamicVariableSequence (Collection< xy_point > *xyPoints*, Collection< envelope_segment > *segments*)

Constructor

Parameters:

xyPoints A Collection of points

segments A Collection of information for the segments between the points

Definition at line 70 of file DynamicVariableSequence.cpp.

References currentInterpolatorRate_, DefineShape(), generatedSegmentTimes_, interpolators_, segments_, totalTime_, and xyPoints_.

8.11.2.4 DynamicVariableSequence::~DynamicVariableSequence ()

Destructor

Definition at line 87 of file DynamicVariableSequence.cpp.

References generatedSegmentTimes_, interpolators_, Collection< Interpolator * >::remove(), segments_, Collection< Interpolator * >::size(), and xyPoints_.

8.11.3 Member Function Documentation

8.11.3.1 void DynamicVariableSequence::addInterpolators (m_rate_type *rate*) [private]

This populates the private member variable with actual interpolators.

Parameters:

rate The sampling rate

Definition at line 722 of file DynamicVariableSequence.cpp.

References Collection< Interpolator * >::add(), Interpolator::addEntry(), CUBIC_SPLINE, EXPONENTIAL, generatedSegmentTimes_, Collection< xy_point >::get(), Collection< m_time_type >::get(), Collection< Interpolator * >::get(), getSegmentInterpolationType(), interpolators_, m_rate_type, m_time_type, segments_, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), Collection< xy_point >::size(), Collection< envelope_segment >::size(), xyPoints_, and xy_point::y.

Referenced by valueIterator().

8.11.3.2 void DynamicVariableSequence::addPoint (xy_point *point*)

This appends a point to the end of the DVS.

Parameters:

point The point to append

Definition at line 474 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::add(), and xyPoints_.

8.11.3.3 void DynamicVariableSequence::addSegment (envelope_segment *segment*)

This appends a segment to the end of the DVS.

Parameters:

segment The segment to append

Definition at line 481 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::add(), and segments_.

8.11.3.4 void DynamicVariableSequence::AddToShape (DynamicVariableSequence * *shape*)

This appends one DVS to another.

Parameters:

shape The DVS to append to this one

Definition at line 414 of file DynamicVariableSequence.cpp.

8.11.3.5 void DynamicVariableSequence::AddToShape (Collection< xy_point > *xyPoints*, Collection< envelope_segment > *segments*)

This adds to the DVS the given point and segment descriptions.

Parameters:

xyPoints A Collection of points

segments A Collection of information about the segments between points

Definition at line 375 of file DynamicVariableSequence.cpp.

References m_time_type, and xy_point::x.

8.11.3.6 bool DynamicVariableSequence::checkValidPointIndex (int *index*) [inline, private]

This checks if a number is a valid index for a point.

Parameters:

index The index to check

Return values:

true Is valid

false Is not valid

Definition at line 558 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::size(), and xyPoints_.

8.11.3.7 bool DynamicVariableSequence::checkValidSegmentIndex (int *index*) [inline, private]

This checks if a number is a valid index for a segment.

Parameters:
index The index to check

Return values:
true Is valid

false Is not valid

Definition at line 550 of file DynamicVariableSequence.cpp.

References segments_, and Collection< envelope_segment >::size().

8.11.3.8 DynamicVariableSequence * DynamicVariableSequence::clone () [virtual]

This makes a copy of this object.

Returns:
A pointer to a DynamicVariableSequence

Implements DynamicVariable.

Definition at line 190 of file DynamicVariableSequence.cpp.

References DynamicVariableSequence(), segments_, and xyPoints_.

8.11.3.9 void DynamicVariableSequence::DefineShape (Collection< xy_point > *xyPoints*, Collection< envelope_segment > *segments*) [inline]

This (re)defines the DVS point and segment descriptions.

Parameters:
xyPoints A Collection of points

segments A Collection of information about the segments between points

Definition at line 349 of file DynamicVariableSequence.cpp.

Referenced by DynamicVariableSequence().

8.11.3.10 void DynamicVariableSequence::generateTimes (m_time_type *totalTime*) [private]

This populates the private member variable with actual times.

Parameters:
totalTime The time

Definition at line 592 of file DynamicVariableSequence.cpp.

References Collection< m_time_type >::add(), FIXED, FLEXIBLE, generatedSegmentTimes_, Collection< m_time_type >::get(), getSegmentTime(), getSegmentTimeType(), m_time_type, segments_, Collection< m_time_type >::set(), Collection< envelope_segment >::size(), and totalTime_.

Referenced by getValue(), and valueIterator().

8.11.3.11 m_value_type DynamicVariableSequence::getMaxValue () [virtual]

This returns the maximum value of all the entries.

Returns:
The max value

Implements DynamicVariable.

Definition at line 802 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::get(), m_value_type, Collection< xy_point >::size(), xyPoints_, and xy_point::y.

8.11.3.12 xy_point DynamicVariableSequence::getPoint (int *index*)

This gets a point by index.

Parameters:
index Which point to get

Returns:
A point

Definition at line 466 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::get(), and xyPoints_.

Referenced by getValue().

8.11.3.13 Collection< xy_point > * DynamicVariableSequence::getPoints ()

This returns the Collection of points that make up the shape.

Returns:
A Collection of points

Definition at line 426 of file DynamicVariableSequence.cpp.

Referenced by DynamicVariableSequence().

8.11.3.14 envelope_segment DynamicVariableSequence::getSegment (int *index*)

This gets a segment by index.

Parameters:
index Which segment to get

Returns:
An envelope_segment

Definition at line 450 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::get(), and segments_.

8.11.3.15 interpolation_type DynamicVariableSequence::getSegment-InterpolationType (int *index*)

This gets the interpolation type for the segment with the given index. (LINEAR, EXPONENTIAL, etc).

Parameters:
index Which segment to get

Returns:
The interpolation type

Definition at line 531 of file DynamicVariableSequence.cpp.

Referenced by addInterpolators(), getValue(), and Print().

8.11.3.16 Collection< envelope_segment > * DynamicVariableSequence::get-Segments ()

This returns the collection of segments that make up the shape.

Returns:
A Collection of segments

Definition at line 434 of file DynamicVariableSequence.cpp.

Referenced by DynamicVariableSequence().

8.11.3.17 m_time_type DynamicVariableSequence::getSegmentTime (int *index*)

This gets the indexed segment's time, that is, it's actual playing time (if using FIXED time) or percentage of time available (if using FLEXIBLE time).

Parameters:
index Which segment to get

Returns:
the playing time

Definition at line 488 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::get(), m_time_type, segments_, and envelope_segment::timeValue.

Referenced by generateTimes(), and Print().

8.11.3.18 stretch_type DynamicVariableSequence::getSegmentTimeType (int *index*)

This gets the time type for the segment with the given index

Parameters:
index Which segment to examine

Return values:
FIXED

FLEXIBLE

Definition at line 511 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::get(), segments_, stretch_type, and envelope_segment::timeType.

Referenced by generateTimes(), and Print().

8.11.3.19 m_value_type DynamicVariableSequence::getValue (m_time_type *time*, m_time_type *totalTime*)

This gets an approximation of the value at the specified time.

Parameters:
time The time to begin looking at the value

totalTime The elapsed time to look at the value

Returns:
The approximate value during that time

Definition at line 199 of file DynamicVariableSequence.cpp.

References Interpolator::addEntry(), CUBIC_SPLINE, EXPONENTIAL, generatedSegmentTimes_, generateTimes(), Collection< m_time_type >::get(), Collection< xy_point >::get(), getPoint(), getSegmentInterpolationType(), m_time_type, m_value_type, Iterator< T >::next(), DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), Collection< xy_point >::size(), Interpolator::valueIterator(), xyPoints_, and xy_point::y.

8.11.3.20 void DynamicVariableSequence::Print ()

This will give a text output for what is stored in the DVS.

Note:
mainly for debugging

Definition at line 120 of file DynamicVariableSequence.cpp.

References CUBIC_SPLINE, EXPONENTIAL, FLEXIBLE, Collection< xy_point >::get(), getSegmentInterpolationType(), getSegmentTime(), getSegmentTimeType(), LINEAR, segments_, Collection< envelope_segment >::size(), xy_point::x, xyPoints_, and xy_point::y.

8.11.3.21 void DynamicVariableSequence::scale (m_value_type *factor*) [virtual]

This scales all Entries' values by this factor.

Parameters:
factor The factor to scale by

Implements DynamicVariable.

Definition at line 785 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::get(), m_value_type, Collection< xy_point >::set(), Collection< xy_point >::size(), xy_point::x, xyPoints_, and xy_point::y.

8.11.3.22 void DynamicVariableSequence::setPoint (int *index*, xy_point *point*)

This sets a point by index.

Parameters:
index Which point to set

point The new point

Definition at line 458 of file DynamicVariableSequence.cpp.

References Collection< xy_point >::set(), and xyPoints_.

8.11.3.23 void DynamicVariableSequence::setSegment (int *index*, envelope_segment *segment*)

This sets a segment by index.

Parameters:

index Which segment to set

segment The new segment

Definition at line 442 of file DynamicVariableSequence.cpp.

References segments_, and Collection< envelope_segment >::set().

8.11.3.24 void DynamicVariableSequence::setSegmentInterpolationType (int *index*, interpolation_type *interType*)

This sets the interpolation type for the segment with the given index.

Parameters:

index Which segment to set

interType The interpolation type

Definition at line 540 of file DynamicVariableSequence.cpp.

References interpolation_type, and envelope_segment::interType.

8.11.3.25 void DynamicVariableSequence::setSegmentTime (int *index*, m_time_type *time*)

This sets the indexed segment's time.

Parameters:

index Which segment to set

time The duration of the segment

Definition at line 496 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::get(), m_time_type, segments_, Collection< envelope_segment >::set(), and envelope_segment::timeValue.

8.11.3.26 void DynamicVariableSequence::setSegmentTimeType (int *index*, stretch_type *timeType*)

This sets the indexed segment's time type.

Parameters:

index Which segment to set

timeType The type of time to set (FIXED/FLEXIBLE)

Definition at line 519 of file DynamicVariableSequence.cpp.

References Collection< envelope_segment >::get(), segments_, Collection< envelope_segment >::set(), and envelope_segment::timeType.

8.11.3.27 Iterator< m_value_type > DynamicVariableSequence::valueIterator () [virtual]

An iterator through all the values.

Returns:

An iterator

Implements DynamicVariable.

Definition at line 566 of file DynamicVariableSequence.cpp.

References addInterpolators(), currentInterpolatorRate_, generateTimes(), DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), interpolators_, and totalTime_.

8.11.3.28 void DynamicVariableSequence::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 821 of file DynamicVariableSequence.cpp.

References currentInterpolatorRate_, DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), Iterator< T >::hasNext(), envelope_segment::interType, Collection< envelope_segment >::iterator(), Collection< xy_point >::iterator(), Iterator< T >::next(), segments_, envelope_segment::timeType, envelope_segment::timeValue, totalTime_, xy_point::x, xyPoints_, and xy_point::y.

8.11.3.29 void DynamicVariableSequence::xml_print (ofstream & *xmlOutput*, list< DynamicVariable * > & *dynObjs*) [virtual]

Deprecated
This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 858 of file DynamicVariableSequence.cpp.

8.11.3.30 void DynamicVariableSequence::xml_read (XmlReader * *xml*, char * *tagname*)

Deprecated
This reads the DVS from xml

Definition at line 879 of file DynamicVariableSequence.cpp.

References XmlReader::xmltag::getParamValue(), XmlReader::xmltag::isClosing, XmlReader::xmltag::name, and XmlReader::readTag().

8.11.4 Member Data Documentation

8.11.4.1 m_rate_type DynamicVariableSequence::currentInterpolatorRate_ [private]

This is the sampling rate with which interpolators were generated

Definition at line 307 of file DynamicVariableSequence.h.

Referenced by DynamicVariableSequence(), valueIterator(), and xml_print().

8.11.4.2 Collection<m_time_type>* DynamicVariableSequence::generatedSegmentTimes_ [private]

Time values for segments that were generated based on the stored value

Definition at line 297 of file DynamicVariableSequence.h.

Referenced by addInterpolators(), DynamicVariableSequence(), generateTimes(), getValue(), and ~DynamicVariableSequence().

8.11.4.3 Collection<Interpolator*>* DynamicVariableSequence::interpolators_ [private]

This is a Collection to hold interpolators

Definition at line 292 of file DynamicVariableSequence.h.

Referenced by addInterpolators(), DynamicVariableSequence(), valueIterator(), and ~DynamicVariableSequence().

8.11.4.4 Collection<envelope_segment>* DynamicVariableSequence::segments_ [private]

This is a Collection to hold the segment data for this DVS

Definition at line 287 of file DynamicVariableSequence.h.

Referenced by addInterpolators(), addSegment(), checkValidSegmentIndex(), clone(), DynamicVariableSequence(), generateTimes(), getSegment(), getSegmentTime(), getSegmentTimeType(), Print(), setSegment(), setSegmentTime(), setSegmentTimeType(), xml_print(), and ~DynamicVariableSequence().

8.11.4.5 m_time_type DynamicVariableSequence::totalTime_ [private]

This is the value with which the segment times were generated

Definition at line 302 of file DynamicVariableSequence.h.

Referenced by DynamicVariableSequence(), generateTimes(), valueIterator(), and xml_print().

8.11.4.6 Collection<xy_point>* DynamicVariableSequence::xyPoints_ [private]

This is a Collection to hold the x-y points that define this DVS

Definition at line 282 of file DynamicVariableSequence.h.

Referenced by addInterpolators(), addPoint(), checkValidPointIndex(), clone(), DynamicVariableSequence(), getMaxValue(), getPoint(), getValue(), Print(), scale(), setPoint(), xml_print(), and ~DynamicVariableSequence().

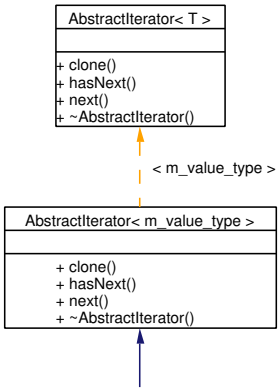
The documentation for this class was generated from the following files:

- DynamicVariableSequence.h
- DynamicVariableSequence.cpp

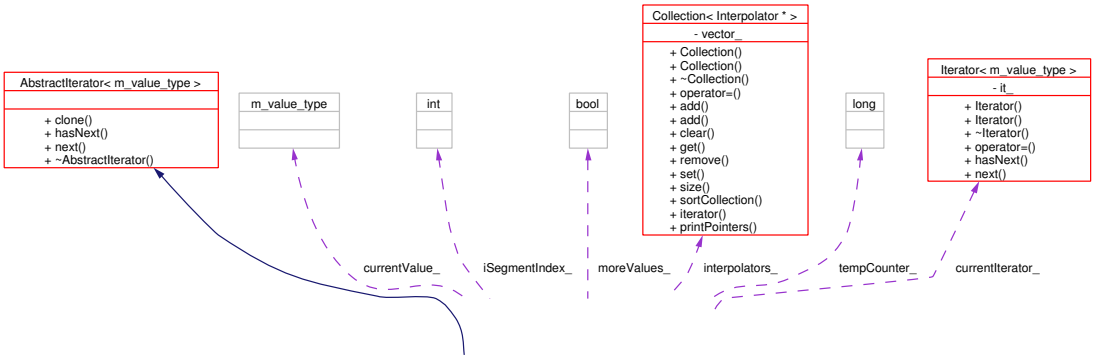
8.12 DynamicVariableSequenceIterator Class Reference

`#include <DynamicVariableSequenceIterator.h>`

Inheritance diagram for DynamicVariableSequenceIterator:



Collaboration diagram for DynamicVariableSequenceIterator:



Public Member Functions

- DynamicVariableSequenceIterator (Collection< Interpolator * > interpolators)
- ~DynamicVariableSequenceIterator ()
- DynamicVariableSequenceIterator * clone ()
- bool hasNext ()
- m_value_type & next ()

Private Attributes

- Collection< Interpolator * > * interpolators_
- int iSegmentIndex_
- Iterator< m_value_type > * currentIterator_
- m_value_type currentValue_
- bool moreValues_
- long tempCounter_

8.12.1 Detailed Description

This is an iterator that interpolates over a DynamicVariableSequence. Since a DynamicVariableSequence acts like a single DynamicVariable, when we want to use it as such, we need a way to iterate through its values. That's what this does.

Author:
Jon Kishkunas

Definition at line 50 of file DynamicVariableSequenceIterator.h.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 DynamicVariableSequenceIterator::DynamicVariableSequenceIterator (Collection< Interpolator * > *interpolators*)

This is a constructor which initalizes some basic values.

Parameters:
interpolators

Definition at line 37 of file DynamicVariableSequenceIterator.cpp.

References currentIterator_, currentValue_, Collection< InterpolatorEntry >::get(), Collection< Interpolator * >::get(), interpolators_, iSegmentIndex_, moreValues_, Collection< InterpolatorEntry >::size(), Collection< Interpolator * >::size(), Collection< T >::size(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

Referenced by clone().

8.12.2.2 DynamicVariableSequenceIterator::~DynamicVariableSequenceIterator ()

This is the destructor.

Definition at line 81 of file DynamicVariableSequenceIterator.cpp.

References currentIterator_, and interpolators_.

8.12.3 Member Function Documentation

8.12.3.1 DynamicVariableSequenceIterator * DynamicVariableSequenceIterator::clone () [virtual]

This makes a copy of this iterator.

Returns:
A DynamicVariableSequenceIterator that is a clone of this one

Implements AbstractIterator< m_value_type >.

Definition at line 100 of file DynamicVariableSequenceIterator.cpp.

References DynamicVariableSequenceIterator().

8.12.3.2 bool DynamicVariableSequenceIterator::hasNext () [virtual]

Indicates whether the iterator has another value to return

Return values:

true If there is another value to return

false If there is not another value to return

Implements AbstractIterator< m_value_type >.

Definition at line 107 of file DynamicVariableSequenceIterator.cpp.

References moreValues_.

8.12.3.3 m_value_type & DynamicVariableSequenceIterator::next () [virtual]

Returns the next value in the iteration.

Note:

Because this returns a reference type, value_ can be changed by the caller. Steps should be taken to prevent this (with a pass-to-caller member variable perhaps).

Returns:

A reference to the next value in the iteration

Implements AbstractIterator< m_value_type >.

Definition at line 115 of file DynamicVariableSequenceIterator.cpp.

References currentIterator_, currentValue_, Collection< Interpolator * >::get(), Iterator< m_value_type >::hasNext(), interpolators_, iSegmentIndex_, m_value_type, moreValues_, Iterator< m_value_type >::next(), Collection< Interpolator * >::size(), tempCounter_, and Interpolator::valueIterator().

8.12.4 Member Data Documentation

8.12.4.1 Iterator<m_value_type>* DynamicVariableSequenceIterator::currentIterator_ [private]

This is the current DV iterator.

Definition at line 102 of file DynamicVariableSequenceIterator.h.

Referenced by DynamicVariableSequenceIterator(), next(), and ~DynamicVariableSequenceIterator().

8.12.4.2 m_value_type DynamicVariableSequenceIterator::currentValue_ [private]

This is the current value.

Definition at line 107 of file DynamicVariableSequenceIterator.h.

Referenced by DynamicVariableSequenceIterator(), and next().

8.12.4.3 Collection<Interpolator*>* DynamicVariableSequenceIterator::interpolators_ [private]

This is a Collection that holds the interpolators for all the segments.

Definition at line 92 of file DynamicVariableSequenceIterator.h.

Referenced by DynamicVariableSequenceIterator(), next(), and ~DynamicVariableSequenceIterator().

8.12.4.4 int DynamicVariableSequenceIterator::iSegmentIndex_ [private]

This is the current segment's index.

Definition at line 97 of file DynamicVariableSequenceIterator.h.

Referenced by DynamicVariableSequenceIterator(), and next().

8.12.4.5 bool DynamicVariableSequenceIterator::moreValues_ [private]

This indicates whether we have more values.

Definition at line 112 of file DynamicVariableSequenceIterator.h.

Referenced by DynamicVariableSequenceIterator(), hasNext(), and next().

8.12.4.6 long DynamicVariableSequenceIterator::tempCounter_ [private]

Definition at line 114 of file DynamicVariableSequenceIterator.h.

Referenced by next().

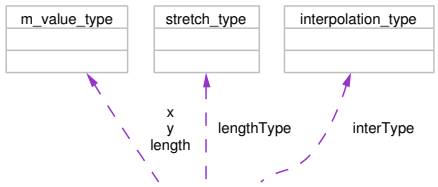
The documentation for this class was generated from the following files:

- DynamicVariableSequenceIterator.h
- DynamicVariableSequenceIterator.cpp

8.13 env_seg Struct Reference

`#include <Types.h>`

Collaboration diagram for env_seg:



Public Attributes

- interpolation_type interType

  interpolation type to use for the segment

- stretch_type lengthType

  length stretch type to use for the segment

- m_value_type length

  length of the segment if applicable

- m_value_type x

  x position of the segment point

- m_value_type y

  value at the right endpoint

8.13.1 Member Data Documentation

8.13.1.1 interpolation_type env_seg::interType

interpolation type to use for the segment

Definition at line 108 of file Types.h.

8.13.1.2 m_value_type env_seg::length

length of the segment if applicable

Definition at line 112 of file Types.h.

8.13.1.3 stretch_type env_seg::lengthType

length stretch type to use for the segment

Definition at line 110 of file Types.h.

8.13.1.4 m_value_type env_seg::x

x position of the segment point

Definition at line 114 of file Types.h.

8.13.1.5 m_value_type env_seg::y

value at the right endpoint

Definition at line 116 of file Types.h.

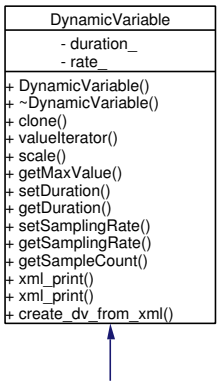
The documentation for this struct was generated from the following file:

- Types.h

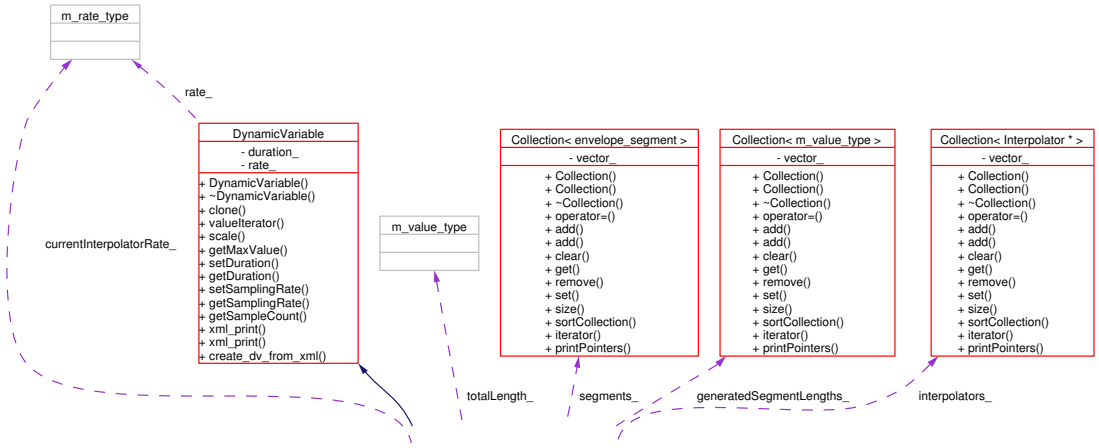
8.14 Envelope Class Reference

`#include <Envelope.h>`

Inheritance diagram for Envelope:



Collaboration diagram for Envelope:



Public Member Functions

- Envelope ()
- Envelope (Envelope &env)
- Envelope (Collection< xy_point > xy_points, Collection< envelope_segment > segs)
- Envelope (Collection< envelope_segment > segs)
- ~Envelope ()
- Envelope * clone ()
- m_value_type getValue (m_value_type x, m_value_type totalLength)
- void Print ()

- void DefineShape (Collection< envelope_segment > segs)
- void AddToShape (Collection< envelope_segment > segs)
- void AddToShape (Envelope *shape)
- Envelope * multiply (Envelope &env1, Envelope &env2)
- Collection< xy_point > * getPoints ()
- Collection< envelope_segment > * getSegments ()
- void setSegment (int index, envelope_segment segment)
- envelope_segment getSegment (int index)
- void setPoint (int index, xy_point point)
- xy_point getPoint (int index)
- void addPoint (xy_point point)
- void addSegment (envelope_segment segment)
- m_value_type getSegmentLength (int index)
- void setSegmentLength (int index, m_value_type length)
- stretch_type getSegmentLengthType (int index)
- void setSegmentLengthType (int index, stretch_type lengthType)
- interpolation_type getSegmentInterpolationType (int index)
- void setSegmentInterpolationType (int index, interpolation_type interType)
- Iterator< m_value_type > valueIterator ()
- void scale (m_value_type factor)
- m_value_type getMaxValue ()
- void xml_print (ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)
- void xml_print (ofstream &xmlOutput)
- void xml_read (XmlReader::xmltag *soundtag)

Private Member Functions

- void addInterpolators (m_rate_type rate)
- void generateLengths (m_time_type totalLength)
- bool checkValidSegmentIndex (int index)

Private Attributes

- Collection< envelope_segment > * segments_
- Collection< Interpolator * > * interpolators_
- Collection< m_value_type > * generatedSegmentLengths_
- m_value_type totalLength_
- m_rate_type currentInterpolatorRate_

8.14.1 Detailed Description

This class is designed to allow users to specify the shape of an envelope. This is done by using a collection of the env_seg type, which can be found in Types.h. For each segment, a user must specify the x value of the right endpoint of the segment, the type of interpolation to use for the segment, and whether or not the segment has a FIXED or FLEXIBLE length. If the total length specified by the user is greater than the sum of the lengths of the segments, then every segment with a length type of FLEXIBLE will be scaled up to fit the total length specified. Sample programs of how to use this class can be found in the samples directory, named sampleenvelope∗.cpp

Note:

All of the public functions of the class increment the index by 1. This means that a sample program will normally not access the dummy segment information used for the first point. In other words, getSegment(0) will actually retrieve segments_-[1]. This is why the index is changed when segments_->get() is used internally by the Envelope functions.

Author:

Michael Aikins
Greg Augustine

Definition at line 56 of file Envelope.h.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Envelope::Envelope ()

This is the constructor. There are no parameters since it assumes data will be given later.

Definition at line 44 of file Envelope.cpp.

References currentInterpolatorRate_, generatedSegmentLengths_, interpolators_-, segments_, and totalLength_.

Referenced by clone(), and multiply().

8.14.2.2 Envelope::Envelope (Envelope & *env*)

This is a copy constructor.

Parameters:

env The envolope to copy

Returns:
A copy of env

Definition at line 54 of file Envelope.cpp.

References currentInterpolatorRate_, generatedSegmentLengths_, getSegments(), interpolators_, segments_, and totalLength_.

8.14.2.3 Envelope::Envelope (Collection< xy_point > *xy_points*, Collection< envelope_segment > *segs*)

Constructor - Make new envelope using a collection of envelope segments and points

Definition at line 68 of file Envelope.cpp.

References Collection< T >::add(), currentInterpolatorRate_, DefineShape(), generatedSegmentLengths_, Collection< T >::get(), interpolators_, segments_, Collection< T >::size(), totalLength_, xy_point::x, envelope_segment::x, xy_point::y, and envelope_segment::y.

8.14.2.4 Envelope::Envelope (Collection< envelope_segment > *segs*)

This is a constructor. It makes a new envelope using a collection of envelope segments.

Parameters:
segs A Collection of envelop segments

Definition at line 94 of file Envelope.cpp.

References currentInterpolatorRate_, DefineShape(), generatedSegmentLengths_, interpolators_, segments_, and totalLength_.

8.14.2.5 Envelope::~Envelope ()

This is a destructor.

Definition at line 108 of file Envelope.cpp.

References generatedSegmentLengths_, interpolators_, Collection< Interpolator * >::remove(), segments_, and Collection< Interpolator * >::size().

8.14.3 Member Function Documentation

8.14.3.1 void Envelope::addInterpolators (m_rate_type *rate*) [private]

This function populate the private member variable with actual interpolators.

Parameters:
rate The rate

Definition at line 909 of file Envelope.cpp.

References Collection< Interpolator * >::add(), Interpolator::addEntry(), CUBIC_SPLINE, EXPONENTIAL, generatedSegmentLengths_, Collection< m_value_type >::get(), Collection< Interpolator * >::get(), Collection< envelope_segment >::get(), getSegmentInterpolationType(), interpolators_, m_rate_type, m_value_type, segments_, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), Collection< envelope_segment >::size(), and envelope_segment::y.

Referenced by valueIterator().

8.14.3.2 void Envelope::addPoint (xy_point *point*)

This function appends a point to the end of the envelope. It's default interpolation type is LINEAR and stretch type is FIXED.

Parameters:
point The point to append

Definition at line 658 of file Envelope.cpp.

References addSegment(), FIXED, envelope_segment::interType, envelope_segment::lengthType, LINEAR, xy_point::x, envelope_segment::x, xy_point::y, and envelope_segment::y.

8.14.3.3 void Envelope::addSegment (envelope_segment *segment*)

This function appends a segment to the end of the envelope.

Parameters:
segment The segment to append

Definition at line 648 of file Envelope.cpp.

References Collection< envelope_segment >::add(), Collection< envelope_segment >::get(), envelope_segment::length, segments_, Collection< envelope_segment >::size(), and envelope_segment::x.

Referenced by addPoint().

8.14.3.4 void Envelope::AddToShape (Envelope * *shape*)

This overloaded function appends one envelope to another.

Parameters:
shape The Envelope to append

Definition at line 432 of file Envelope.cpp.

8.14.3.5 void Envelope::AddToShape (Collection< envelope_segment > *segs*)

This function adds to the env the given segment descriptions.

Parameters:
segs A Collection of envelope segments

Definition at line 392 of file Envelope.cpp.

References envelope_segment::length, m_value_type, and envelope_segment::x.

8.14.3.6 bool Envelope::checkValidSegmentIndex (int *index*) [inline, private]

This function checks if a number is a valid index for a segment.

Parameters:
index The index to check

Return values:
true Is valid
false Is not valid

Definition at line 735 of file Envelope.cpp.

References segments_, and Collection< envelope_segment >::size().

Referenced by setSegment().

8.14.3.7 Envelope * Envelope::clone () [virtual]

This makes a copy of this object.

Returns:
A new Envelope

Implements DynamicVariable.

Definition at line 207 of file Envelope.cpp.

References Envelope(), and segments_.

Referenced by EnvelopeLibrary::EnvelopeLibrary().

8.14.3.8 void Envelope::DefineShape (Collection< envelope_segment > *segs*) [inline]

This function (re)defines the Envelope given point and segment descriptions.

Parameters:
segs A Collection of envelope segments

Definition at line 366 of file Envelope.cpp.

References Collection< T >::get(), envelope_segment::length, m_value_type, segments_, Collection< T >::set(), Collection< T >::size(), and envelope_segment::x.

Referenced by Envelope(), and xml_read().

8.14.3.9 void Envelope::generateLengths (m_time_type *totalLength*) [private]

This function populates the private member variable with actual lengths. This functions scales all FIXED and FLEXIBLE lengths appropriately.

Note:
If you want to scale the envelope publicly, use the function scale.

Parameters:
totalLength The total length of time to scale to

Definition at line 768 of file Envelope.cpp.

References Collection< m_value_type >::add(), FIXED, FLEXIBLE, generatedSegmentLengths_, Collection< m_value_type >::get(), getSegmentLength(), getSegmentLengthType(), m_time_type, segments_, Collection< m_value_type >::set(), setSegmentLength(), Collection< envelope_segment >::size(), and totalLength_.

Referenced by getValue(), and valueIterator().

8.14.3.10 m_value_type Envelope::getMaxValue () [virtual]

This function returns the maximum value in all the entries.

Returns:
The maximum value

Implements DynamicVariable.

Definition at line 989 of file Envelope.cpp.

References Collection< envelope_segment >::get(), m_value_type, segments_, Collection< envelope_segment >::size(), and envelope_segment::y.

8.14.3.11 xy_point Envelope::getPoint (int *index*)

This function gets a point by its index.

Parameters:
index Which point to get

Returns:
The point

Definition at line 635 of file Envelope.cpp.

References Collection< envelope_segment >::get(), segments_, envelope_segment::x, xy_point::x, envelope_segment::y, and xy_point::y.

Referenced by getValue().

8.14.3.12 Collection< xy_point > * Envelope::getPoints ()

This function returns a collection of points that make up the shape.

Returns:
A Collection of points

Definition at line 573 of file Envelope.cpp.

References Collection< T >::add(), Collection< envelope_segment >::get(), segments_, Collection< envelope_segment >::size(), envelope_segment::x, xy_point::x, envelope_segment::y, and xy_point::y.

8.14.3.13 envelope_segment Envelope::getSegment (int *index*)

This function gets a segment by its index.

Parameters:
index Which segment to get

Returns:
An envelope segment

Definition at line 615 of file Envelope.cpp.

References Collection< envelope_segment >::get(), and segments_.

8.14.3.14 interpolation_type Envelope::getSegmentInterpolationType (int *index*)

This function gets the interpolation type for the segment with the given index.

Parameters:
index Which segment to examine

Returns:
The interpolation type

Return values:
LINEAR

EXPONENTIAL

etc.

Definition at line 714 of file Envelope.cpp.

Referenced by addInterpolators(), getValue(), and Print().

8.14.3.15 m_value_type Envelope::getSegmentLength (int *index*)

This function gets the indexed segment's length of time. If the segment length type is FLEXIBLE, the length returned will be the unscaled, original value.

Parameters:
index Which segment to examine

Returns:
The segment's length

Definition at line 668 of file Envelope.cpp.

References Collection< envelope_segment >::get(), envelope_segment::length, m_value_type, and segments_.

Referenced by generateLengths(), and Print().

8.14.3.16 stretch_type Envelope::getSegmentLengthType (int *index*)

This function gets the type for the segment with the given index.

Parameters:
index Which segment to examine

Returns:
The time type

Return values:
FIXED
FLEXIBLE

Definition at line 693 of file Envelope.cpp.

References Collection< envelope_segment >::get(), envelope_segment::lengthType, segments_, and stretch_type.

Referenced by generateLengths(), and Print().

8.14.3.17 Collection< envelope_segment > * Envelope::getSegments ()

This function returns a collection of segments that make up the shape.

Returns:
A Collection of envelope segments

Definition at line 589 of file Envelope.cpp.

Referenced by Envelope(), and multiply().

8.14.3.18 m_value_type Envelope::getValue (m_value_type *x*, m_value_type *totalLength*)

This gets an approximation of the value at the specified time.

Parameters:
x The type of value
totalLength The length of time

Returns:
The approximate value

Definition at line 216 of file Envelope.cpp.

References Interpolator::addEntry(), CUBIC_SPLINE, EXPONENTIAL, generatedSegmentLengths_, generateLengths(), Collection< m_value_type >::get(), Collection< envelope_segment >::get(), getPoint(), getSegmentInterpolationType(), m_value_type, Iterator< T >::next(), segments_, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), Collection< envelope_segment >::size(), Interpolator::valueIterator(), xy_point::y, and envelope_segment::y.

Referenced by multiply().

8.14.3.19 Envelope * Envelope::multiply (Envelope & *env1*, Envelope & *env2*)

This function multiplies two envelopes together.

Parameters:

env1 The first Envelope

env2 The second Envelope

Returns:

The new Envelope

Definition at line 444 of file Envelope.cpp.

References Collection< T >::add(), CUBIC_SPLINE, Envelope(), EXPONENTIAL, FLEXIBLE, Collection< T >::get(), getSegments(), getValue(), envelope_segment::interType, envelope_segment::lengthType, LINEAR, envelope_segment::x, and envelope_segment::y.

8.14.3.20 void Envelope::Print ()

This is a debugging function that gives a text output of what is stored in the Envelope.

Definition at line 137 of file Envelope.cpp.

References CUBIC_SPLINE, EXPONENTIAL, FLEXIBLE, Collection< envelope_segment >::get(), getSegmentInterpolationType(), getSegmentLength(), getSegmentLengthType(), LINEAR, segments_, Collection< envelope_segment >::size(), envelope_segment::x, and envelope_segment::y.

8.14.3.21 void Envelope::scale (m_value_type *factor*) [virtual]

This function scales all Entries' values by this factor.

Parameters:

factor The factor by which to scale

Implements DynamicVariable.

Definition at line 972 of file Envelope.cpp.

References Collection< envelope_segment >::get(), m_value_type, segments_, Collection< envelope_segment >::set(), Collection< envelope_segment >::size(), and envelope_segment::y.

8.14.3.22 void Envelope::setPoint (int *index*, xy_point *point*)

This function sets a point by identified by its index.

Parameters:

index Which point to set

point The new point value

Definition at line 625 of file Envelope.cpp.

References Collection< envelope_segment >::get(), segments_, Collection< envelope_segment >::set(), xy_point::x, envelope_segment::x, xy_point::y, and envelope_segment::y.

8.14.3.23 void Envelope::setSegment (int *index*, envelope_segment *segment*)

This function sets an envelope segment identified by its index.

Parameters:

index Which segment to set

segment The new segment value

Definition at line 597 of file Envelope.cpp.

References checkValidSegmentIndex(), Collection< envelope_segment >::get(), envelope_segment::length, segments_, Collection< envelope_segment >::set(), and envelope_segment::x.

8.14.3.24 void Envelope::setSegmentInterpolationType (int *index*, interpolation_type *interType*)

This function sets the interpolation type for the segment with the given index.

Parameters:

index Which segment to set

interType The new interpolation type

Definition at line 724 of file Envelope.cpp.

References interpolation_type, and envelope_segment::interType.

8.14.3.25 void Envelope::setSegmentLength (int *index*, m_value_type *length*)

This function sets the indexed segment's length of time.

Parameters:

index Which segment to set

length The new length of time

Definition at line 677 of file Envelope.cpp.

References Collection< envelope_segment >::get(), envelope_segment::length, m_value_type, segments_, and Collection< envelope_segment >::set().

Referenced by generateLengths().

8.14.3.26 void Envelope::setSegmentLengthType (int *index*, stretch_type *lengthType*)

This function sets the indexed segment's time type.

Parameters:

index Which segment to set

lengthType The new time type

Definition at line 702 of file Envelope.cpp.

References Collection< envelope_segment >::get(), envelope_segment::lengthType, segments_, and Collection< envelope_segment >::set().

8.14.3.27 Iterator< m_value_type > Envelope::valueIterator () [virtual]

This function returns an iterator that iterates through all the values.

Returns:

An iterator

Implements DynamicVariable.

Definition at line 743 of file Envelope.cpp.

References addInterpolators(), currentInterpolatorRate_, generateLengths(), DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), interpolators_, and totalLength_.

8.14.3.28 void Envelope::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 1009 of file Envelope.cpp.

References CUBIC_SPLINE, currentInterpolatorRate_, EXPONENTIAL, FIXED, FLEXIBLE, DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(),

Iterator< T >::hasNext(), envelope_segment::interType, Collection< envelope_-segment >::iterator(), envelope_segment::length, envelope_segment::lengthType, LINEAR, Iterator< T >::next(), segments_, totalLength_, envelope_segment::x, and envelope_segment::y.

8.14.3.29 void Envelope::xml_print (ofstream & *xmlOutput*, list< DynamicVariable * > & *dynObjs*) [virtual]

Deprecated

This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 1068 of file Envelope.cpp.

8.14.3.30 void Envelope::xml_read (XmlReader::xmltag * *soundtag*)

Deprecated

Definition at line 1089 of file Envelope.cpp.

References Collection< T >::add(), XmlReader::xmltag::children, CUBIC_SPLINE, DefineShape(), EXPONENTIAL, XmlReader::xmltag::findChildParamValue(), FIXED, FLEXIBLE, XmlReader::xmltag::getParamValue(), envelope_segment::inter-Type, envelope_segment::lengthType, LINEAR, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), envelope_segment::x, and envelope_-segment::y.

Referenced by DynamicVariable::create_dv_from_xml().

8.14.4 Member Data Documentation

8.14.4.1 m_rate_type Envelope::currentInterpolatorRate_ [private]

This is the sampling rate with which interpolators were generated *

Definition at line 301 of file Envelope.h.

Referenced by Envelope(), valueIterator(), and xml_print().

8.14.4.2 Collection<m_value_type>* Envelope::generatedSegmentLengths_ [private]

This is a Collection of Time values for segments that were generated based on the stored value. This collection will not include the 0 dummy segment, so element 0 will

correspond to element 1 in segments_.

Definition at line 295 of file Envelope.h.

Referenced by addInterpolators(), Envelope(), generateLengths(), getValue(), and ∼Envelope().

8.14.4.3 Collection<Interpolator∗>∗ Envelope::interpolators_ [private]

This is a Collection to hold interpolators.

Definition at line 287 of file Envelope.h.

Referenced by addInterpolators(), Envelope(), valueIterator(), and ∼Envelope().

8.14.4.4 Collection<envelope_segment>∗ Envelope::segments_ [private]

This is a Collection to hold the segment data for this DVS.

Definition at line 282 of file Envelope.h.

Referenced by addInterpolators(), addSegment(), checkValidSegmentIndex(), clone(), DefineShape(), Envelope(), generateLengths(), getMaxValue(), getPoint(), getPoints(), getSegment(), getSegmentLength(), getSegmentLengthType(), getValue(), Print(), scale(), setPoint(), setSegment(), setSegmentLength(), setSegmentLengthType(), xml_print(), and ∼Envelope().

8.14.4.5 m_value_type Envelope::totalLength_ [private]

This is the value with which the segment times were generated ∗

Definition at line 298 of file Envelope.h.

Referenced by Envelope(), generateLengths(), valueIterator(), and xml_print().

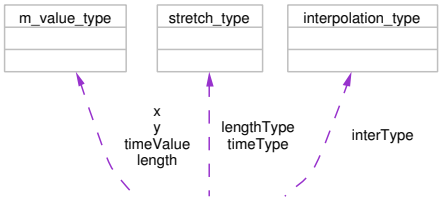
The documentation for this class was generated from the following files:

- Envelope.h
- Envelope.cpp

8.15 envelope_segment Struct Reference

`#include <Types.h>`

Collaboration diagram for envelope_segment:



Public Attributes

- interpolation_type interType

  interpolation type to use for the segment

- stretch_type timeType

  time stretch type to use for the segment

- m_value_type timeValue

  number of seconds or percentage of flex time that segment should play

- stretch_type lengthType

  length stretch type to use for the segment

- m_value_type length

  length if applicable

- m_value_type x

  x position of the right endpoint

- m_value_type y

  y value at the right endpoint

8.15.1 Detailed Description

This is used as a container for everything that a Envelope needs to know to create a particular segment in a shape.

Todo

Once DVS is completely gone and done for, timeType and timeValue should be deleted!

Definition at line 87 of file Types.h.

8.15.2 Member Data Documentation

8.15.2.1 interpolation_type envelope_segment::interType

interpolation type to use for the segment

Definition at line 90 of file Types.h.

Referenced by MultiPan::addEntryHelperFn(), Envelope::addPoint(), EnvelopeLibrary::loadLibrary(), Envelope::multiply(), Envelope::setSegmentInterpolationType(), DynamicVariableSequence::setSegmentInterpolationType(), Envelope::xml_print(), DynamicVariableSequence::xml_print(), and Envelope::xml_read().

8.15.2.2 m_value_type envelope_segment::length

length if applicable

Definition at line 98 of file Types.h.

Referenced by Envelope::addSegment(), Envelope::AddToShape(), Envelope::DefineShape(), Envelope::getSegmentLength(), EnvelopeLibrary::loadLibrary(), Envelope::setSegment(), Envelope::setSegmentLength(), and Envelope::xml_print().

8.15.2.3 stretch_type envelope_segment::lengthType

length stretch type to use for the segment

Definition at line 96 of file Types.h.

Referenced by MultiPan::addEntryHelperFn(), Envelope::addPoint(), Envelope::getSegmentLengthType(), EnvelopeLibrary::loadLibrary(), Envelope::multiply(), Envelope::setSegmentLengthType(), Envelope::xml_print(), and Envelope::xml_read().

8.15.2.4 stretch_type envelope_segment::timeType

time stretch type to use for the segment

Definition at line 92 of file Types.h.

Referenced by DynamicVariableSequence::getSegmentTimeType(), DynamicVariableSequence::setSegmentTimeType(), and DynamicVariableSequence::xml_print().

8.15.2.5 m_value_type envelope_segment::timeValue

number of seconds or percentage of flex time that segment should play

Definition at line 94 of file Types.h.

Referenced by MultiPan::addEntryHelperFn(), DynamicVariableSequence::getSegmentTime(), DynamicVariableSequence::setSegmentTime(), and DynamicVariableSequence::xml_print().

8.15.2.6 m_value_type envelope_segment::x

x position of the right endpoint

Definition at line 100 of file Types.h.

Referenced by Envelope::addPoint(), Envelope::addSegment(), Envelope::AddToShape(), Envelope::DefineShape(), Envelope::Envelope(), Envelope::getPoint(), Envelope::getPoints(), Envelope::multiply(), Envelope::Print(), Envelope::setPoint(), Envelope::setSegment(), Envelope::xml_print(), and Envelope::xml_read().

8.15.2.7 m_value_type envelope_segment::y

y value at the right endpoint

Definition at line 102 of file Types.h.

Referenced by Envelope::addInterpolators(), Envelope::addPoint(), Envelope::Envelope(), Envelope::getMaxValue(), Envelope::getPoint(), Envelope::getPoints(), Envelope::getValue(), Envelope::multiply(), Envelope::Print(), Envelope::scale(), Envelope::setPoint(), Envelope::xml_print(), and Envelope::xml_read().

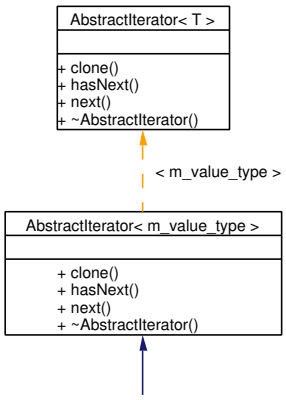
The documentation for this struct was generated from the following file:

- Types.h

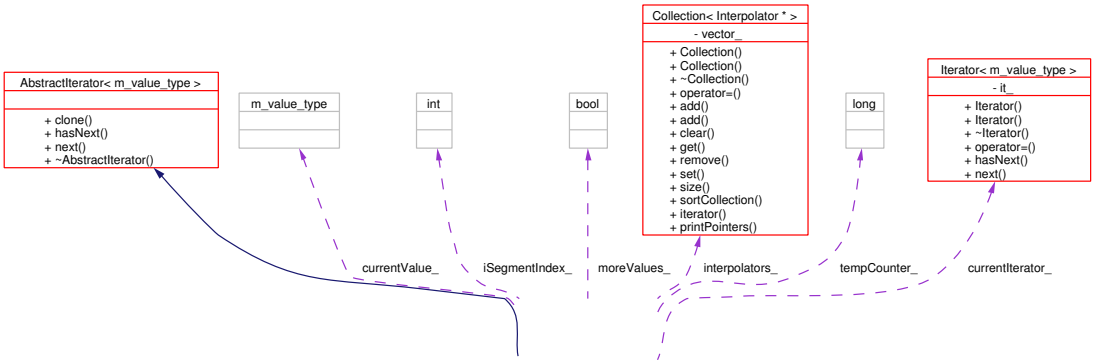
8.16 EnvelopeIterator Class Reference

`#include <EnvelopeIterator.h>`

Inheritance diagram for EnvelopeIterator:



Collaboration diagram for EnvelopeIterator:



Public Member Functions

- EnvelopeIterator (Collection< Interpolator * > interpolators)
- ~EnvelopeIterator ()
- EnvelopeIterator * clone ()
- bool hasNext ()
- m_value_type & next ()

Private Attributes

- Collection< Interpolator * > * interpolators_
- int iSegmentIndex_
- Iterator< m_value_type > * currentIterator_
- m_value_type currentValue_
- bool moreValues_
- long tempCounter_

8.16.1 Detailed Description

An iterator that interpolates over a Envelope. Since a Envelope acts like a single DynamicVariable, when we want to use it as such, we need a way to iterate through its value. That's what this does.

Author:
Michael Aikins

Definition at line 48 of file EnvelopeIterator.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 EnvelopeIterator::EnvelopeIterator (Collection< Interpolator * > *interpolators*)

This is the constructor which initalizes some basic values.

Parameters:
interpolators

Definition at line 37 of file EnvelopeIterator.cpp.

References currentIterator_, currentValue_, Collection< InterpolatorEntry >::get(), Collection< Interpolator * >::get(), interpolators_, iSegmentIndex_, moreValues_, Collection< InterpolatorEntry >::size(), Collection< Interpolator * >::size(), Collection< T >::size(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

Referenced by clone().

8.16.2.2 EnvelopeIterator::~EnvelopeIterator ()

This is the Destructor.

Definition at line 81 of file EnvelopeIterator.cpp.

References currentIterator_, and interpolators_.

8.16.3 Member Function Documentation

8.16.3.1 EnvelopeIterator * EnvelopeIterator::clone () [virtual]

This makes a copy of this iterator.

Returns:
A copy of this iterator.

Implements AbstractIterator< m_value_type >.

Definition at line 100 of file EnvelopeIterator.cpp.

References EnvelopeIterator().

8.16.3.2 bool EnvelopeIterator::hasNext () [virtual]

Indicate whether the iterator has another value to return.

Return values:
true If there is another value to return

false If there is not another value to return

Implements AbstractIterator< m_value_type >.

Definition at line 107 of file EnvelopeIterator.cpp.

References moreValues_.

8.16.3.3 m_value_type & EnvelopeIterator::next () [virtual]

Gets the next value in the iteration.

Note:
Because this returns a reference type, value_ can be changed by the caller. Steps should be taken to prevent this (with a pass-to-caller member variable perhaps).

Returns:
a reference to the next value in the iteration

Implements AbstractIterator< m_value_type >.

Definition at line 115 of file EnvelopeIterator.cpp.

References currentIterator_, currentValue_, Collection< Interpolator * >::get(), Iterator< m_value_type >::hasNext(), interpolators_, iSegmentIndex_, m_value_type, moreValues_, Iterator< m_value_type >::next(), Collection< Interpolator * >::size(), tempCounter_, and Interpolator::valueIterator().

8.16.4 Member Data Documentation

8.16.4.1 Iterator<m_value_type>* EnvelopeIterator::currentIterator_ [private]

This is the current DV iterator.

Definition at line 99 of file EnvelopeIterator.h.

Referenced by EnvelopeIterator(), next(), and ~EnvelopeIterator().

8.16.4.2 m_value_type EnvelopeIterator::currentValue_ [private]

This is the current value.

Definition at line 104 of file EnvelopeIterator.h.

Referenced by EnvelopeIterator(), and next().

8.16.4.3 Collection<Interpolator*>* EnvelopeIterator::interpolators_ [private]

This is a collection that holds the interpolators for all the segments.

Definition at line 89 of file EnvelopeIterator.h.

Referenced by EnvelopeIterator(), next(), and ~EnvelopeIterator().

8.16.4.4 int EnvelopeIterator::iSegmentIndex_ [private]

This is the current segment's index.

Definition at line 94 of file EnvelopeIterator.h.

Referenced by EnvelopeIterator(), and next().

8.16.4.5 bool EnvelopeIterator::moreValues_ [private]

This indicates whether we have more values.

Definition at line 109 of file EnvelopeIterator.h.

Referenced by EnvelopeIterator(), hasNext(), and next().

8.16.4.6 long EnvelopeIterator::tempCounter_ [private]

Definition at line 111 of file EnvelopeIterator.h.

Referenced by next().

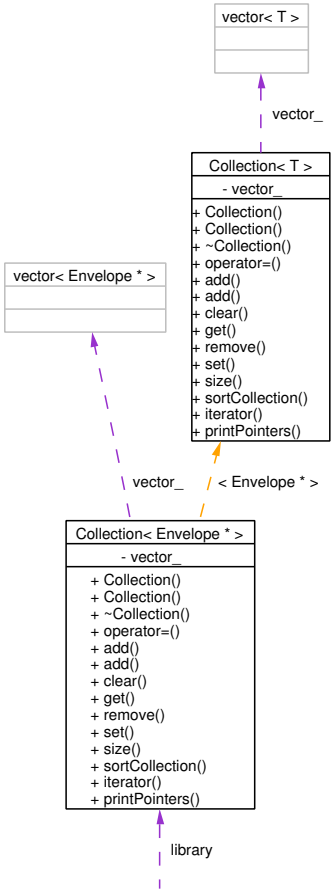
The documentation for this class was generated from the following files:

- EnvelopeIterator.h
- EnvelopeIterator.cpp

8.17 EnvelopeLibrary Class Reference

`#include <EnvelopeLibrary.h>`

Collaboration diagram for EnvelopeLibrary:



Public Member Functions

- EnvelopeLibrary ()
- ~EnvelopeLibrary ()
- EnvelopeLibrary (EnvelopeLibrary &lib)
- EnvelopeLibrary & operator= (EnvelopeLibrary &lib)
- bool saveLibrary (char *filename)
- int loadLibrary (char *filename)
- Envelope * getEnvelope (int index)
- int addEnvelope (Envelope *env)
- int addEnvelope (Collection< xy_point > points, Collection< envelope_segment > segments)
- bool updateEnvelope (int index, Envelope *env)
- void showEnvelope (int index)
- int size ()

Private Attributes

- Collection< Envelope * > library

8.17.1 Detailed Description

This class manages a collection of Envelopes, providing methods for loading and saving the collection to a library file as well as functionality for adding to, retrieving from, updating the collection with, and displaying individual Envelope items. The name Envelope is used informally to designate an Envelope.

Note:
All methods in this class that take an index argument begin with a first value of 1 for user ease, though the actual Collection indices begin at 0.

Author:
Philipp Fraund

Definition at line 48 of file EnvelopeLibrary.h.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 EnvelopeLibrary::EnvelopeLibrary ()

This is the Constructor.

Definition at line 42 of file EnvelopeLibrary.cpp.

8.17.2.2 EnvelopeLibrary::~EnvelopeLibrary ()

This is the destructor. It deallocates all the Envelopes stored in the library.

Definition at line 46 of file EnvelopeLibrary.cpp.

References library, Collection< Envelope * >::remove(), and Collection< Envelope * >::size().

8.17.2.3 EnvelopeLibrary::EnvelopeLibrary (EnvelopeLibrary & *lib*)

This is a copy constructor.

Parameters:

lib The EnvelopeLibrary to copy

Returns:

A copy of lib

Definition at line 54 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::add(), Envelope::clone(), Collection< Envelope * >::get(), library, and Collection< Envelope * >::size().

8.17.3 Member Function Documentation

8.17.3.1 int EnvelopeLibrary::addEnvelope (Collection< xy_point > *points*, Collection< envelope_segment > *segments*)

This function adds a Collection of n+1 points and a Collection of n segments to the EnvelopeLibrary and returns the index at which it has been added.

Parameters:

points A Collection of n+1 points

segments A Collection of n segments

Returns:

The index where they were added

Definition at line 268 of file EnvelopeLibrary.cpp.

References addEnvelope().

8.17.3.2 int EnvelopeLibrary::addEnvelope (Envelope * *env*)

This function adds an Envelope (passed by pointer) to the EnvelopeLibrary and returns the index at which it has been added.

Parameters:
env The new Envelope to add to the EnvelopeLibrary

Returns:
The index of the new Envelope in the EnvelopeLibrary

Definition at line 260 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::add(), library, and Collection< Envelope * >::size().

Referenced by addEnvelope().

8.17.3.3 Envelope * EnvelopeLibrary::getEnvelope (int *index*)

This function returns a pointer to a new Envelope given an index to an existing EnvelopeLibrary, or NULL if index is out of range.

Note:
This returns a clone of the env and not a pointer to the library object itself

Parameters:
index Which Envelope in the EnvelopeLibrary to find

Returns:
A pointer to a COPY of the original envelope

Definition at line 250 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::get(), library, and Collection< Envelope * >::size().

8.17.3.4 int EnvelopeLibrary::loadLibrary (char * *filename*)

This function reads the Envelope library from a disk file.

Parameters:
filename The name of the file to read from

Returns:
The number of entries in the library of -1 if unsucessful

Definition at line 153 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::add(), Collection< T >::add(), Collection< T >::clear(), CUBIC_SPLINE, EXPONENTIAL, FIXED, FLEXIBLE, interpolation_type, envelope_segment::interType, envelope_segment::length, envelope_segment::lengthType, library, LINEAR, m_time_type, Collection< Envelope * >::size(), stretch_type, xy_point::x, and xy_point::y.

8.17.3.5 EnvelopeLibrary & EnvelopeLibrary::operator= (EnvelopeLibrary & *lib*)

This is an overloaded assignment operator for EvelopeLibraries

Definition at line 62 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::add(), Collection< Envelope * >::get(), library, Collection< Envelope * >::remove(), and Collection< Envelope * >::size().

8.17.3.6 bool EnvelopeLibrary::saveLibrary (char * *filename*)

This function writes the Envelope library to a disk file.

Parameters:

filename The name of the file to write to

Return values:

true If successful

false If unsuccessful

Definition at line 80 of file EnvelopeLibrary.cpp.

References CUBIC_SPLINE, EXPONENTIAL, FLEXIBLE, Collection< Envelope * >::get(), library, LINEAR, size(), and Collection< Envelope * >::size().

8.17.3.7 void EnvelopeLibrary::showEnvelope (int *index*)

This function displays the indexed Envelope's contents using the standard output stream.

Parameters:

index The Envelope in the EvelopeLibrary to show

Definition at line 291 of file EnvelopeLibrary.cpp.

References Collection< Envelope * >::get(), library, and Collection< Envelope * >::size().

8.17.3.8 int EnvelopeLibrary::size ()

This function returns the number of envelopes existing in the EnvelopeLibrary

Returns:
The number of envelopes

Definition at line 306 of file EnvelopeLibrary.cpp.

References library, and Collection< Envelope * >::size().

Referenced by saveLibrary().

8.17.3.9 bool EnvelopeLibrary::updateEnvelope (int *index*, Envelope * *env*)

This function updates the EnvelopeLibrary entry at the index with an Envelope (passed by pointer)

Parameters:
index Which Envelope in the EnvelopeLibrary to update
env A pointer to the new Envelope

Return values:
true Successful
false Unsuccessful (index out of range)

Definition at line 278 of file EnvelopeLibrary.cpp.

References library, Collection< Envelope * >::set(), and Collection< Envelope * >::size().

8.17.4 Member Data Documentation

8.17.4.1 Collection<Envelope *> EnvelopeLibrary::library [private]

This is a data structure to hold the library of envelopes.

Definition at line 152 of file EnvelopeLibrary.h.

Referenced by addEnvelope(), EnvelopeLibrary(), getEnvelope(), loadLibrary(), operator=(), saveLibrary(), showEnvelope(), size(), updateEnvelope(), and ~EnvelopeLibrary().

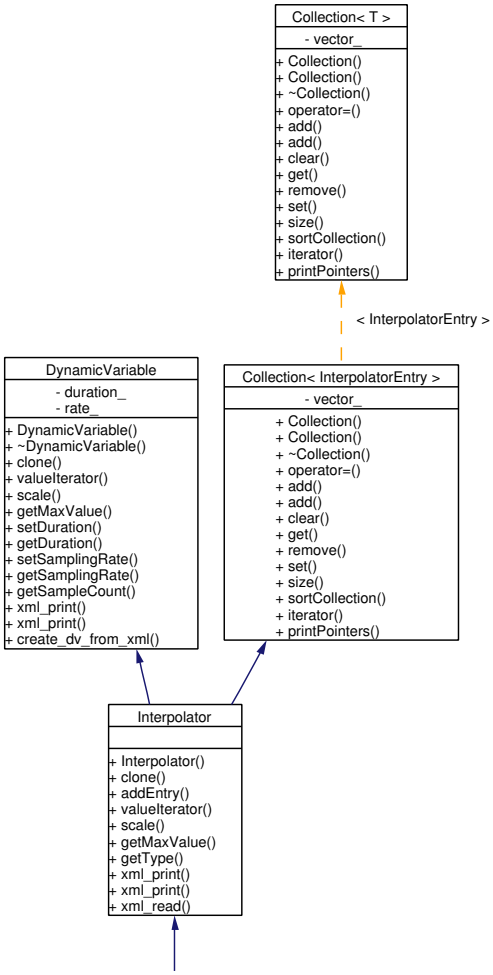
The documentation for this class was generated from the following files:

- EnvelopeLibrary.h
- EnvelopeLibrary.cpp

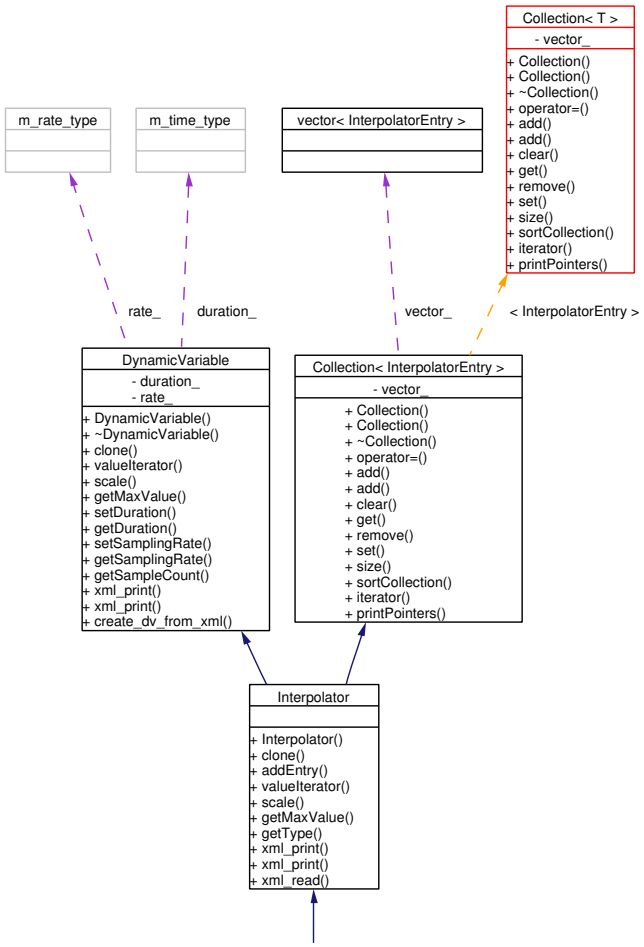
8.18 ExponentialInterpolator Class Reference

`#include <InterpolatorTypes.h>`

Inheritance diagram for ExponentialInterpolator:



Collaboration diagram for ExponentialInterpolator:



Public Member Functions

- ExponentialInterpolator ()
- ExponentialInterpolator * clone ()
- Iterator< m_value_type > valueIterator ()
- virtual interpolation_type getType ()

8.18.1 Detailed Description

This is a DynamicVariable that changes over time. This exponentially interpolates between a set of points ordered in time.

Author:
Braden Kowitz
Philipp Fraund

Definition at line 81 of file InterpolatorTypes.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 ExponentialInterpolator::ExponentialInterpolator ()

This is the default constructor

Definition at line 117 of file InterpolatorTypes.cpp.

Referenced by clone().

8.18.3 Member Function Documentation

8.18.3.1 ExponentialInterpolator * ExponentialInterpolator::clone () [virtual]

This makes a clone of a ExponentialInterpolator.

Returns:
A new ExponentialInterpolator

Implements Interpolator.

Definition at line 122 of file InterpolatorTypes.cpp.

References ExponentialInterpolator().

8.18.3.2 interpolation_type ExponentialInterpolator::getType () [virtual]

This provides an implementation to get the type of interpolator.

Returns:
The interpolation type

Implements Interpolator.

Definition at line 189 of file InterpolatorTypes.cpp.

References EXPONENTIAL, and interpolation_type.

8.18.3.3 Iterator< m_value_type > ExponentialInterpolator::valueIterator () [virtual]

This creates an iterator over ExponentialInterpolators.

Returns:
An iterator

Implements Interpolator.

Definition at line 131 of file InterpolatorTypes.cpp.

References InterpolatorIterator::append(), Collection< InterpolatorEntry >::get(), DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), m_sample_count_type, m_time_type, m_value_type, Iterator< T >::next(), Collection< InterpolatorEntry >::size(), Collection< InterpolatorEntry >::sortCollection(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

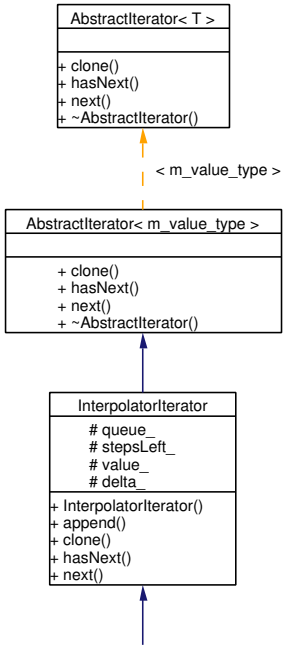
The documentation for this class was generated from the following files:

- InterpolatorTypes.h
- InterpolatorTypes.cpp

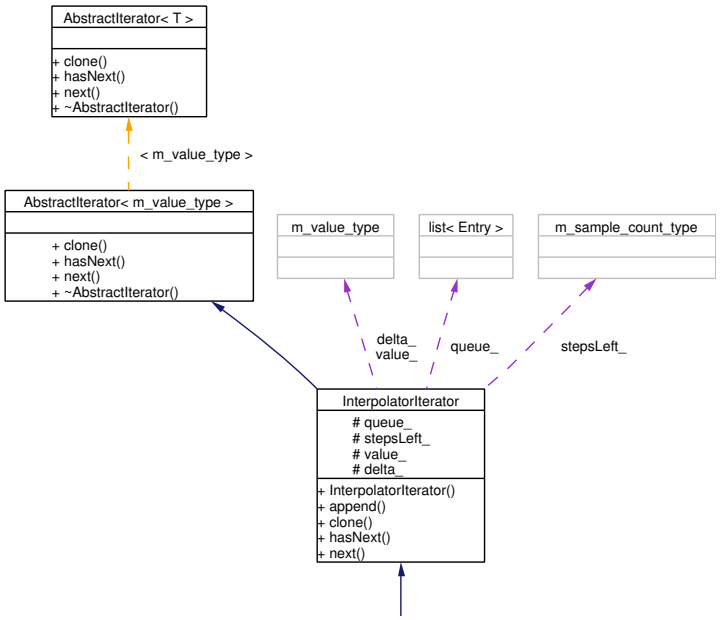
8.19 ExponentialInterpolatorIterator Class Reference

`#include <InterpolatorIterator.h>`

Inheritance diagram for ExponentialInterpolatorIterator:



Collaboration diagram for ExponentialInterpolatorIterator:



Public Member Functions

- ExponentialInterpolatorIterator ()

  constructor for iterator

- ExponentialInterpolatorIterator * clone ()

  make a clone of the iterator

- m_value_type & next ()

  get the next value in the iterator

8.19.1 Detailed Description

This is an interator that will interate over values in an ExponentialInterpolator.

Definition at line 170 of file InterpolatorIterator.h.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 ExponentialInterpolatorIterator::ExponentialInterpolatorIterator ()

constructor for iterator

Definition at line 111 of file InterpolatorIterator.cpp.

Referenced by clone().

8.19.3 Member Function Documentation

8.19.3.1 ExponentialInterpolatorIterator * ExponentialInterpolatorIterator::clone () [virtual]

make a clone of the iterator

Implements InterpolatorIterator.

Definition at line 115 of file InterpolatorIterator.cpp.

References ExponentialInterpolatorIterator().

8.19.3.2 m_value_type & ExponentialInterpolatorIterator::next () [virtual]

get the next value in the iterator

Implements InterpolatorIterator.

Definition at line 150 of file InterpolatorIterator.cpp.

References m_time_type, m_value_type, InterpolatorIterator::queue_, InterpolatorIterator::stepsLeft_, and InterpolatorIterator::value_.

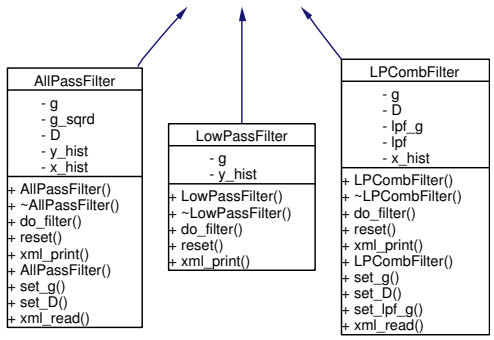
The documentation for this class was generated from the following files:

- InterpolatorIterator.h
- InterpolatorIterator.cpp

8.20 Filter Class Reference

`#include <Filter.h>`

Inheritance diagram for Filter:



Public Member Functions

- MultiTrack & do_filter_MultiTrack (MultiTrack &inWave)
- Track & do_filter_Track (Track &inWave)
- SoundSample * do_filter_SoundSample (SoundSample *inWave)
- virtual m_sample_type do_filter (m_sample_type x_t)=0
- virtual void reset (void)=0

8.20.1 Detailed Description

The filter class is a pure virtual class only meant to ensure that all filters have a common interface. The idea is that a collection of filters, specified by the user at run-time, perhaps, could be applied one after the next without regard to what type of filter each one is.

Each filter will likely have its own configuration that can be done, and if, in the future, this becomes something that needs to be generalized as well, perhaps this class could be expanded to include a ParameterLib.

Note:

The Filter objects are stateful machines with internal feedback mechanisms. Thus

this filter should be allocated anew each time you begin a new channel and should not be mixed between channels.

Author:
Jim Lindstrom

Definition at line 52 of file Filter.h.

8.20.2 Member Function Documentation

8.20.2.1 virtual m_sample_type Filter::do_filter (m_sample_type *x_t*) [pure virtual]

This method should be redefined by each class derived from Filter to perform the actual filtering. It should take in a single sample and return the filtered sample. If, in the future, real-time filtering should become feasible and is desired, this is the entry-point to use.

Parameters:
x_t A sample to filter

Returns:
the filtered sample

Implemented in AllPassFilter, LowPassFilter, and LPCombFilter.

Referenced by do_filter_SoundSample().

8.20.2.2 MultiTrack & Filter::do_filter_MultiTrack (MultiTrack & *inWave*)

This method applies the filter to a MultiTrack source, track by track. It does so by decomposing the wave into Track objects (organized as a Collection inside the MultiTrack object) and calling the virtual function, do_filter(SoundSample *inWave), for each Track.

Parameters:
inWave A reference to a MultiTrack object to Filter

Returns:
A reference to a NEW MultiTrack, which the caller is responsible for deleting when done. The original MultiTrack remains intact and untouched

Definition at line 40 of file Filter.cpp.

References Collection< Track * >::add(), do_filter_SoundSample(), Iterator< T >::hasNext(), Collection< Track * >::iterator(), Iterator< T >::next(), and reset().

8.20.2.3 SoundSample * Filter::do_filter_SoundSample (SoundSample * *inWave*)

This method calls do_filter(m_sample_type x) for each sample in a track and builds a new SoundSample on the fly.

Parameters:

inWave A pointer to a SoundSample object to Filter

Returns:

A reference to a NEW MultiTrack, which the caller is responsible for deleting when done. The original Track remains intact and untouched

Definition at line 84 of file Filter.cpp.

References do_filter(), SoundSample::getSampleCount(), and SoundSample::getSamplingRate().

Referenced by do_filter_MultiTrack(), and do_filter_Track().

8.20.2.4 Track & Filter::do_filter_Track (Track & *inWave*)

This method applies the filter to a source Track by calling the virtual function, do_filter(SoundSample *inWave).

Parameters:

inWave A reference to a Track object to Filter

Returns:

A reference to a NEW MultiTrack, which the caller is responsible for deleting when done. The original Track remains intact and untouched.

Definition at line 69 of file Filter.cpp.

References do_filter_SoundSample(), and Track::getWave().

8.20.2.5 virtual void Filter::reset (void) [pure virtual]

This method should be redefined by each class derived from Filter to reset the filter to an initial state. It should have the same effect as deleting the filter and creating a new one.

Implemented in AllPassFilter, LowPassFilter, and LPCombFilter.

Referenced by do_filter_MultiTrack().

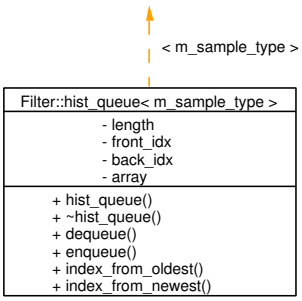
The documentation for this class was generated from the following files:

- Filter.h
- Filter.cpp

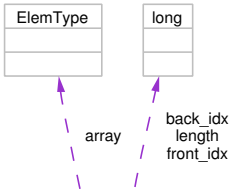
8.21 Filter::hist_queue< ElemType > Class Template Reference

`#include <Filter.h>`

Inheritance diagram for Filter::hist_queue< ElemType >:



Collaboration diagram for Filter::hist_queue< ElemType >:



Public Member Functions

- hist_queue (long len)
- ~hist_queue ()
- ElemType dequeue (void)
- void enqueue (ElemType new_val)
- ElemType index_from_oldest (long i)
- ElemType index_from_newest (long i)

Private Attributes

- long length
- long front_idx
- long back_idx
- ElemType * array

8.21.1 Detailed Description

template<class ElemType> class Filter::hist_queue< ElemType >

Every filter requires some sort of history of previous values. Even the simplest 1st order IIR filter requires the previous output. For filters, we only need one of the previous values. In that case, a queue structure is useful to enqueue the current output for later use, and to dequeue a value from previous runs.

Note 1: The C++ STL includes a queue, but (1) it uses nonstandard 'push' and 'pop' instead of 'enqueue' and 'dequeue' (respectively), which decreases readability; (2) in filters with a fixed delay factor, we can optimize for speed by using a fixed-length ring buffer of the appropriate size (let queue length = D). Note 2: Some filters, such as FIR filters require a different type of history that needs access to all the previous values (or some subset of them that includes more than just the oldest value). For these filters, this queue allows indexed access from either end of the queue.

Three examples: 1. Consider the 1st order IIR filter. It uses the equation y(t) = x(t) + g*y(t-1). In this case, the history queue will always have one old value. Thus, we must inititalize it by enqueing a 0 at the start. Then we can perform one time step with as follows: y_t = x_t + (g * hist_queue.dequeue()); hist_queue.enqueue(y_t); return y_t;

2. Now compare this to the comb filter. It uses the equation y(t) = x(t-D) + g*y(t-D). Now we need two queues. This is the reason that the queue class is provided, but not instantiated. Child-class filters can use the history queue however they like as needed. init: for(i=0;i<D;i++) { x_hist.enqueue(0); y_hist.enqueue(0); }

y_d = x_hist.dequeue() + (g * y_hist.dequeue()); y_hist.enqueue(y_d); x_hist.enqueue(x_d);

Note 3: If you're using a D-length queue for a delay of D cycles, be sure that you dequeue before enqueuing so you don't overflow the queue.

Author:
Jim Lindstrom

Definition at line 159 of file Filter.h.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 template<class ElemType> Filter::hist_queue< ElemType >::hist_queue (long *len*) [inline]

This constructor initializes the queue to a given length.

Parameters:
len The length of the queue

Definition at line 168 of file Filter.h.

8.21.2.2 template<class ElemType> Filter::hist_queue< ElemType >::~hist_queue () [inline]

This is the destructor.

Definition at line 182 of file Filter.h.

8.21.3 Member Function Documentation

8.21.3.1 template<class ElemType> ElemType Filter::hist_queue< ElemType >::dequeue (void) [inline]

This function removes and returns the oldest element in the queue (FIFO).

Returns:
The oldest element

Definition at line 192 of file Filter.h.

8.21.3.2 template<class ElemType> void Filter::hist_queue< ElemType >::enqueue (ElemType *new_val*) [inline]

This function adds an element to the queue (FIFO).

Parameters:
new_val The element to add to the queue

Definition at line 206 of file Filter.h.

8.21.3.3 template<class ElemType> ElemType Filter::hist_queue< ElemType >::index_from_newest (long *i*) [inline]

Return, without removing, a value from the queue indexed from the newest element.

Parameters:
i The index from the end of the queue to return

Definition at line 237 of file Filter.h.

8.21.3.4 template<class ElemType> ElemType Filter::hist_queue< ElemType >::index_from_oldest (long *i*) [inline]

This function returns, without removing, a value from the queue indexed from the oldest element

Parameters:
i The index from the end of the queue to return

Definition at line 218 of file Filter.h.

8.21.4 Member Data Documentation

8.21.4.1 template<class ElemType> ElemType* Filter::hist_queue< ElemType >::array [private]

This holds the values of the queue.

Definition at line 270 of file Filter.h.

Referenced by Filter::hist_queue< m_sample_type >::dequeue(), Filter::hist_queue< m_sample_type >::enqueue(), Filter::hist_queue< m_sample_type >::hist_queue(), Filter::hist_queue< m_sample_type >::index_from_newest(), Filter::hist_queue< m_sample_type >::index_from_oldest(), and Filter::hist_queue< m_sample_type >::~hist_queue().

8.21.4.2 template<class ElemType> long Filter::hist_queue< ElemType >::back_idx [private]

This is the index TO the oldest item.

Definition at line 265 of file Filter.h.

Referenced by Filter::hist_queue< m_sample_type >::dequeue(), Filter::hist_queue< m_sample_type >::hist_queue(), and Filter::hist_queue< m_sample_type >::index_from_oldest().

8.21.4.3 template<class ElemType> long Filter::hist_queue< ElemType >::front_idx [private]

This is the index PAST the newest item.

Definition at line 260 of file Filter.h.

Referenced by Filter::hist_queue< m_sample_type >::enqueue(), Filter::hist_queue< m_sample_type >::hist_queue(), and Filter::hist_queue< m_sample_type >::index_from_newest().

8.21.4.4 template<class ElemType> long Filter::hist_queue< ElemType >::length [private]

This is the number of valid positions within the queue.

Definition at line 255 of file Filter.h.

Referenced by Filter::hist_queue< m_sample_type >::dequeue(), Filter::hist_queue< m_sample_type >::enqueue(), Filter::hist_queue< m_sample_type >::hist_queue(), Filter::hist_queue< m_sample_type >::index_from_newest(), and Filter::hist_queue< m_sample_type >::index_from_oldest().

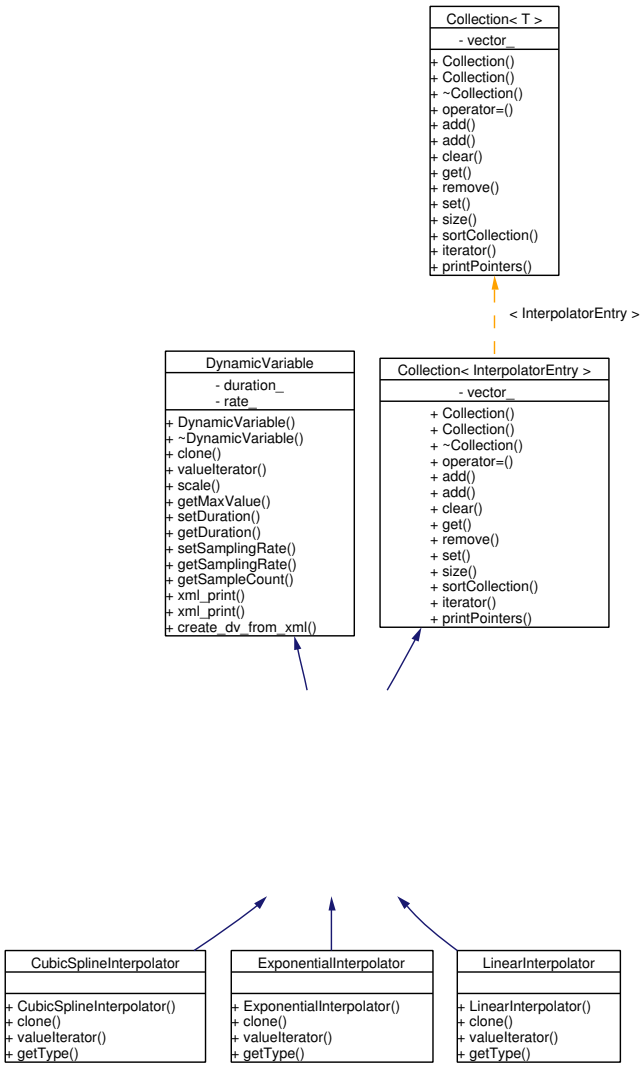
The documentation for this class was generated from the following file:

- Filter.h

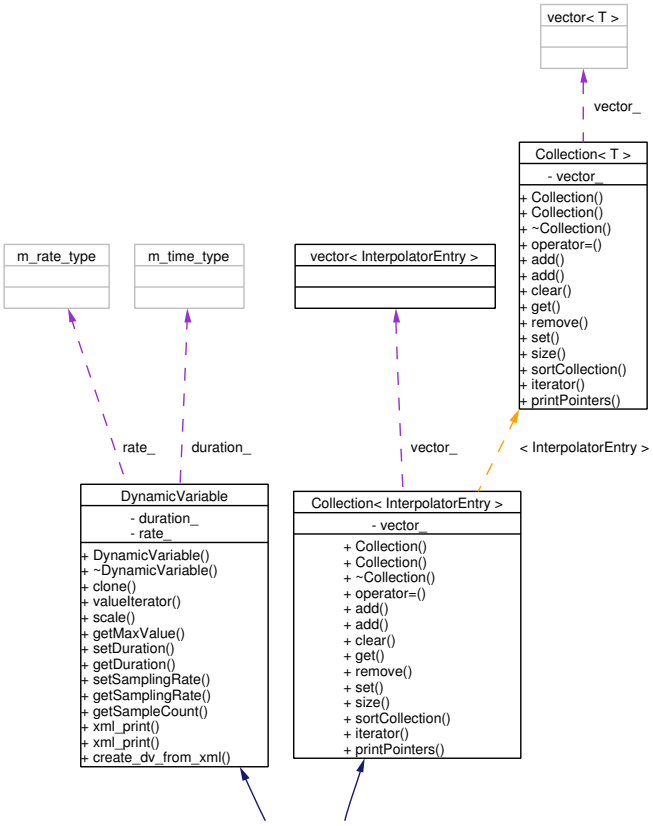
8.22 Interpolator Class Reference

`#include <Interpolator.h>`

Inheritance diagram for Interpolator:



Collaboration diagram for Interpolator:



Public Member Functions

- Interpolator ()
- virtual Interpolator * clone ()=0
- void addEntry (m_time_type time, m_value_type value)
- virtual Iterator< m_value_type > valueIterator ()=0
- void scale (m_value_type scale)
- m_value_type getMaxValue ()
- virtual interpolation_type getType ()=0
- void xml_print (ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)

- void xml_print (ofstream &xmlOutput)
- void xml_read (XmlReader::xmltag *soundtag)

8.22.1 Detailed Description

This is an abstract interpolator class from which more specific interpolators (DynamicVariables used for interpolating between points) can be derived.

See InterpolatorTypes.h for interpolators.

Author:
Zeke McKinney
Braden Kowitz
Philipp Fraund

Definition at line 108 of file Interpolator.h.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 Interpolator::Interpolator ()

This is the default constructor.

Definition at line 36 of file Interpolator.cpp.

8.22.3 Member Function Documentation

8.22.3.1 void Interpolator::addEntry (m_time_type *time*, m_value_type *value*)

This Adds an entry to this variable. This is added for convenience - this way, a user does not have to deal with InterpolatorEntry objects when simply creating an Interpolator.

Parameters:
time The time of the new entry

value The value of the new entry

Definition at line 42 of file Interpolator.cpp.

References Collection< InterpolatorEntry >::add(), m_time_type, and m_value_type.

Referenced by Envelope::addInterpolators(), DynamicVariableSequence::addInterpolators(), Envelope::getValue(), DynamicVariableSequence::getValue(), and Sound::setup_detuning_env().

8.22.3.2 virtual Interpolator* Interpolator::clone () [pure virtual]

This function creates an exact copy of this object.

Returns:
A new Interpolator

Implements DynamicVariable.

Implemented in LinearInterpolator, ExponentialInterpolator, and CubicSpline-Interpolator.

8.22.3.3 m_value_type Interpolator::getMaxValue () [virtual]

This function returns the maximum value of all the entries.

Returns:
The maximum value

Implements DynamicVariable.

Definition at line 63 of file Interpolator.cpp.

References Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), m_-value_type, and Iterator< T >::next().

8.22.3.4 virtual interpolation_type Interpolator::getType () [pure virtual]

This function returns the type of interpolator for use in reconstructing envelope parameters for EnvelopeLibrary from a DynamicVariableSequence. Virtual function implemented in InterpolatorTypes.

Returns:
Interpolator type

Implemented in LinearInterpolator, ExponentialInterpolator, and CubicSpline-Interpolator.

Referenced by xml_print().

8.22.3.5 void Interpolator::scale (m_value_type *scale*) [virtual]

This function scales every entry's value by this factor.

Parameters:
scale The factor by which to scale

Implements DynamicVariable.

Definition at line 51 of file Interpolator.cpp.

References Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), m_value_type, and Iterator< T >::next().

8.22.3.6 virtual Iterator<m_value_type> Interpolator::valueIterator () [pure virtual]

This function returns an InterpolatorIterator.

Returns:
An iterator

Implements DynamicVariable.

Implemented in LinearInterpolator, ExponentialInterpolator, and CubicSplineInterpolator.

Referenced by Envelope::getValue(), DynamicVariableSequence::getValue(), EnvelopeIterator::next(), and DynamicVariableSequenceIterator::next().

8.22.3.7 void Interpolator::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

Implements DynamicVariable.

Definition at line 79 of file Interpolator.cpp.

References DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), getType(), Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), Iterator< T >::next(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

8.22.3.8 void Interpolator::xml_print (ofstream & *xmlOutput*, list< DynamicVariable * > & *dynObjs*) [virtual]

Deprecated
This outputs an XML representation of the object to STDOUT

Implements DynamicVariable.

Definition at line 104 of file Interpolator.cpp.

8.22.3.9 void Interpolator::xml_read (XmlReader::xmltag * *soundtag*)

Deprecated

Definition at line 126 of file Interpolator.cpp.

References Collection< InterpolatorEntry >::add(), XmlReader::xmltag::children, XmlReader::xmltag::findChildParamValue(), XmlReader::xmltag::getParamValue(), DynamicVariable::setDuration(), and DynamicVariable::setSamplingRate().

Referenced by DynamicVariable::create_dv_from_xml().

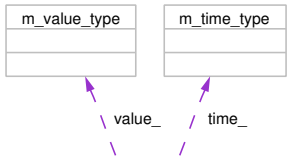
The documentation for this class was generated from the following files:

- Interpolator.h
- Interpolator.cpp

8.23 InterpolatorEntry Class Reference

`#include <Interpolator.h>`

Collaboration diagram for InterpolatorEntry:



Public Member Functions

- InterpolatorEntry ()
- InterpolatorEntry (m_time_type t, m_value_type v)
- bool operator< (const InterpolatorEntry &ie) const
- bool operator== (const InterpolatorEntry &ie) const
- bool operator> (const InterpolatorEntry &ie) const

Public Attributes

- m_time_type time_
- m_value_type value_

8.23.1 Detailed Description

This is a single entry for an interpolator.

Author:
Zeke McKinney
Braden Kowitz
Philipp Fraund

Definition at line 47 of file Interpolator.h.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 InterpolatorEntry::InterpolatorEntry () [inline]

This is a constructor that makes an entry with no parameters.

Definition at line 53 of file Interpolator.h.

8.23.2.2 InterpolatorEntry::InterpolatorEntry (m_time_type *t*, m_value_type *v*) [inline]

This is a constructor for convenience.

Definition at line 58 of file Interpolator.h.

References m_time_type, m_value_type, time_, and value_.

8.23.3 Member Function Documentation

8.23.3.1 bool InterpolatorEntry::operator< (const InterpolatorEntry & *ie*) const [inline]

This is an overloaded less than operator

Definition at line 77 of file Interpolator.h.

References time_.

8.23.3.2 bool InterpolatorEntry::operator== (const InterpolatorEntry & *ie*) const [inline]

This is an overloaded equal to operator

Definition at line 84 of file Interpolator.h.

References time_.

8.23.3.3 bool InterpolatorEntry::operator> (const InterpolatorEntry & *ie*) const [inline]

This is an overloaded greater than operator

Definition at line 91 of file Interpolator.h.

References time_.

8.23.4 Member Data Documentation

8.23.4.1 m_time_type InterpolatorEntry::time_

The time for this entry.

Definition at line 66 of file Interpolator.h.

Referenced by DynamicVariableSequenceIterator::DynamicVariableSequenceIterator(), EnvelopeIterator::EnvelopeIterator(), InterpolatorEntry(), operator<(), operator==(), operator>(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), and Interpolator::xml_print().

8.23.4.2 m_value_type InterpolatorEntry::value_

The value for this entry.

Definition at line 71 of file Interpolator.h.

Referenced by DynamicVariableSequenceIterator::DynamicVariableSequenceIterator(), EnvelopeIterator::EnvelopeIterator(), InterpolatorEntry(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), and Interpolator::xml_print().

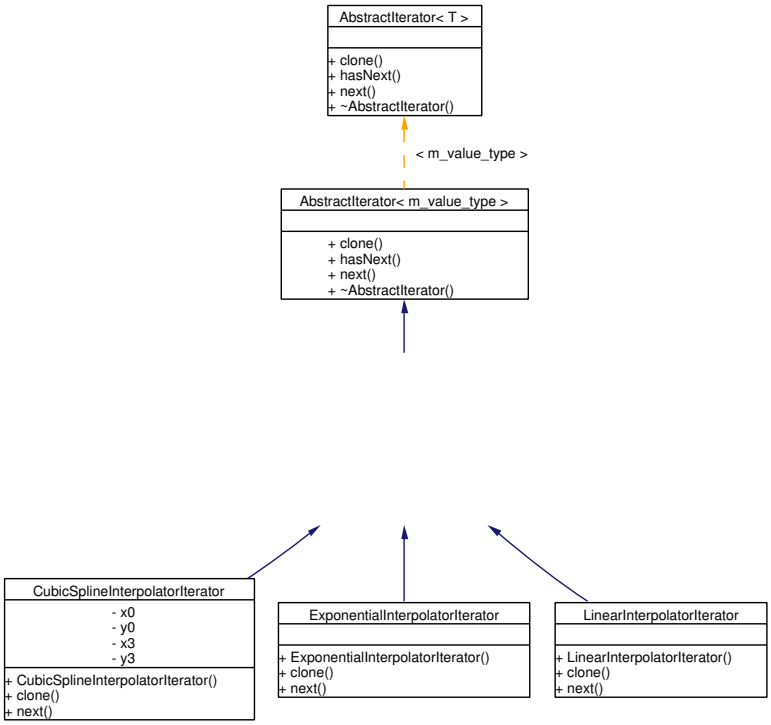
The documentation for this class was generated from the following file:

- Interpolator.h

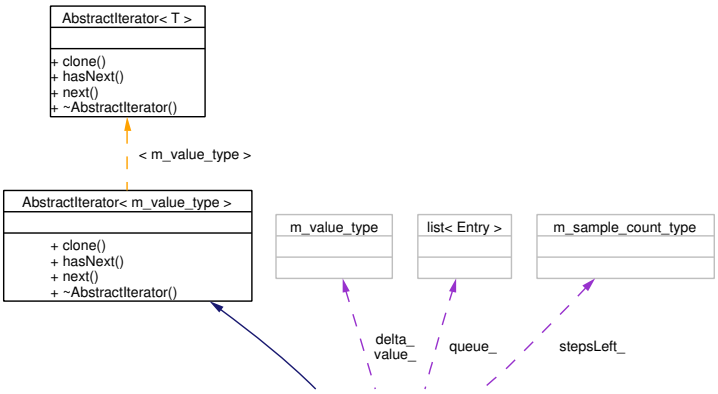
8.24 InterpolatorIterator Class Reference

`#include <InterpolatorIterator.h>`

Inheritance diagram for InterpolatorIterator:



Collaboration diagram for InterpolatorIterator:



Public Member Functions

- InterpolatorIterator ()
- void append (m_time_type t_from, m_time_type t_to, m_value_type v_from, m_value_type v_to, m_sample_count_type steps)
- virtual InterpolatorIterator * clone ()=0
- bool hasNext ()
- virtual m_value_type & next ()=0

Protected Attributes

- list< Entry > queue_
- m_sample_count_type stepsLeft_
- m_value_type value_
- m_value_type delta_

8.24.1 Detailed Description

This is an iterator that interpolates between set values. Users append entries of from-value to-value and num-steps. The iterator will then provide iteratation over those ranges.

Author:
Zeke McKinney

Definition at line 46 of file InterpolatorIterator.h.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 InterpolatorIterator::InterpolatorIterator ()

This is a constructor which initalizes some basic values.

Definition at line 35 of file InterpolatorIterator.cpp.

8.24.3 Member Function Documentation

8.24.3.1 void InterpolatorIterator::append (m_time_type *t_from*, m_time_type *t_to*, m_value_type *v_from*, m_value_type *v_to*, m_sample_count_type *steps*)

This defines a linear segment to append to this iterator.

Note:
This interface makes discontinuities possible.

Todo
Perhaps for future versions: void append(const InterpolatorIterator&);

Parameters:
t_from The start time

t_to The end time

v_from The beginning value

v_to The ending value

steps The number of steps to take

Definition at line 42 of file InterpolatorIterator.cpp.

References m_sample_count_type, m_time_type, m_value_type, and queue_.

Referenced by CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), and LinearInterpolator::valueIterator().

8.24.3.2 virtual InterpolatorIterator* InterpolatorIterator::clone () [pure virtual]

This makes a copy of the this iterator.

Returns:
A copy of the iterator

Implements AbstractIterator< m_value_type >.

Implemented in LinearInterpolatorIterator, ExponentialInterpolatorIterator, and CubicSplineInterpolatorIterator.

8.24.3.3 bool InterpolatorIterator::hasNext () [virtual]

Indicates whether there is another value to get.

Return values:
true If there is another value to return.
false If there is no other value to return.

Implements AbstractIterator< m_value_type >.

Definition at line 55 of file InterpolatorIterator.cpp.

References queue_, and stepsLeft_.

8.24.3.4 virtual m_value_type& InterpolatorIterator::next () [pure virtual]

Returns the next value in the iteration.

Note:
Because this returns a reference type, value_ can be changed by the caller. Steps should be taken to prevent this (with a pass-to-caller member variable perhaps)

Implements AbstractIterator< m_value_type >.

Implemented in LinearInterpolatorIterator, ExponentialInterpolatorIterator, and CubicSplineInterpolatorIterator.

8.24.4 Member Data Documentation

8.24.4.1 m_value_type InterpolatorIterator::delta_ [protected]

The amount the value changed each call to next().

Definition at line 99 of file InterpolatorIterator.h.

Referenced by LinearInterpolatorIterator::next().

8.24.4.2 list<Entry> InterpolatorIterator::queue_ [protected]

This class works by keeping entries in a queue.

Definition at line 83 of file InterpolatorIterator.h.

Referenced by append(), hasNext(), CubicSplineInterpolatorIterator::next(), ExponentialInterpolatorIterator::next(), and LinearInterpolatorIterator::next().

8.24.4.3 m_sample_count_type InterpolatorIterator::stepsLeft_ [protected]

The number of steps left until the next entry needs to be accessed.

Definition at line 89 of file InterpolatorIterator.h.

Referenced by hasNext(), CubicSplineInterpolatorIterator::next(), Exponential-InterpolatorIterator::next(), and LinearInterpolatorIterator::next().

8.24.4.4 m_value_type InterpolatorIterator::value_ [protected]

The current value.

Definition at line 94 of file InterpolatorIterator.h.

Referenced by CubicSplineInterpolatorIterator::next(), ExponentialInterpolator-Iterator::next(), and LinearInterpolatorIterator::next().

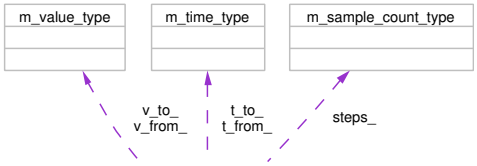
The documentation for this class was generated from the following files:

- InterpolatorIterator.h
- InterpolatorIterator.cpp

8.25 InterpolatorIterator::Entry Class Reference

`#include <InterpolatorIterator.h>`

Collaboration diagram for InterpolatorIterator::Entry:



Public Member Functions

- Entry (m_time_type t_from, m_time_type t_to, m_value_type v_from, m_value_type v_to, m_sample_count_type steps)

Public Attributes

- m_time_type t_from_

  time starts at this

- m_time_type t_to_

  time ends at this

- m_value_type v_from_

  value starts at this

- m_value_type v_to_

  value ends at this

- m_sample_count_type steps_

  number of steps to take inbetween

8.25.1 Detailed Description

An entry for this iterator. This is kept private so that it may change in the future without difficulty.

Definition at line 55 of file InterpolatorIterator.h.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 InterpolatorIterator::Entry::Entry (m_time_type *t_from*, m_time_type *t_to*, m_value_type *v_from*, m_value_type *v_to*, m_sample_count_type *steps*) [inline]

This is a constructor for an interpolator iterator entry.

Parameters:

t_from The start time

t_to The end time

v_from The beginning value

v_to The end value

steps the number of steps to take

Definition at line 66 of file InterpolatorIterator.h.

References m_sample_count_type, m_time_type, m_value_type, steps_, t_from_, t_to_, v_from_, and v_to_.

8.25.3 Member Data Documentation

8.25.3.1 m_sample_count_type InterpolatorIterator::Entry::steps_

number of steps to take inbetween

Definition at line 77 of file InterpolatorIterator.h.

Referenced by Entry().

8.25.3.2 m_time_type InterpolatorIterator::Entry::t_from_

time starts at this

Definition at line 69 of file InterpolatorIterator.h.

Referenced by Entry().

8.25.3.3 m_time_type InterpolatorIterator::Entry::t_to_

time ends at this

Definition at line 71 of file InterpolatorIterator.h.

Referenced by Entry().

8.25.3.4 m_value_type InterpolatorIterator::Entry::v_from_

value starts at this

Definition at line 73 of file InterpolatorIterator.h.

Referenced by Entry().

8.25.3.5 m_value_type InterpolatorIterator::Entry::v_to_

value ends at this

Definition at line 75 of file InterpolatorIterator.h.

Referenced by Entry().

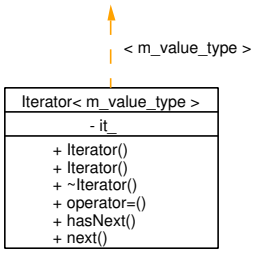
The documentation for this class was generated from the following file:

- InterpolatorIterator.h

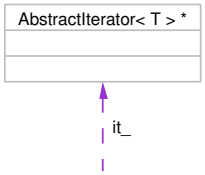
8.26 Iterator< T > Class Template Reference

`#include <Iterator.h>`

Inheritance diagram for Iterator< T >:



Collaboration diagram for Iterator< T >:



Public Member Functions

- Iterator (AbstractIterator< T > *it)
- Iterator (const Iterator &I)
- ~Iterator ()
- Iterator & operator= (const Iterator &I)
- bool hasNext ()
- T & next ()

Private Attributes

- AbstractIterator< T > * it_

8.26.1 Detailed Description

template<class T> class Iterator< T >

This is a wrapper around an AbstractIterator pointer. This gets around C++'s polymorphic limitations. It allows for simple management of this AbstractClass.

Author:
Braden Kowitz

Definition at line 42 of file Iterator.h.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 template<class T> Iterator< T >::Iterator (AbstractIterator< T > * *it*) [inline]

This is a constructor.

Parameters:
it A pointer to an AbstractIterator

Definition at line 53 of file Iterator.h.

8.26.2.2 template<class T> Iterator< T >::Iterator (const Iterator< T > & *I*) [inline]

This is a copy constructor

Parameters:
I The iterator to make a copy of

Definition at line 61 of file Iterator.h.

8.26.2.3 template<class T> Iterator< T >::~Iterator () [inline]

This is the destructor which deletes the underlying object.

Definition at line 69 of file Iterator.h.

8.26.3 Member Function Documentation

8.26.3.1 template<class T> bool Iterator< T >::hasNext () [inline]

This checks whether there is another element in the iterator.

Return values:

true If there is another element

false If there is not another element

Definition at line 96 of file Iterator.h.

Referenced by MultiTrack::composite(), Filter::do_filter_MultiTrack(), Reverb::do_reverb_MultiTrack(), Interpolator::getMaxValue(), Sound::getTotalDuration(), MultiTrack::MultiTrack(), MultiTrack::operator=(), Sound::render(), Score::render(), Interpolator::scale(), Sound::setPartialParam(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), AuWriter::write(), AuWriter::write_one_per_track(), Sound::xml_print(), Score::xml_print(), Interpolator::xml_print(), Envelope::xml_print(), DynamicVariableSequence::xml_print(), and MultiTrack::~MultiTrack().

8.26.3.2 template<class T> T& Iterator< T >::next () [inline]

This returns the next iterator.

Returns:

The next iterator.

Definition at line 105 of file Iterator.h.

Referenced by MultiTrack::composite(), Filter::do_filter_MultiTrack(), Reverb::do_reverb_MultiTrack(), Interpolator::getMaxValue(), Sound::getTotalDuration(), Envelope::getValue(), DynamicVariableSequence::getValue(), MultiTrack::MultiTrack(), MultiTrack::operator=(), Sound::render(), Score::render(), Interpolator::scale(), Sound::setPartialParam(), Pan::spatialize(), MultiPan::spatialize(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), AuWriter::write(), AuWriter::write_one_per_track(), Sound::xml_print(), Score::xml_print(), Interpolator::xml_print(), Envelope::xml_print(), DynamicVariableSequence::xml_print(), and MultiTrack::~MultiTrack().

8.26.3.3 template<class T> Iterator& Iterator< T >::operator= (const Iterator< T > & *I*) [inline]

This is an overloaded assignment operator.

Parameters:

I An iterator to assign

Definition at line 78 of file Iterator.h.

8.26.4 Member Data Documentation

8.26.4.1 template<class T> AbstractIterator<T>* Iterator< T >::it_ [private]

Definition at line 45 of file Iterator.h.

Referenced by Iterator< m_value_type >::hasNext(), Iterator< m_value_type >::Iterator(), Iterator< m_value_type >::next(), Iterator< m_value_type >::operator=(), and Iterator< m_value_type >::~Iterator().

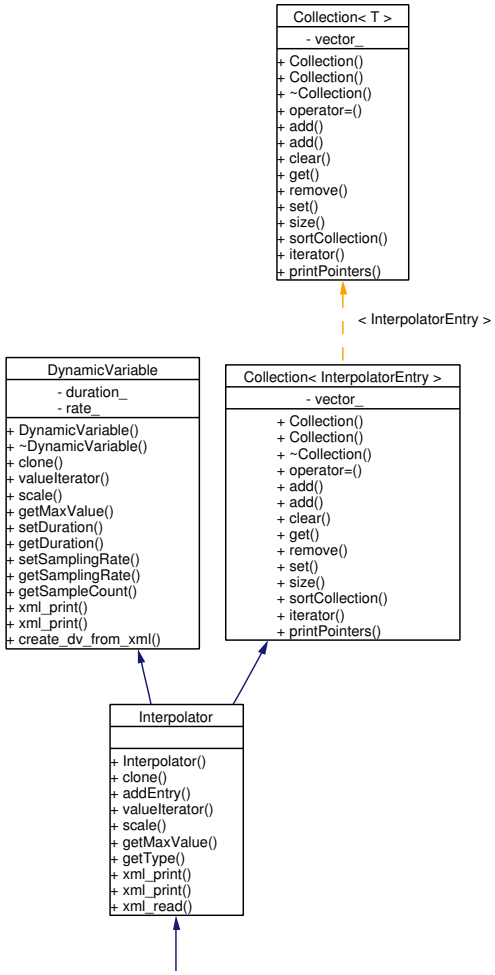
The documentation for this class was generated from the following file:

- Iterator.h

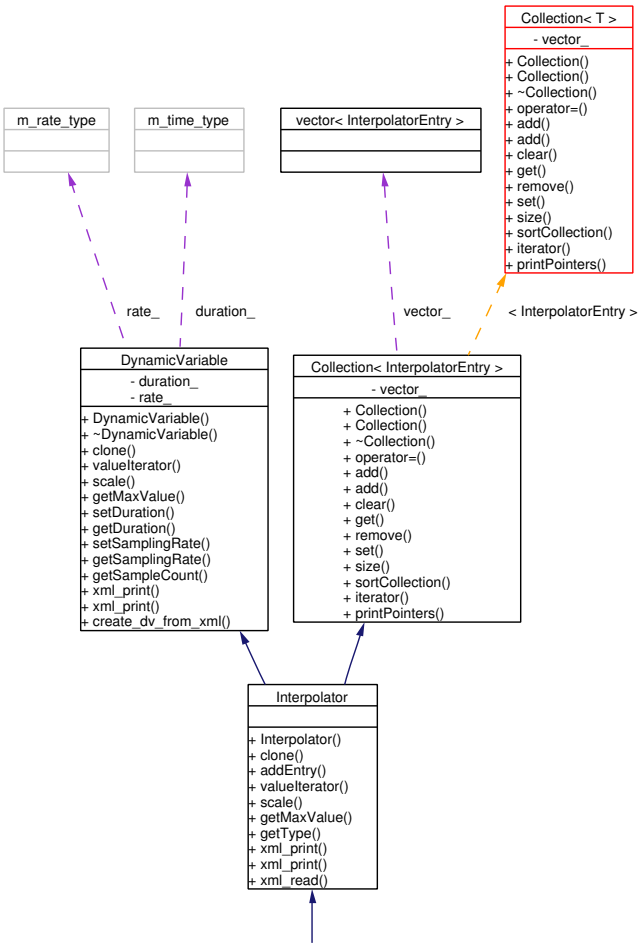
8.27 LinearInterpolator Class Reference

`#include <InterpolatorTypes.h>`

Inheritance diagram for LinearInterpolator:



Collaboration diagram for LinearInterpolator:



Public Member Functions

- LinearInterpolator ()
- LinearInterpolator * clone ()
- Iterator< m_value_type > valueIterator ()
- virtual interpolation_type getType ()

8.27.1 Detailed Description

This is a DynamicVariable that changes over time. This linearly interpolates between a set of points ordered in time. (LinearInterpolatorEntry)

Author:
Braden Kowitz
Philipp Fraund

Definition at line 45 of file InterpolatorTypes.h.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 LinearInterpolator::LinearInterpolator ()

This is the default constructor.

Definition at line 39 of file InterpolatorTypes.cpp.

Referenced by clone().

8.27.3 Member Function Documentation

8.27.3.1 LinearInterpolator * LinearInterpolator::clone () [virtual]

This makes a clone of a LinearInterpolator.

Returns:
A new LinearInterpolator

Implements Interpolator.

Definition at line 44 of file InterpolatorTypes.cpp.

References LinearInterpolator().

8.27.3.2 interpolation_type LinearInterpolator::getType () [virtual]

This provides an implementation to get the type of interpolator.

Returns:
The interpolation type

Implements Interpolator.

Definition at line 111 of file InterpolatorTypes.cpp.

References interpolation_type, and LINEAR.

8.27.3.3 Iterator< m_value_type > LinearInterpolator::valueIterator () [virtual]

This creates an iterator over LinearInterpolators.

Returns:
An iterator

Implements Interpolator.

Definition at line 53 of file InterpolatorTypes.cpp.

References InterpolatorIterator::append(), Collection< InterpolatorEntry >::get(), DynamicVariable::getDuration(), DynamicVariable::getSamplingRate(), Iterator< T >::hasNext(), Collection< InterpolatorEntry >::iterator(), m_sample_count_type, m_time_type, m_value_type, Iterator< T >::next(), Collection< InterpolatorEntry >::size(), Collection< InterpolatorEntry >::sortCollection(), InterpolatorEntry::time_, and InterpolatorEntry::value_.

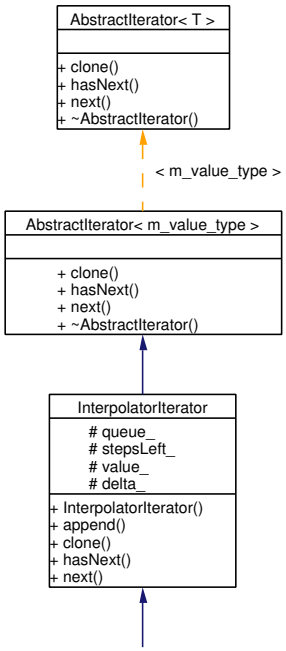
The documentation for this class was generated from the following files:

- InterpolatorTypes.h
- InterpolatorTypes.cpp

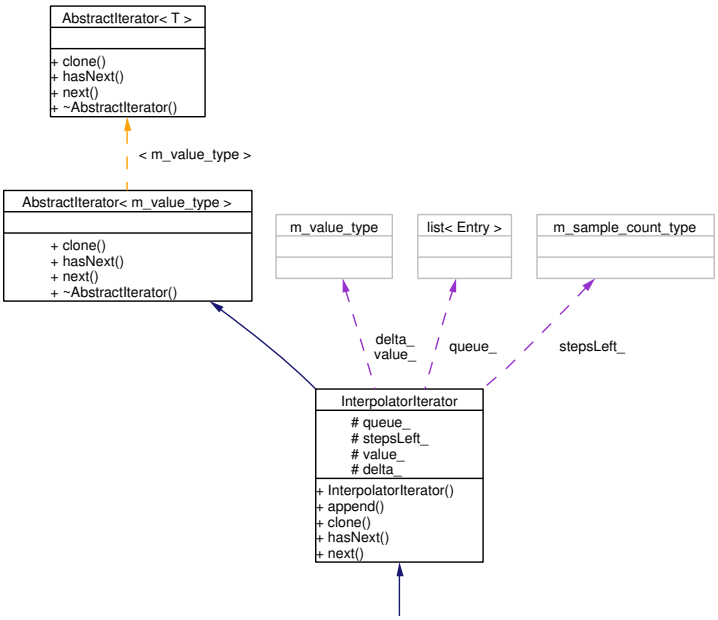
8.28 LinearInterpolatorIterator Class Reference

`#include <InterpolatorIterator.h>`

Inheritance diagram for LinearInterpolatorIterator:



Collaboration diagram for LinearInterpolatorIterator:



Public Member Functions

- LinearInterpolatorIterator ()

  constructor for iterator

- LinearInterpolatorIterator * clone ()

  clone for iterator

- m_value_type & next ()

  get next value function

8.28.1 Detailed Description

This is an iterator that will iterate over values in a LinearInterpolator.

Definition at line 152 of file InterpolatorIterator.h.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 LinearInterpolatorIterator::LinearInterpolatorIterator ()

constructor for iterator

Definition at line 69 of file InterpolatorIterator.cpp.

Referenced by clone().

8.28.3 Member Function Documentation

8.28.3.1 LinearInterpolatorIterator * LinearInterpolatorIterator::clone () [virtual]

clone for iterator

Implements InterpolatorIterator.

Definition at line 73 of file InterpolatorIterator.cpp.

References LinearInterpolatorIterator().

8.28.3.2 m_value_type & LinearInterpolatorIterator::next () [virtual]

get next value function

Implements InterpolatorIterator.

Definition at line 78 of file InterpolatorIterator.cpp.

References InterpolatorIterator::delta_, m_value_type, InterpolatorIterator::queue_, InterpolatorIterator::stepsLeft_, and InterpolatorIterator::value_.

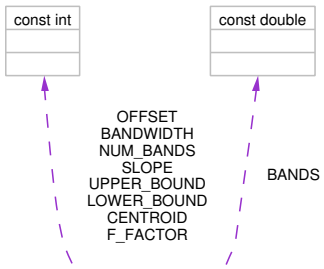
The documentation for this class was generated from the following files:

- InterpolatorIterator.h
- InterpolatorIterator.cpp

8.29 Loudness Class Reference

`#include <Loudness.h>`

Collaboration diagram for Loudness:



Static Public Member Functions

- void calculate (Sound &snd, m_rate_type rate=DEFAULT_LOUDNESS_-RATE)

Static Private Member Functions

- int criticalBandIndex (m_value_type frequency)

Static Private Attributes

- const int LOWER_BOUND = 0
- const int UPPER_BOUND = 1
- const int CENTROID = 2
- const int BANDWIDTH = 3
- const int F_FACTOR = 4
- const int OFFSET = 5
- const int SLOPE = 6
- const double BANDS [24][7]
- const int NUM_BANDS = 24

8.29.1 Detailed Description

This performs loudness calculations on sound objects:

STEPS:

1. Separate all partials into groups of critical bands.
    - This division is done by the partial's frequency
2. Find the loudest partial by virtue of it's WaveShape variable.
    - And thus, find the highest WaveShape value.
3. Calculate the Band Gamma for each critical band. [eq 2.34]
    - The max WaveShape from step 2 becomes the maxWaveShape here.
    - bandGamma = 0
    - for each partial
        - bandGamma += pow((thisWaveShape / maxWaveShape), (1.0 / log10(2.0) ∗ this_band_b_value));
    - bandGamma = pow(bandGamma, (log10(2.0) ∗ this_band_b_value));
4. Find the maximum band-gamma: maxGamma
5. Calculate the numerator needed for [eq 2.37]
    - gammaTotal = 0;
    - for each band
        - gammaTotal += bandGamma ∗ this_band_f_value;
    - numerator = desiredSonesValue / (maxGamma + gammaTotal);
6. Calculate scaling factor for each partial in each of the bands.
    - (uses maxWaveShape & numerator)
    - [eq 2.37] Ls = (thisWaveShape / maxWaveShape) ∗ numerator;
    - [eq 2.4] Lp = log(ls) / log(2.0) ∗ 10.0 + 40.0;
    - [eq 2.13] L = (-a/b) + (1/b) ∗ Lp
    - [eq 2.5] A = pow(10.0, (-1.0 ∗ ((120.0 - L) / 20.0)));
    - ScalingFactor = A / thisWaveShape Phew - done!

    Author:
    Braden Kowitz

Definition at line 79 of file Loudness.h.

8.29.2 Member Function Documentation

8.29.2.1 void Loudness::calculate (Sound & *snd*, m_rate_type *rate* = DEFAULT_LOUDNESS_RATE) [static]

This performs the Loudness calculation on a sound object. Basically, it looks at each partial's WAVE_SHPE, FREQUENCY, and RELATIVE_AMPLITUDE parameters. It looks at the sounds LOUDNESS parameter, and sets the LOUDNES_SCALER DynamicVariable for each partial.

Definition at line 45 of file Loudness.cpp.

References BANDS, criticalBandIndex(), DURATION, F_FACTOR, FREQUENCY, Collection< Partial >::get(), DynamicVariable::getMaxValue(), ParameterLib< PartialStaticParam, PartialDynamicParam >::getParam(), ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), LOUDNESS, LOUDNESS_SCALAR, m_rate_type, m_sample_count_type, m_time_type, m_value_type, NUM_BANDS, RELATIVE_AMPLITUDE, DynamicVariable::setDuration(), ParameterLib< PartialStaticParam, PartialDynamicParam >::setParam(), DynamicVariable::setSamplingRate(), Collection< Partial >::size(), DynamicVariable::valueIterator(), and WAVE_SHAPE.

Referenced by Sound::render().

8.29.2.2 int Loudness::criticalBandIndex (m_value_type *frequency*) [static, private]

This function returns the critical band index that matches the specified frequency.

Parameters:

frequency The freqency of the band to return

Returns:

The index of a CriticalBand

Definition at line 161 of file Loudness.cpp.

References BANDS, LOWER_BOUND, m_value_type, NUM_BANDS, and UPPER_BOUND.

Referenced by calculate().

8.29.3 Member Data Documentation

8.29.3.1 const double Loudness::BANDS [static, private]

Initial value:

```
{
  {2.000000E+01,1.000000E+02,5.000000E+01,8.000000E+01,7.500099E-02,3.434098E+01,7.426288E-01
  {1.000000E+02,2.000000E+02,1.500000E+02,1.000000E+02,3.000000E-01,1.102623E+01,8.863131E-01
  {2.000000E+02,3.000000E+02,2.500000E+02,1.000000E+02,1.800000E-01,3.658798E+00,9.307306E-01
  {3.000000E+02,4.000000E+02,3.500000E+02,1.000000E+02,1.300000E-01,6.079359E-01,9.552449E-01
  {4.000000E+02,5.100000E+02,4.500000E+02,1.100000E+02,1.113188E-01,-1.050592E+00,9.798435E-0
  {5.100000E+02,6.300000E+02,5.700000E+02,1.200000E+02,9.825261E-02,-1.768491E+00,9.962857E-0
  {6.300000E+02,7.700000E+02,7.000000E+02,1.400000E+02,9.388891E-02,-1.488782E+00,9.993805E-0
  {7.700000E+02,9.200000E+02,8.400000E+02,1.500000E+02,8.463656E-02,-7.874900E-01,1.001088E+0
  {9.200000E+02,1.080000E+03,1.000000E+03,1.600000E+02,7.749530E-02,-2.397041E-02,9.996908E-0
  {1.080000E+03,1.270000E+03,1.170000E+03,1.900000E+02,7.818675E-02,3.330517E-01,9.926001E-01
  {1.270000E+03,1.480000E+03,1.370000E+03,2.100000E+02,7.454711E-02,2.924882E-01,9.866766E-01
  {1.480000E+03,1.720000E+03,1.600000E+03,2.400000E+02,7.344415E-02,-9.319628E-02,9.820759E-0
  {1.720000E+03,2.000000E+03,1.850000E+03,2.800000E+02,7.366146E-02,-7.555149E-01,9.725524E-0
  {2.000000E+03,2.320000E+03,2.150000E+03,3.200000E+02,7.269603E-02,-1.368559E+00,9.559184E-0
  {2.320000E+03,2.700000E+03,2.500000E+03,3.800000E+02,7.400790E-02,-2.310496E+00,9.405730E-0
  {2.700000E+03,3.150000E+03,2.900000E+03,4.500000E+02,7.500003E-02,-4.106658E+00,9.374723E-0
  {3.150000E+03,3.700000E+03,3.400000E+03,5.500000E+02,7.773245E-02,-5.443059E+00,9.383848E-0
  {3.700000E+03,4.400000E+03,4.000000E+03,7.000000E+02,8.272464E-02,-5.612481E+00,9.440814E-0
  {4.400000E+03,5.300000E+03,4.800000E+03,9.000000E+02,8.793907E-02,-4.374397E+00,9.618959E-0
  {5.300000E+03,6.400000E+03,5.800000E+03,1.100000E+03,8.895339E-02,1.201688E-01,9.705871E-01
   {6.400000E+03,7.700000E+03,7.000000E+03,1.300000E+03,8.745852E-02,6.446360E+00,9.652756E-0
  {7.700000E+03,9.500000E+03,8.500000E+03,1.800000E+03,9.774416E-02,1.050712E+01,9.499248E-01
  {9.500000E+03,1.200000E+04,1.050000E+04,2.500000E+03,1.074515E-01,8.187814E+00,9.339424E-01
  {1.200000E+04,1.550000E+04,1.350000E+04,3.500000E+03,1.167188E-01,-1.822547E+00,9.533570E-0
}
```

This holds the CriticalBand data.

Definition at line 262 of file Loudness.cpp.

Referenced by calculate(), and criticalBandIndex().

8.29.3.2 const int Loudness::BANDWIDTH = 3 [static, private]

Definition at line 166 of file Loudness.h.

8.29.3.3 const int Loudness::CENTROID = 2 [static, private]

Definition at line 165 of file Loudness.h.

8.29.3.4 const int Loudness::F_FACTOR = 4 [static, private]

Definition at line 167 of file Loudness.h.

Referenced by calculate().

8.29.3.5 const int Loudness::LOWER_BOUND = 0 [static, private]

These are the indices of the Critical Band Field.

Definition at line 163 of file Loudness.h.

Referenced by criticalBandIndex().

8.29.3.6 const int Loudness::NUM_BANDS = 24 [static, private]

There are 24 critical bands ∗

Definition at line 175 of file Loudness.h.

Referenced by calculate(), and criticalBandIndex().

8.29.3.7 const int Loudness::OFFSET = 5 [static, private]

Definition at line 168 of file Loudness.h.

8.29.3.8 const int Loudness::SLOPE = 6 [static, private]

Definition at line 169 of file Loudness.h.

8.29.3.9 const int Loudness::UPPER_BOUND = 1 [static, private]

Definition at line 164 of file Loudness.h.

Referenced by criticalBandIndex().

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

- Loudness.h
- Loudness.cpp

8.30 Loudness::CriticalBand Class Reference

Collaboration diagram for Loudness::CriticalBand:

Public Member Functions

- CriticalBand (int ID)
- m_value_type getBandGamma (m_value_type maxAmp)

Public Attributes

- int ID_
- vector< PartialSnapshot > partials_

8.30.1 Detailed Description

This is a class to hold a critical band.

Author:
Braden Kowitz

Definition at line 130 of file Loudness.h.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 Loudness::CriticalBand::CriticalBand (int *ID*)

This is a constructor.

Parameters:
ID Which critical band

Definition at line 194 of file Loudness.cpp.

8.30.3 Member Function Documentation

8.30.3.1 m_value_type Loudness::CriticalBand::getBandGamma (m_value_type *maxAmp*)

This function gets the gamma for the band

Parameters:
maxAmp A cap on the amplitude

Returns:
The band gamma

Definition at line 199 of file Loudness.cpp.

References ID_, m_value_type, and partials_.

8.30.4 Member Data Documentation

8.30.4.1 int Loudness::CriticalBand::ID_

An identifier for the critical band ∗

Definition at line 134 of file Loudness.h.

Referenced by getBandGamma().

8.30.4.2 vector<PartialSnapshot> Loudness::CriticalBand::partials_

A vector to hold partial snapshots ∗

Definition at line 136 of file Loudness.h.

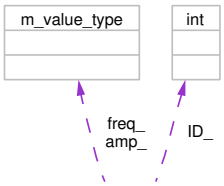
Referenced by getBandGamma().

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

- Loudness.h
- Loudness.cpp

8.31 Loudness::PartialSnapshot Class Reference

Collaboration diagram for Loudness::PartialSnapshot:



Public Member Functions

- PartialSnapshot (int ID, m_value_type freq, m_value_type amp)
- m_value_type getScalingFactor (int bandID, m_value_type maxAmp, m_value_type numerator)

Public Attributes

- int ID_
- m_value_type freq_
- m_value_type amp_

8.31.1 Detailed Description

This is the class that contains a snapshot of part of a partial.

Author:
Braden Kowitz

Definition at line 97 of file Loudness.h.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 Loudness::PartialSnapshot::PartialSnapshot (int *ID*, m_value_type *freq*, m_value_type *amp*)

This is a constructor for a PartialSnapshot

Parameters:

ID Which part of the partial

freq The frequency of the part of the partial

amp The amplitude of the part of the partial

Definition at line 223 of file Loudness.cpp.

References m_value_type.

8.31.3 Member Function Documentation

8.31.3.1 m_value_type Loudness::PartialSnapshot::getScalingFactor (int *bandID*, m_value_type *maxAmp*, m_value_type *numerator*)

This returns the scalingfactor of the band.

Parameters:

bandID Which band

maxAmp The cap on the amplitude

numerator

Returns:

The scaling factor

Definition at line 227 of file Loudness.cpp.

References amp_, and m_value_type.

8.31.4 Member Data Documentation

8.31.4.1 m_value_type Loudness::PartialSnapshot::amp_

The amplitude ∗

Definition at line 107 of file Loudness.h.

Referenced by getScalingFactor().

8.31.4.2 m_value_type Loudness::PartialSnapshot::freq_

The frequency ∗

Definition at line 104 of file Loudness.h.

8.31.4.3 int Loudness::PartialSnapshot::ID_

An identifier for the part of the partial ∗

Definition at line 101 of file Loudness.h.

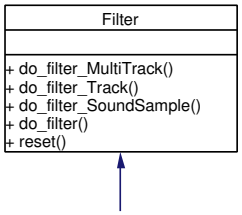
The documentation for this class was generated from the following files:

- Loudness.h
- Loudness.cpp

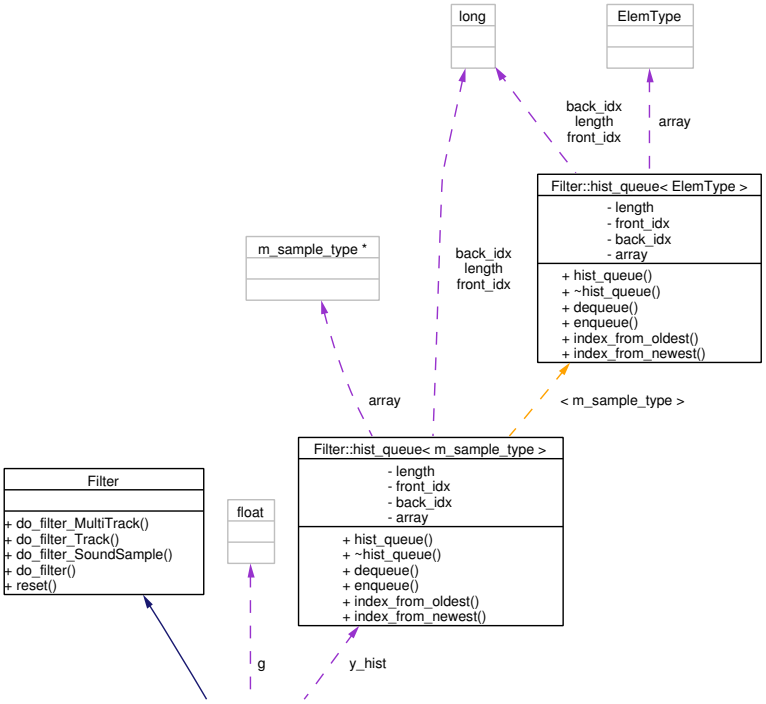
8.32 LowPassFilter Class Reference

`#include <LowPassFilter.h>`

Inheritance diagram for LowPassFilter:



Collaboration diagram for LowPassFilter:



Public Member Functions

- LowPassFilter (float gain)
- ~LowPassFilter ()
- m_sample_type do_filter (m_sample_type x_t)
- void reset (void)
- void xml_print ()

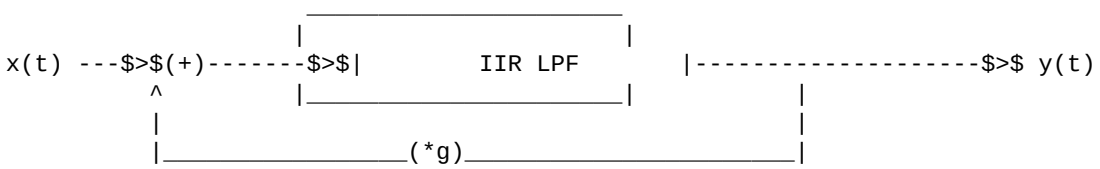
Private Attributes

- float g
- Filter::hist_queue< m_sample_type > * y_hist

8.32.1 Detailed Description

This is a simple 1st order IIR low-pass filter:

$y(t) = x(t) + g*y(t-1)$



Note:
This filter (and all other filters) are stateful machines with internal feedback mechanisms. Thus this filter should be allocated anew each time you begin a new channel and should not be mixed between channels.

Author:
Jim LIndstrom

Definition at line 60 of file LowPassFilter.h.

8.32.2 Constructor & Destructor Documentation

8.32.2.1 LowPassFilter::LowPassFilter (float *gain*)

This is a constructor.

Parameters:
gain The low-pass feedback gain

Definition at line 41 of file LowPassFilter.cpp.

References Filter::hist_queue< m_sample_type >::enqueue(), g, and y_hist.

8.32.2.2 LowPassFilter::~LowPassFilter ()

This is a destructor.

Definition at line 52 of file LowPassFilter.cpp.

References y_hist.

8.32.3 Member Function Documentation

8.32.3.1 m_sample_type LowPassFilter::do_filter (m_sample_type *x_t*) [virtual]

This method applies a low-pass filter to a single sample

Parameters:
x_t The input sample

Returns:
The filtered sample

Implements Filter.

Definition at line 58 of file LowPassFilter.cpp.

References Filter::hist_queue< m_sample_type >::dequeue(), Filter::hist_queue< m_sample_type >::enqueue(), g, m_sample_type, and y_hist.

Referenced by LPCombFilter::do_filter().

8.32.3.2 void LowPassFilter::reset (void) [virtual]

This method should be redefined by each class derived from Filter to reset the filter to an initial state. It should have the same effect as deleting the filter and creating a new one.

Implements Filter.

Definition at line 70 of file LowPassFilter.cpp.

References Filter::hist_queue< m_sample_type >::enqueue(), and y_hist.

8.32.3.3 void LowPassFilter::xml_print ()

Deprecated
This outputs an XML representation of the object to STDOUT

Definition at line 79 of file LowPassFilter.cpp.

References g.

8.32.4 Member Data Documentation

8.32.4.1 float LowPassFilter::g [private]

The gain for the filter

Definition at line 100 of file LowPassFilter.h.

Referenced by do_filter(), LowPassFilter(), and xml_print().

8.32.4.2 Filter::hist_queue<m_sample_type>* LowPassFilter::y_hist [private]

This queue holds past samples to implement the delay

Definition at line 105 of file LowPassFilter.h.

Referenced by do_filter(), LowPassFilter(), reset(), and ~LowPassFilter().

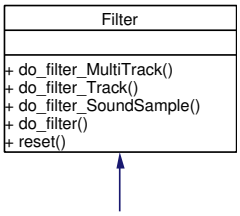
The documentation for this class was generated from the following files:

- LowPassFilter.h
- LowPassFilter.cpp

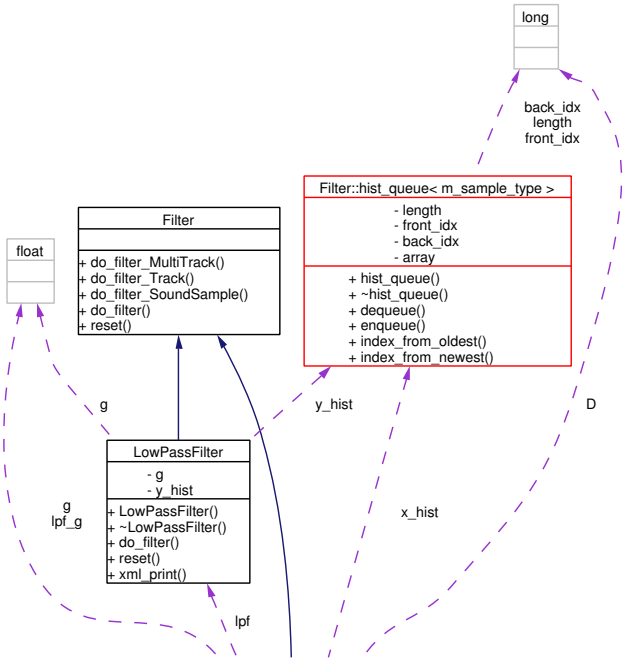
8.33 LPCombFilter Class Reference

`#include <LPCombFilter.h>`

Inheritance diagram for LPCombFilter:



Collaboration diagram for LPCombFilter:



Public Member Functions

- LPCombFilter (float gain, long delay, float lpf_gain)
- ~LPCombFilter ()
- m_sample_type do_filter (m_sample_type x_t)
- void reset (void)
- void xml_print (ofstream &xmlOutput)
- LPCombFilter ()
- void set_g (float new_g)
- void set_D (long D)
- void set_lpf_g (float new_lpf_g)
- void xml_read (XmlReader::xmltag *lptag)

Private Attributes

- float g
- long D
- float lpf_g
- LowPassFilter * lpf
- Filter::hist_queue< m_sample_type > * x_hist

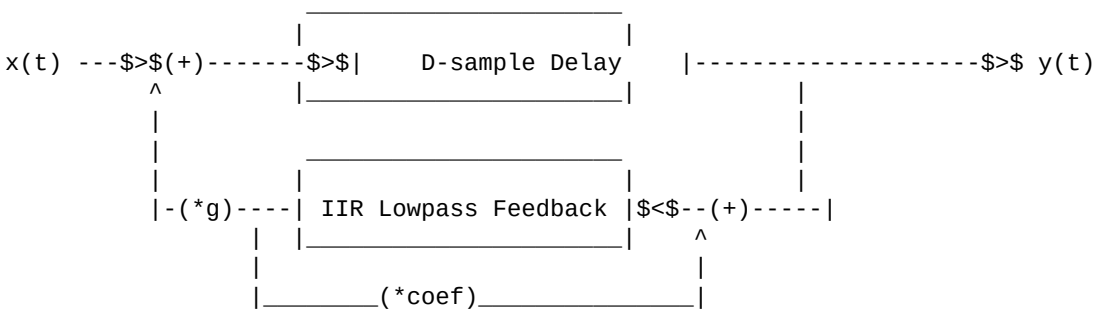
8.33.1 Detailed Description

This is a comb filter with a lowpass feedback element. The comb filter class implements a comb filter with a lowpass feedback loop as described on page 385 of 'Elements of Computer Music':

y(t) = x[t-D] + g * lowpass(x(t-D))

where lowpass(s[t]) = s[t] + coef*lowpass(s[t-1])

Note that for stability, we must make sure that: -1 < g/(1 - coef) < +1



Note:
: This filter (and all other 'class Filter's) are stateful machines with internal feedback mechanisms. Thus this filter should be reset() each time you begin a new channel and should not be mixed between channels.

Author:
Jim Lindstrom

Definition at line 74 of file LPCombFilter.h.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 LPCombFilter::LPCombFilter (float *gain*, long *delay*, float *lpf_gain*)

This is a constructor.

Parameters:

gain The feedback gain (0.0 to 1.0) applied to the IIR lowpass feedback unit

delay The comb delay, in units of samples.

lpf_gain The internal gain of the lowpass feedback unit

Definition at line 43 of file LPCombFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), g, lpf, lpf_g, and x_hist.

8.33.2.2 LPCombFilter::~LPCombFilter ()

This is the destructor.

Definition at line 59 of file LPCombFilter.cpp.

References lpf, and x_hist.

8.33.2.3 LPCombFilter::LPCombFilter ()

This constructor is used when recreating the object from an XML file when you don't know all the parameters until you've read them in. Don't use this constructor unless you intend to use the proper calls to set the gain, delay, and lpf.

Definition at line 108 of file LPCombFilter.cpp.

8.33.3 Member Function Documentation

8.33.3.1 m_sample_type LPCombFilter::do_filter (m_sample_type *x_t*) [virtual]

This method applies a lpcomb filter to a single sample

Parameters:

x_t The input sample

Returns:

The filtered sample

Implements Filter.

Definition at line 66 of file LPCombFilter.cpp.

References Filter::hist_queue< m_sample_type >::dequeue(), LowPassFilter::do_filter(), Filter::hist_queue< m_sample_type >::enqueue(), g, lpf, m_sample_type, and x_hist.

Referenced by Reverb::do_reverb().

8.33.3.2 void LPCombFilter::reset (void) [virtual]

This method should be redefined by each class derived from Filter to reset the filter to an initial state. It should have the same effect as deleting the filter and creating a new one.

Implements Filter.

Definition at line 77 of file LPCombFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), lpf, lpf_g, and x_hist.

Referenced by Reverb::reset().

8.33.3.3 void LPCombFilter::set_D (long *D*)

This sets the delay.

Parameters:
D The delay

Definition at line 115 of file LPCombFilter.cpp.

References D, Filter::hist_queue< m_sample_type >::enqueue(), and x_hist.

Referenced by xml_read().

8.33.3.4 void LPCombFilter::set_g (float *new_g*)

This sets the gain.

Parameters:
new_g The gain

Definition at line 111 of file LPCombFilter.cpp.

References g.

Referenced by xml_read().

8.33.3.5 void LPCombFilter::set_lpf_g (float *new_lpf_g*)

This sets the low-pass feedback.

Parameters:
new_lpf_g

Definition at line 124 of file LPCombFilter.cpp.

References lpf, and lpf_g.

Referenced by xml_read().

8.33.3.6 void LPCombFilter::xml_print (ofstream & *xmlOutput*)

Deprecated
This outputs an XML representation of the object to STDOUT

Definition at line 93 of file LPCombFilter.cpp.

References D, g, and lpf_g.

Referenced by Reverb::xml_print().

8.33.3.7 void LPCombFilter::xml_read (XmlReader::xmltag * *lptag*)

Deprecated

Definition at line 132 of file LPCombFilter.cpp.

References XmlReader::xmltag::findChildParamValue(), set_D(), set_g(), and set_lpf_g().

Referenced by Reverb::xml_read().

8.33.4 Member Data Documentation

8.33.4.1 long LPCombFilter::D [private]

The delay for the comb component of the filter

Definition at line 152 of file LPCombFilter.h.

Referenced by LPCombFilter(), reset(), set_D(), and xml_print().

8.33.4.2 float LPCombFilter::g [private]

The gain for the comb component of the filter

Definition at line 147 of file LPCombFilter.h.

Referenced by do_filter(), LPCombFilter(), set_g(), and xml_print().

8.33.4.3 LowPassFilter* LPCombFilter::lpf [private]

This implements the lowpass-feedback portion of the filter

Definition at line 162 of file LPCombFilter.h.

Referenced by do_filter(), LPCombFilter(), reset(), set_lpf_g(), and ~LPCombFilter().

8.33.4.4 float LPCombFilter::lpf_g [private]

The gain for the lowpass-feedback component of the filter

Definition at line 157 of file LPCombFilter.h.

Referenced by LPCombFilter(), reset(), set_lpf_g(), and xml_print().

8.33.4.5 Filter::hist_queue<m_sample_type>* LPCombFilter::x_hist [private]

This queue holds past samples to implement the delay

Definition at line 167 of file LPCombFilter.h.

Referenced by do_filter(), LPCombFilter(), reset(), set_D(), and ~LPCombFilter().

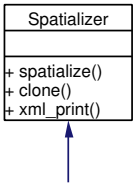
The documentation for this class was generated from the following files:

- LPCombFilter.h
- LPCombFilter.cpp

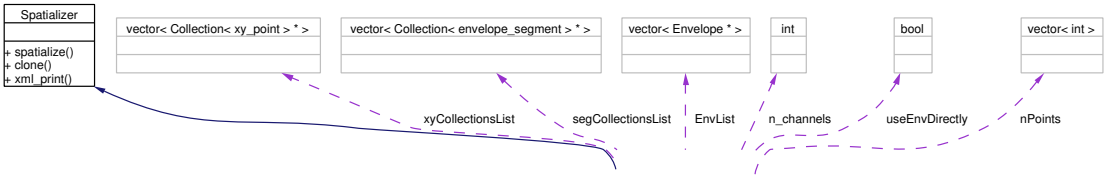
8.34 MultiPan Class Reference

`#include <MultiPan.h>`

Inheritance diagram for MultiPan:



Collaboration diagram for MultiPan:



Public Member Functions

- MultiPan (int nChans)

- MultiPan (int nChans, vector< Envelope * > &List)
- ~MultiPan ()
- MultiPan * clone ()
- void addEntry (float t,...)
- void addEntryLocation (float t, float theta, float radius)
- MultiTrack * spatialize (Track &t, int numTracks)
- void xml_print (ofstream &xmlOutput)

Private Member Functions

- void addEntryHelperFn (int envIdx, float t, float amp)

Private Attributes

- bool useEnvDirectly
- vector< Envelope * > EnvList
- vector< Collection< xy_point > * > xyCollectionsList
- vector< Collection< envelope_segment > * > segCollectionsList
- vector< int > nPoints
- int n_channels

8.34.1 Detailed Description

MultiPan is a Spatializer. It is to be used instead of Pan when you need to spatialize a sound over more than 2 speakers. To spatialize the sound over time you call addEntry or addEntryLocation one or more times and give a timestamp for each call.

Author:
Jim Lindstrom

Definition at line 48 of file MultiPan.h.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 MultiPan::MultiPan (int *nChans*)

This is a constructor. To use this constructor, you must call addEntry or addEntryLocation one or more times to set the spatialization data. Use the other constructor if you want to instead pass in a list of Interpolators (one per channel).

Parameters:
nChans The number of channels (speakers) to pan across

Definition at line 49 of file MultiPan.cpp.

References EnvList, n_channels, segCollectionsList, useEnvDirectly, and xyCollectionsList.

Referenced by clone().

8.34.2.2 MultiPan::MultiPan (int *nChans*, vector< Envelope * > & *List*)

This is a constructor. To use this constructor, pass in a list of envs (one per channel). If you want to call addEntry or addEntryLocation to set the spatialization data, use the other constructor.

Parameters:

nChans The number of channels (speakers) to pan across

List A vector of envs (nChan number of envs, to be precise). Each env will control the percent that the corresponding speaker will respond to input sound.

Definition at line 77 of file MultiPan.cpp.

References clone(), EnvList, n_channels, and useEnvDirectly.

8.34.2.3 MultiPan::~MultiPan ()

This is the destructor.

Definition at line 90 of file MultiPan.cpp.

References EnvList, segCollectionsList, and xyCollectionsList.

8.34.3 Member Function Documentation

8.34.3.1 void MultiPan::addEntry (float *t*, ...)

Add another spatialization point to the dynamic variables (using the parameters as ratios of speaker volumes). Pass in the time value, t, and a list of numbers (0.0 to 1.0 for each) that gives each speaker's response (as a percentage) to a given sound.

Parameters:

t The time stamp (as a percent of total sound length, from 0.0 to 1.0)

Definition at line 145 of file MultiPan.cpp.

References addEntryHelperFn(), n_channels, and useEnvDirectly.

8.34.3.2 void MultiPan::addEntryHelperFn (int *envIdx*, float *t*, float *amp*) [private]

This is a helper function used by addEntry and addEntryLocation to add a point to an env.

Definition at line 321 of file MultiPan.cpp.

References EnvList, FIXED, envelope_segment::interType, envelope_segment::lengthType, LINEAR, segCollectionsList, envelope_segment::timeValue, xy_point::x, xyCollectionsList, and xy_point::y.

Referenced by addEntry(), and addEntryLocation().

8.34.3.3 void MultiPan::addEntryLocation (float *t*, float *theta*, float *radius*)

Add another spatialization point to the dynamic variables using the parameters as a location within a circular array of speakers. The value theta=0 means straight ahead, positive thetas to the left, negatives to the right. The radius ranges from 0 to 1.

Parameters:

t The time stamp (as a percent of total sound length, from 0.0 to 1.0)

theta The rotation angle (in radians) where the sound will appear to originate from. An angle of zero is directly in front of you. Positive angles rotate to your left, with positive pi (3.14...) being directly behind you. Negative angles rotate to your left and negative pi (-3.14) is another name for the angle directly behind you. For reference, pi/2 is straight left and -pi/2 is straight right. To use pi, math.h defines the constant for you as 'M_PI'. For instance, you could use 'M_PI/2' to refer to the angle at your left.

radius The distance from which the sound will appear to originate from. These should range from 0.0 to 1.0.

Definition at line 190 of file MultiPan.cpp.

References addEntryHelperFn(), n_channels, and useEnvDirectly.

8.34.3.4 MultiPan * MultiPan::clone () [virtual]

This returns an exact duplicate of this MultiPan object.

Returns:

An exact copy of this MultiPan object, with its own env

Reimplemented from Spatializer.

Definition at line 127 of file MultiPan.cpp.

References EnvList, MultiPan(), and n_channels.

Referenced by MultiPan().

8.34.3.5 MultiTrack * MultiPan::spatialize (Track & *t*, int *numTracks*) [virtual]

This spatializes a track and returns a new MultiTrack object with numTracks. The track will be panned accross the channels by the passed in DynamicVariable.

Parameters:

t A track to spatialize

numTracks The number of tracks to spatialize to. (This is redundant, but is kept for similarity of interface with the regular pan object. MultiPan already knows because you have to pass in nChans in the constructor)

Returns:

the track, spatialized to 'numTracks' number of tracks

Reimplemented from Spatializer.

Definition at line 273 of file MultiPan.cpp.

References EnvList, Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), SoundSample::getSamplingRate(), Track::getWave(), m_rate_type, m_sample_count_type, m_value_type, n_channels, and Iterator< T >::next().

8.34.3.6 void MultiPan::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

Reimplemented from Spatializer.

Definition at line 344 of file MultiPan.cpp.

8.34.4 Member Data Documentation

8.34.4.1 vector<Envelope*> MultiPan::EnvList [private]

Definition at line 152 of file MultiPan.h.

Referenced by addEntryHelperFn(), clone(), MultiPan(), spatialize(), and ~MultiPan().

8.34.4.2 int MultiPan::n_channels [private]

Definition at line 156 of file MultiPan.h.

Referenced by addEntry(), addEntryLocation(), clone(), MultiPan(), and spatialize().

8.34.4.3 vector<int> MultiPan::nPoints [private]

Definition at line 155 of file MultiPan.h.

8.34.4.4 vector<Collection<envelope_segment>*> MultiPan::segCollectionsList [private]

Definition at line 154 of file MultiPan.h.

Referenced by addEntryHelperFn(), MultiPan(), and ~MultiPan().

8.34.4.5 bool MultiPan::useEnvDirectly [private]

Definition at line 151 of file MultiPan.h.

Referenced by addEntry(), addEntryLocation(), and MultiPan().

8.34.4.6 vector<Collection<xy_point>*> MultiPan::xyCollectionsList [private]

Definition at line 153 of file MultiPan.h.

Referenced by addEntryHelperFn(), MultiPan(), and ~MultiPan().

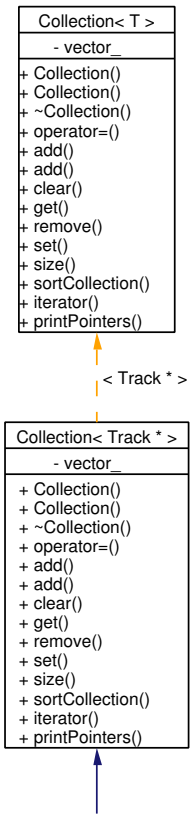
The documentation for this class was generated from the following files:

- MultiPan.h
- MultiPan.cpp

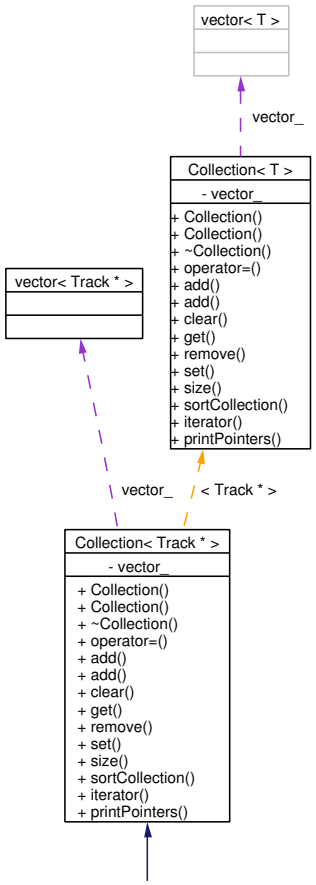
8.35 MultiTrack Class Reference

`#include <MultiTrack.h>`

Inheritance diagram for MultiTrack:



Collaboration diagram for MultiTrack:



Public Member Functions

- MultiTrack ()
- MultiTrack (int channels, m_sample_count_type numSamples, m_rate_type samplingRate)
- MultiTrack (MultiTrack &mt)
- MultiTrack & operator= (MultiTrack &mt)
- ~MultiTrack ()
- void composite (MultiTrack &mt, m_time_type startTime=0)

8.35.1 Detailed Description

A MultiTrack is basically a collection of track pointers. When the MultiTrack object is deleted, it deletes each of the tracks in the collection. When it is copied or assigned, it copies the underlying objects. When tracks are removed or replaced, they are not deleted.

The idea is that copying entire MultiTrack objects should be covered, but editing elements in the MultiTrack objects should leave users to their own object management. Just remember, a MultiTrack owns a pointer when it is added to it.

Author:
Braden Kowitz

Definition at line 50 of file MultiTrack.h.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 MultiTrack::MultiTrack ()

This is the default constructor which creates a MultiTrack with 0 tracks.

Definition at line 38 of file MultiTrack.cpp.

8.35.2.2 MultiTrack::MultiTrack (int *channels*, m_sample_count_type *numSamples*, m_rate_type *samplingRate*)

This constructor creates an empty MultiTrack.

- This is good for composition.
- It also zeros out data if requested.

Parameters:
channels The number of channels
numSamples The number of samples
samplingRate The sampling rate

Definition at line 44 of file MultiTrack.cpp.

References Collection< Track * >::add(), m_rate_type, and m_sample_count_type.

8.35.2.3 MultiTrack::MultiTrack (MultiTrack & *mt*)

This is a copy Constructor.

Todo
The argument should be const.

Parameters:
mt The MultiTrack to copy

Definition at line 60 of file MultiTrack.cpp.

References Collection< Track * >::add(), Iterator< T >::hasNext(), Collection< Track * >::iterator(), and Iterator< T >::next().

8.35.2.4 MultiTrack::~MultiTrack ()

This is a destructor.

Definition at line 91 of file MultiTrack.cpp.

References Collection< Track * >::clear(), Iterator< T >::hasNext(), Collection< Track * >::iterator(), and Iterator< T >::next().

8.35.3 Member Function Documentation

8.35.3.1 void MultiTrack::composite (MultiTrack & *mt*, m_time_type *startTime* = 0)

This composites another MultiTrack object on top of this one.

- Any extra tracks are skipped (with warning).
- startTime will offset the argument MultiTrack before compositing.

Definition at line 103 of file MultiTrack.cpp.

References Iterator< T >::hasNext(), Collection< Track * >::iterator(), m_time_type, and Iterator< T >::next().

Referenced by Score::render().

8.35.3.2 MultiTrack & MultiTrack::operator= (MultiTrack & *mt*)

This is an overloaded assignment operator.

Todo
The argument should be const.

Parameters:
mt The MultiTrack to assign.

Returns:
A MultiTrack

Definition at line 69 of file MultiTrack.cpp.

References Collection< Track * >::add(), Collection< Track * >::clear(), Iterator< T >::hasNext(), Collection< Track * >::iterator(), and Iterator< T >::next().

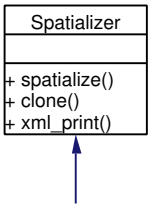
The documentation for this class was generated from the following files:

- MultiTrack.h
- MultiTrack.cpp

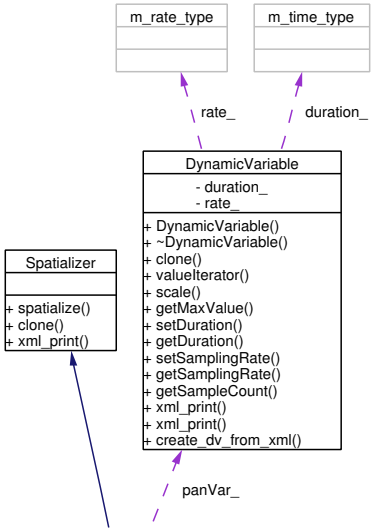
8.36 Pan Class Reference

`#include <Pan.h>`

Inheritance diagram for Pan:



Collaboration diagram for Pan:



Public Member Functions

- Pan (DynamicVariable &v)
- Pan * clone ()
- void set (DynamicVariable &v)
- MultiTrack * spatialize (Track &t, int numTracks)
- void xml_print (ofstream &xmlOutput)

Private Attributes

- DynamicVariable * panVar_

8.36.1 Detailed Description

Pan is a simple Spatializer. It implements simple panning accross a number of channels.

Todo
Add destructor to Spatializer.

Author:
Braden Kowitz

Definition at line 45 of file Pan.h.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 Pan::Pan (DynamicVariable & *v*)

This is a simple constructor which creates a pan object around a dynamic variable. The range should be [0,1] (else undefined).

Parameters:
v The DynamicVariable

Definition at line 38 of file Pan.cpp.

Referenced by clone().

8.36.3 Member Function Documentation

8.36.3.1 Pan * Pan::clone () [virtual]

This returns an exact duplicate of this Pan object.

Returns:
The new Pan object

Reimplemented from Spatializer.

Definition at line 47 of file Pan.cpp.

References Pan(), and panVar_.

8.36.3.2 void Pan::set (DynamicVariable & *v*)

This sets the dynamic variable for a Pan object to something different than specified in the constructor.

Parameters:
v The new DynamicVariable

Definition at line 41 of file Pan.cpp.

References DynamicVariable::clone(), and panVar_.

8.36.3.3 MultiTrack * Pan::spatialize (Track & *t*, int *numTracks*) [virtual]

This will return a new MultiTrack object with numTracks. The given track will be panned accross the channels.

Parameters:
t The Track to pan

numTracks The number of tracks

Returns:
A pointer to a new MultiTrack

Reimplemented from Spatializer.

Definition at line 52 of file Pan.cpp.

References Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), SoundSample::getSamplingRate(), Track::getWave(), m_rate_type, m_sample_count_type, m_value_type, Iterator< T >::next(), panVar_, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), and DynamicVariable::valueIterator().

8.36.3.4 void Pan::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

Reimplemented from Spatializer.

Definition at line 101 of file Pan.cpp.

8.36.4 Member Data Documentation

8.36.4.1 DynamicVariable* Pan::panVar_ [private]

Definition at line 48 of file Pan.h.

Referenced by clone(), set(), and spatialize().

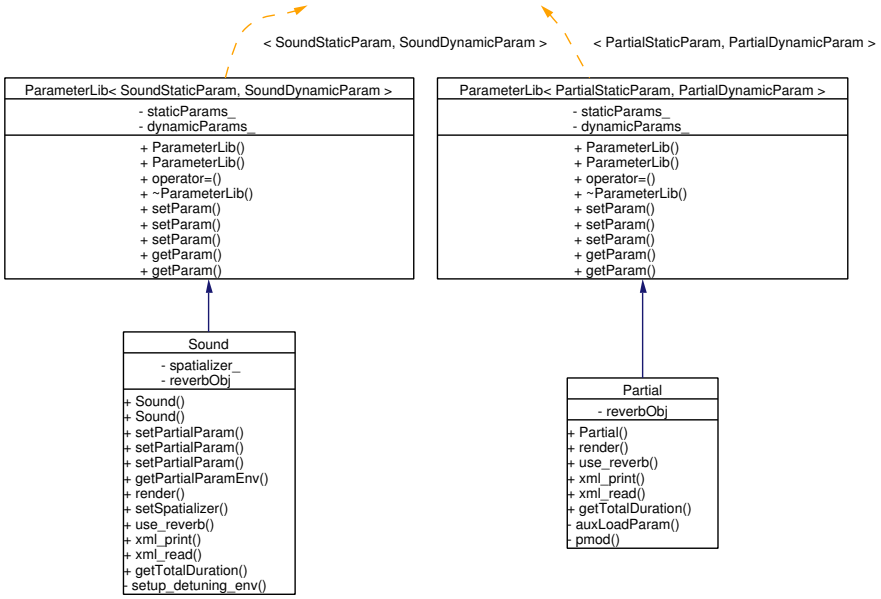
The documentation for this class was generated from the following files:

- Pan.h
- Pan.cpp

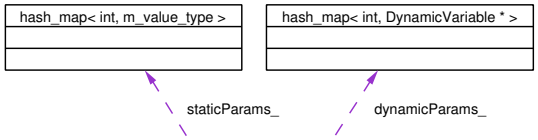
8.37 ParameterLib< StaticT, DynamicT > Class Template Reference

`#include <ParameterLib.h>`

Inheritance diagram for ParameterLib< StaticT, DynamicT >:



Collaboration diagram for ParameterLib< StaticT, DynamicT >:



Public Member Functions

- ParameterLib ()
- ParameterLib (const ParameterLib< StaticT, DynamicT > &pl)
- ParameterLib< StaticT, DynamicT > & operator= (const ParameterLib< StaticT, DynamicT > &pl)
- virtual ~ParameterLib ()
- void setParam (DynamicT p, DynamicVariable &v)
- void setParam (DynamicT p, m_value_type v)
- DynamicVariable & getParam (DynamicT p)
- void setParam (StaticT p, m_value_type v)
- m_value_type getParam (StaticT p)

Private Attributes

- hash_map< int, m_value_type > staticParams_
- hash_map< int, DynamicVariable * > dynamicParams_

8.37.1 Detailed Description

template<class StaticT, class DynamicT> class ParameterLib< StaticT, DynamicT >

A parameter lib object stores and retrieves DynamicVariables and static variables by given indices which are represented as enums.

Author:
Braden Kowitz

Definition at line 48 of file ParameterLib.h.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 template<class StaticT, class DynamicT> ParameterLib< StaticT, DynamicT >::ParameterLib ()

This is the default constructor.

Definition at line 41 of file ParameterLib.cpp.

8.37.2.2 template<class StaticT, class DynamicT> ParameterLib< StaticT, DynamicT >::ParameterLib (const ParameterLib< StaticT, DynamicT > & *pl*)

This is a copy constructor, which makes a duplicate of the given ParameterLib

Parameters:

pl The ParameterLib to copy

Definition at line 49 of file ParameterLib.cpp.

References ParameterLib< StaticT, DynamicT >::dynamicParams_, and ParameterLib< StaticT, DynamicT >::staticParams_.

8.37.2.3 template<class StaticT, class DynamicT> ParameterLib< StaticT, DynamicT >::~ParameterLib () [virtual]

This is the destructor.

Definition at line 116 of file ParameterLib.cpp.

8.37.3 Member Function Documentation

8.37.3.1 template<class StaticT, class DynamicT> m_value_type ParameterLib< StaticT, DynamicT >::getParam (StaticT *p*)

This returns a static parameter of this object. If no parameter was ever set, it returns 0.0.

Parameters:

p The position of the static parameter

Returns:

The value of the static parameter

Definition at line 231 of file ParameterLib.cpp.

8.37.3.2 template<class StaticT, class DynamicT> DynamicVariable & ParameterLib< StaticT, DynamicT >::getParam (DynamicT *p*)

Returns a reference to the dynamic variable at a position. If no variable exists, a Constant object set to zero is used instead.

Parameters:

p The position of the DynamicVariable

Returns:

A reference to the DynamicVariable

Definition at line 178 of file ParameterLib.cpp.

8.37.3.3 template<class StaticT, class DynamicT> ParameterLib< StaticT, DynamicT > & ParameterLib< StaticT, DynamicT >::operator= (const ParameterLib< StaticT, DynamicT > & *pl*)

This is an overloaded assignment operator.

Parameters:

pl The ParameterLib to assign

Definition at line 73 of file ParameterLib.cpp.

References ParameterLib< StaticT, DynamicT >::dynamicParams_, and ParameterLib< StaticT, DynamicT >::staticParams_.

8.37.3.4 template<class StaticT, class DynamicT> void ParameterLib< StaticT, DynamicT >::setParam (StaticT *p*, m_value_type *v*)

This sets a static parameter of this object.

Parameters:

p The static parameter

v The new value

Definition at line 222 of file ParameterLib.cpp.

References m_value_type.

8.37.3.5 template<class StaticT, class DynamicT> void ParameterLib< StaticT, DynamicT >::setParam (DynamicT *p*, m_value_type *v*)

This sets a dynamic variable with an m_value_type. Behind the scenes, a Constant object is created.

Parameters:

p The DynamicVariable to set

v The new value type

Definition at line 168 of file ParameterLib.cpp.

References m_value_type.

8.37.3.6 template<class StaticT, class DynamicT> void ParameterLib< StaticT, DynamicT >::setParam (DynamicT *p*, DynamicVariable & *v*)

This sets a DynamicVariable to the specified setting. The dynamic variable is copied into this object.

Parameters:

p The DynamicVariable to set

v The new DynamicVariable

Definition at line 148 of file ParameterLib.cpp.

References DynamicVariable::clone().

8.37.4 Member Data Documentation

8.37.4.1 template<class StaticT, class DynamicT> hash_map<int , DynamicVariable*> ParameterLib< StaticT, DynamicT >::dynamicParams_ [private]

Holds the dynamic parameters in an association

Definition at line 142 of file ParameterLib.h.

Referenced by ParameterLib< StaticT, DynamicT >::operator=(), and ParameterLib< StaticT, DynamicT >::ParameterLib().

8.37.4.2 template<class StaticT, class DynamicT> hash_map<int , m_value_type> ParameterLib< StaticT, DynamicT >::staticParams_ [private]

Holds the static parameters in an association

Definition at line 137 of file ParameterLib.h.

Referenced by ParameterLib< StaticT, DynamicT >::operator=(), and ParameterLib< StaticT, DynamicT >::ParameterLib().

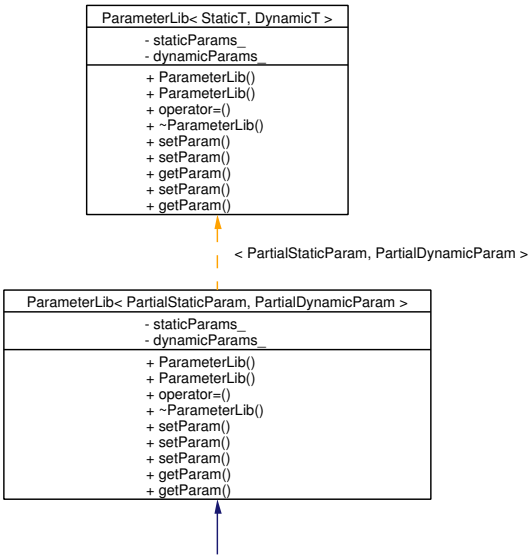
The documentation for this class was generated from the following files:

- ParameterLib.h
- ParameterLib.cpp

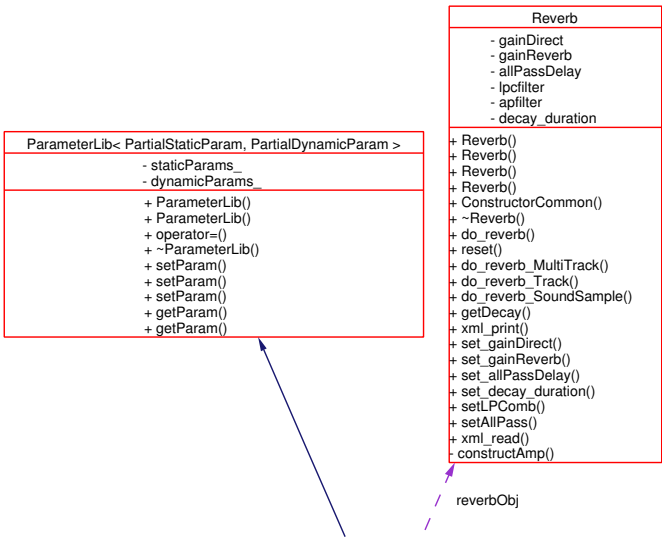
8.38 Partial Class Reference

`#include <Partial.h>`

Inheritance diagram for Partial:



Collaboration diagram for Partial:



Public Member Functions

- Partial ()
- Track * render (m_sample_count_type sampleCount, m_time_type duration, m_rate_type samplingRate=DEFAULT_SAMPLING_RATE)
- void use_reverb (Reverb *newReverbObj)
- void xml_print (ofstream &xmlOutput, list< Reverb * > &revObjs, list< DynamicVariable * > &dynObjs)
- void xml_read (XmlReader::xmltag *partialtag, hash_map< long, Reverb * > *reverbHash, hash_map< long, DynamicVariable * > *dvHash)
- m_time_type getTotalDuration (m_time_type dryDuration)

Private Member Functions

- void auxLoadParam (enum PartialDynamicParam param, XmlReader::xmltag *tag, hash_map< long, DynamicVariable * > *dvHash)
- m_value_type pmod (m_value_type num)

Private Attributes

- Reverb * reverbObj

8.38.1 Detailed Description

A partial is a building block of all sounds. It is a simple Sin wave that is manipulated by several dynamic and static parameters.

Definition at line 163 of file Partial.h.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 Partial::Partial ()

Default constructor that sets a few basic parameters

- RELATIVE_AMPLITUDE = 1.0
- PARTIAL_NUM = 1.0
- WAVE_SHAPE = 1.0
- FREQUENCY = 440
- LOUDNESS_SCALAR = 1.0
- All trans envelopes = 0
- Both TRANS_WIDTH = 1103

Definition at line 41 of file Partial.cpp.

References AMPTRANS_AMP_ENV, AMPTRANS_RATE_ENV, AMPTRANS_WIDTH, FREQ_ENV, FREQTRANS_AMP_ENV, FREQTRANS_RATE_ENV, FREQTRANS_WIDTH, FREQUENCY, LOUDNESS_SCALAR, PARTIAL_NUM, RELATIVE_AMPLITUDE, reverbObj, ParameterLib< PartialStaticParam, PartialDynamicParam >::setParam(), WAVE_SHAPE, and WAVE_TYPE.

8.38.3 Member Function Documentation

8.38.3.1 void Partial::auxLoadParam (enum PartialDynamicParam *param*, XmlReader::xmltag * *tag*, hash_map< long, DynamicVariable * > * *dvHash*) [private]

Deprecated

Auxillary function to assist in loading dv's from XML

Definition at line 556 of file Partial.cpp.

References DynamicVariable::create_dv_from_xml(), XmlReader::xmltag::findChildParamValue(), and ParameterLib< PartialStaticParam, PartialDynamicParam >::setParam().

Referenced by xml_read().

8.38.3.2 m_time_type Partial::getTotalDuration (m_time_type *dryDuration*)

This returns the total length (in seconds) of the partial.

Parameters:

dryDuration The input 'dryDuration' is the duration of the sound without any effects added on.

Returns:

The total length of time returned includes time for the reverb to die out.

Definition at line 401 of file Partial.cpp.

References Reverb::getDecay(), m_time_type, and reverbObj.

8.38.3.3 m_value_type Partial::pmod (m_value_type *num*) [inline, private]

This is phase-modulation. It basically does an inline modulus 1 on a float value.

Definition at line 386 of file Partial.cpp.

References m_value_type.

Referenced by render().

8.38.3.4 Track * Partial::render (m_sample_count_type *sampleCount*, m_time_type *duration*, m_rate_type *samplingRate* = DEFAULT_SAMPLING_RATE)

This returns a Track object of the rendered partial. The object must be deleted by the user calling the function.

Parameters:

sampleCount The number of samples

duration The duration

samplingRate The sampling rate

Returns:
A Track

Definition at line 64 of file Partial.cpp.

References AMPTRANS_AMP_ENV, AMPTRANS_RATE_ENV, AMPTRANS_WIDTH, DynamicVariable::clone(), Reverb::do_reverb_Track(), FREQ_ENV, FREQTRANS_AMP_ENV, FREQTRANS_RATE_ENV, FREQTRANS_WIDTH, FREQUENCY, Reverb::getDecay(), ParameterLib< PartialStaticParam, PartialDynamicParam >::getParam(), LOUDNESS_SCALAR, m_rate_type, m_sample_count_type, m_time_type, m_value_type, Iterator< m_value_type >::next(), PARTIAL_NUM, PHASE, pmod(), reverbObj, DynamicVariable::setDuration(), DynamicVariable::setSamplingRate(), TREMOLO_AMP, TREMOLO_RATE, DynamicVariable::valueIterator(), VIBRATO_AMP, VIBRATO_RATE, WAVE_SHAPE, and WAVE_TYPE.

8.38.3.5 void Partial::use_reverb (Reverb * *newReverbObj*)

Use this object to perform reverb in the render() method

Parameters:
newReverbObj A pointer to a Reverb object

Definition at line 395 of file Partial.cpp.

References reverbObj.

Referenced by xml_read().

8.38.3.6 void Partial::xml_print (ofstream & *xmlOutput*, list< Reverb * > & *revObjs*, list< DynamicVariable * > & *dynObjs*)

Deprecated
This outputs an XML representation of the object to STDOUT

Definition at line 407 of file Partial.cpp.

References FREQ_ENV, FREQUENCY, ParameterLib< PartialStaticParam, PartialDynamicParam >::getParam(), LOUDNESS_SCALAR, PARTIAL_NUM, PHASE, RELATIVE_AMPLITUDE, reverbObj, TREMOLO_AMP, TREMOLO_RATE, VIBRATO_AMP, VIBRATO_RATE, WAVE_SHAPE, and DynamicVariable::xml_print().

8.38.3.7 void Partial::xml_read (XmlReader::xmltag * *partialtag*, hash_map< long, Reverb * > * *reverbHash*, hash_map< long, DynamicVariable * > * *dvHash*)

Deprecated

Definition at line 486 of file Partial.cpp.

References auxLoadParam(), XmlReader::xmltag::children, XmlReader::xmltag::findChildParamValue(), FREQ_ENV, FREQUENCY, LOUDNESS_SCALAR, XmlReader::xmltag::name, XmlReader::xmltagset::next, PARTIAL_NUM, PHASE, RELATIVE_AMPLITUDE, ParameterLib< PartialStaticParam, PartialDynamicParam >::setParam(), XmlReader::xmltagset::tag, TREMOLO_AMP, TREMOLO_RATE, use_reverb(), VIBRATO_AMP, VIBRATO_RATE, and WAVE_SHAPE.

Referenced by Sound::xml_read().

8.38.4 Member Data Documentation

8.38.4.1 Reverb* Partial::reverbObj [private]

A Reverb object *

Definition at line 231 of file Partial.h.

Referenced by getTotalDuration(), Partial(), render(), use_reverb(), and xml_print().

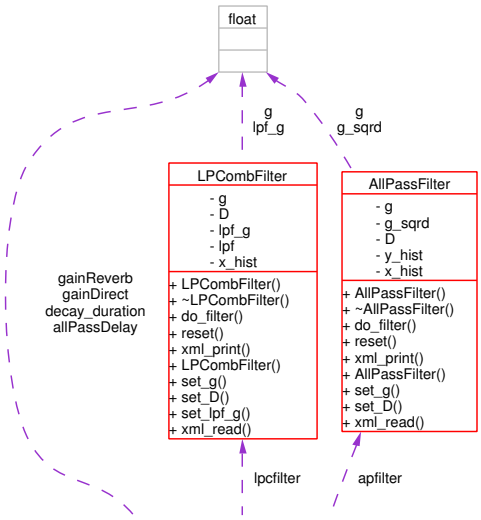
The documentation for this class was generated from the following files:

- Partial.h
- Partial.cpp

8.39 Reverb Class Reference

`#include <Reverb.h>`

Collaboration diagram for Reverb:



Public Member Functions

- Reverb (m_rate_type samplingRate)
- Reverb (float room_size, m_rate_type samplingRate)
- Reverb (float reverbPercent, float hilow_spread, float gainAllPass, float delay, m_rate_type samplingRate)

- Reverb (float reverbPercent, float *combGainList, float *lpGainList, float gainAllPass, float delay, m_rate_type samplingRate)
- void ConstructorCommon (float percentReverb, float *combGainList, float *lpGainList, float gainAllPass, float delay, m_rate_type samplingRate)
- ~Reverb ()
- m_sample_type do_reverb (m_sample_type x_t)
- void reset (void)
- MultiTrack & do_reverb_MultiTrack (MultiTrack &inWave)
- Track & do_reverb_Track (Track &inWave)
- SoundSample * do_reverb_SoundSample (SoundSample *inWave)
- float getDecay (void)
- void xml_print (ofstream &xmlOutput)
- void set_gainDirect (float new_gainDirect)
- void set_gainReverb (float new_gainReverb)
- void set_allPassDelay (float new_allPassDelay)
- void set_decay_duration (float new_decay_duration)
- void setLPComb (int idx, LPCombFilter *f)
- void setAllPass (AllPassFilter *f)
- void xml_read (XmlReader::xmltag *reverbtag)

Private Member Functions

- SoundSample * constructAmp (SoundSample *wave)

Private Attributes

- float gainDirect
- float gainReverb
- float allPassDelay
- LPCombFilter * lpcfilter [REVERB_NUM_COMB_FILTERS]
- AllPassFilter * apfilter
- float decay_duration

8.39.1 Detailed Description

This is an artificial reverberator class. The Reverb class implements an artificial reverberator, built on the model in Moore's "Elements of Computer Music" book. That model, in turn, is based on Schroder's and Moorer's work.

In order to use this class you need to do three things 1. Create a reverb object using the parameters you want. There are a few different constructors to choose from, depending on how much control you want to have (and thus, how much you want to be burdened

with details). 2. Call a Sound, Track, or MultiTrack object's use_reverb method and pass in your Reverb object as the parameter. 3. Render the corresponding object and let reverb run (the sound, track, or multitrack will call Reverb::do_reverb for you once you call its use_reverb member to setup a reverb object).

Note:

Reverb relies on some filters with feedback and delay, thus the filters are partially dependent on old data sent to Reverb. In order to clear out the history in these filters, call Reverb::Reset before reusing a Reverb object.

Author:

Andrew Kurtz
Jim Lindstrom

Definition at line 71 of file Reverb.h.

8.39.2 Constructor & Destructor Documentation

8.39.2.1 Reverb::Reverb (m_rate_type *samplingRate*)

This is the default constructor which uses default values for everything.

Parameters:

samplingRate The number of samples per second that the reverb will assume on any data it is fed

Definition at line 49 of file Reverb.cpp.

References ConstructorCommon(), m_rate_type, and REVERB_NUM_COMB_-FILTERS.

8.39.2.2 Reverb::Reverb (float *room_size*, m_rate_type *samplingRate*)

This is a simple constructor.

Parameters:

room_size this is constructed to give some reasonable results when you input room_size values between 0.0 and 1.0. I found some parameter sets that I felt approximated various familiar rooms (lincoln hall, MB 1201, great hall, a large stadium, etc), and mapped these into a room_size space of 0.0 to 1.0. Then I used gnumeric (spreadsheet for linux, like excel) to find a linear regression to get all the parameters simply in terms of room_size. Thus a value of 0.0 will have virtually no reverb. A value of 1.0 will have a very high degree of reverb.

samplingRate The number of samples per second that the reverb will assume on any data it is fed

Definition at line 70 of file Reverb.cpp.

References ConstructorCommon(), m_rate_type, and REVERB_NUM_COMB_-FILTERS.

8.39.2.3 Reverb::Reverb (float *reverbPercent*, float *hilow_spread*, float *gainAllPass*, float *delay*, m_rate_type *samplingRate*)

This is a moderately advanced constructor.

Parameters:

reverbPercent This determines the mix (in the resultant sound) between the direct sound (the input sound) and the reverbed sound. The value 0.0 is no reverb, all direct sound; 1.0 is all reverb, no direct sound.

hilow_spread In most rooms the low frequecy response will be higher than the high frequency response. If we hold other parameters constant, the math gives a range of possible ratios between low frequence to high frequence response. A value of 1.0 will choose the maximum low-frequence response, holding the high-frequency response at a given. A value of 0.0 will mean an even response across all frequencies.

gainAllPass Moore's reverberator has only one all-pass filter, and this is its gain. The default value is 0.7. Higher values (up to 1.0) cause more ringing at certain frequencies. Lower values (down to 0.0) cause less ringing.

delay This is the length (in seconds) of the first echo response.

samplingRate The number of samples per second that the reverb will assume on any data it is fed

Definition at line 95 of file Reverb.cpp.

References ConstructorCommon(), m_rate_type, and REVERB_NUM_COMB_-FILTERS.

8.39.2.4 Reverb::Reverb (float *reverbPercent*, float * *combGainList*, float * *lpGainList*, float *gainAllPass*, float *delay*, m_rate_type *samplingRate*)

This is an advanced constructor.

Parameters:

reverbPercent This determines the mix (in the resultant sound) between the direct sound (the input sound) and the reverbed sound. The value 0.0 is no reverb, all direct sound; 1.0 is all reverb, no direct sound.

combGainList This is the address of an array of 6 floats that are the comb filter gains.

lpGainList This is the address of an array of 6 floats that are the low-pass filter gains.

gainAllPass Moore's reverberator has only one all-pass filter, and this is its gain. The default value is 0.7. Higher values (up to 1.0) cause more ringing at certain frequencies. Lower values (down to 0.0) cause less ringing.

delay The length (in seconds) of the first echo response

samplingRate The number of samples per second that the reverb will assume on any data it is fed

Definition at line 140 of file Reverb.cpp.

References ConstructorCommon(), and m_rate_type.

8.39.2.5 Reverb::~Reverb ()

This is the destructor. This will destroy the filters and release any dynamically allocated memory.

Definition at line 219 of file Reverb.cpp.

References apfilter, lpcfilter, and REVERB_NUM_COMB_FILTERS.

8.39.3 Member Function Documentation

8.39.3.1 SoundSample * Reverb::constructAmp (SoundSample * *wave*) [private]

This reconstructs an Amp SoundSample given the Wave SoundSample. The input SoundSample is unmodified. The returned SoundSample has been allocated with 'new' and is the caller's responsibility to 'delete' at some point.

Parameters:

wave A pointer to the 'wave' component of a Track

Returns:

A new SoundSample to be used as the 'amp' component of a Track

Definition at line 379 of file Reverb.cpp.

References SoundSample::getSampleCount(), SoundSample::getSamplingRate(), m_rate_type, m_sample_count_type, and m_sample_type.

Referenced by do_reverb_MultiTrack(), and do_reverb_Track().

8.39.3.2 void Reverb::ConstructorCommon (float *percentReverb*, float * *combGainList*, float * *lpGainList*, float *gainAllPass*, float *delay*, m_rate_type *samplingRate*)

This contains common code used by all constructors

Parameters:

percentReverb This determines the mix (in the resultant sound) between the direct sound (the input sound) and the reverbed sound. The value 0.0 is no reverb, all direct sound; 1.0 is all reverb, no direct sound.

combGainList This is the address of an array of 6 floats that are the comb filter gains.

lpGainList This is the address of an array of 6 floats that are the low-pass filter gains.

gainAllPass Moore's reverberator has only one all-pass filter, and this is its gain. The default value is 0.7. Higher values (up to 1.0) cause more ringing at certain frequencies. Lower values (down to 0.0) cause less ringing.

delay The length (in seconds) of the first echo response

samplingRate The number of samples per second that the reverb will assume on any data it is fed

Definition at line 179 of file Reverb.cpp.

References allPassDelay, apfilter, decay_duration, gainDirect, gainReverb, lpcfilter, m_rate_type, m_time_type, and REVERB_NUM_COMB_FILTERS.

Referenced by Reverb().

8.39.3.3 m_sample_type Reverb::do_reverb (m_sample_type *x_t*)

This method applies reverb to a single sample.

Parameters:

x_t The sample to apply reverb to

Returns:

The new sample

Definition at line 232 of file Reverb.cpp.

References apfilter, AllPassFilter::do_filter(), LPCombFilter::do_filter(), gainDirect, gainReverb, lpcfilter, m_sample_type, and REVERB_NUM_COMB_FILTERS.

Referenced by do_reverb_SoundSample().

8.39.3.4 MultiTrack & Reverb::do_reverb_MultiTrack (MultiTrack & *inWave*)

This method applies reverb to a source MultiTrack, track by track. It does so by decomposing the wave into Track objects (organized with a Collection inside the MultiTrack object) and calling the virtual function, do_reverb(SoundSample *inWave), for each Track.

Parameters:

inWave The MultiTrack to apply reverb to

Returns:

The original MultiTrack remains intact and untouched. The return is a reference to a NEW MultiTrack, which the caller is responsible for deleting when done.

Definition at line 278 of file Reverb.cpp.

References Collection< Track * >::add(), constructAmp(), do_reverb_SoundSample(), Track::getWave(), Iterator< T >::hasNext(), Collection< Track * >::iterator(), Iterator< T >::next(), and reset().

Referenced by Score::render().

8.39.3.5 SoundSample * Reverb::do_reverb_SoundSample (SoundSample * *inWave*)

This method calls do_reverb(m_sample_type x) for each sample in a track and builds a new SoundSample on the fly.

Parameters:

inWave The SoundSample to apply reverb to

Returns:

The original TraSoundSample remains intact and untouched. The return is a reference to a NEW SoundSample, which the caller is responsible for deleting when done.

Definition at line 330 of file Reverb.cpp.

References do_reverb(), SoundSample::getSampleCount(), and SoundSample::getSamplingRate().

Referenced by do_reverb_MultiTrack(), and do_reverb_Track().

8.39.3.6 Track & Reverb::do_reverb_Track (Track & *inWave*)

This method applies the reverb to a source Track by calling the virtual function, do_reverb(SoundSample *inWave)

Parameters:

inWave The Track to apply reverb to

Returns:

The original Track remains intact and untouched. The return is a reference to a NEW Track, which the caller is responsible for deleting when done.

Definition at line 311 of file Reverb.cpp.

References constructAmp(), do_reverb_SoundSample(), Track::getWave(), and reset().

Referenced by Sound::render(), and Partial::render().

8.39.3.7 float Reverb::getDecay (void)

This retrieves the decay length. It is intended to be used by Sound, Score, and Partial objects to figure out how many samples to allocate for a sound. I.e., the object must add enough cycles to the end of a sound for the delay to die out. It is in units of seconds.

Returns:

The decay time (in seconds)

Definition at line 349 of file Reverb.cpp.

References decay_duration.

Referenced by Sound::getTotalDuration(), Partial::getTotalDuration(), Score::render(), and Partial::render().

8.39.3.8 void Reverb::reset (void)

This method resets internal parameters in the reverberator to an initial state. The Reverb class relies on some filters with feedback and delay, thus the filters are partially dependent on old data sent to Reverb. In order to clear out the history in these filters, call Reverb::Reset before reusing a Reverb object.

Definition at line 266 of file Reverb.cpp.

References apfilter, lpcfilter, AllPassFilter::reset(), LPCombFilter::reset(), and REVERB_NUM_COMB_FILTERS.

Referenced by do_reverb_MultiTrack(), and do_reverb_Track().

8.39.3.9 void Reverb::set_allPassDelay (float *new_allPassDelay*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 518 of file Reverb.cpp.

References allPassDelay.

Referenced by xml_read().

8.39.3.10 void Reverb::set_decay_duration (float *new_decay_duration*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 522 of file Reverb.cpp.

References decay_duration.

Referenced by xml_read().

8.39.3.11 void Reverb::set_gainDirect (float *new_gainDirect*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 510 of file Reverb.cpp.

References gainDirect.

Referenced by xml_read().

8.39.3.12 void Reverb::set_gainReverb (float *new_gainReverb*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 514 of file Reverb.cpp.

References gainReverb.

Referenced by xml_read().

8.39.3.13 void Reverb::setAllPass (AllPassFilter * *f*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 530 of file Reverb.cpp.

References apfilter.

Referenced by xml_read().

8.39.3.14 void Reverb::setLPComb (int *idx*, LPCombFilter * *f*)

Used to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 526 of file Reverb.cpp.

References lpcfilter.

Referenced by xml_read().

8.39.3.15 void Reverb::xml_print (ofstream & *xmlOutput*)

Deprecated
This outputs an XML representation of the object to STDOUT

Definition at line 440 of file Reverb.cpp.

References allPassDelay, apfilter, decay_duration, gainDirect, gainReverb, lpcfilter, REVERB_NUM_COMB_FILTERS, AllPassFilter::xml_print(), and LPCombFilter::xml_print().

8.39.3.16 void Reverb::xml_read (XmlReader::xmltag * *reverbtag*)

Deprecated
Used by XML parsing code to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Definition at line 461 of file Reverb.cpp.

References XmlReader::xmltag::children, XmlReader::xmltag::findChildParamValue(), XmlReader::xmltag::name, set_allPassDelay(), set_decay_duration(), set_gainDirect(), set_gainReverb(), setAllPass(), setLPComb(), AllPassFilter::xml_read(), and LPCombFilter::xml_read().

Referenced by Score::xml_read().

8.39.4 Member Data Documentation

8.39.4.1 float Reverb::allPassDelay [private]

Definition at line 293 of file Reverb.h.

Referenced by ConstructorCommon(), set_allPassDelay(), and xml_print().

8.39.4.2 AllPassFilter* Reverb::apfilter [private]

Definition at line 295 of file Reverb.h.

Referenced by ConstructorCommon(), do_reverb(), reset(), setAllPass(), xml_print(), and ~Reverb().

8.39.4.3 float Reverb::decay_duration [private]

Definition at line 296 of file Reverb.h.

Referenced by ConstructorCommon(), getDecay(), set_decay_duration(), and xml_print().

8.39.4.4 float Reverb::gainDirect [private]

Definition at line 291 of file Reverb.h.

Referenced by ConstructorCommon(), do_reverb(), set_gainDirect(), and xml_print().

8.39.4.5 float Reverb::gainReverb [private]

Definition at line 292 of file Reverb.h.

Referenced by ConstructorCommon(), do_reverb(), set_gainReverb(), and xml_print().

8.39.4.6 LPCombFilter* Reverb::lpcfilter[REVERB_NUM_COMB_FILTERS] [private]

Definition at line 294 of file Reverb.h.

Referenced by ConstructorCommon(), do_reverb(), reset(), setLPComb(), xml_print(), and ~Reverb().

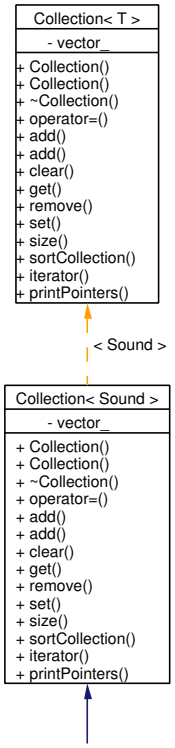
The documentation for this class was generated from the following files:

- Reverb.h
- Reverb.cpp

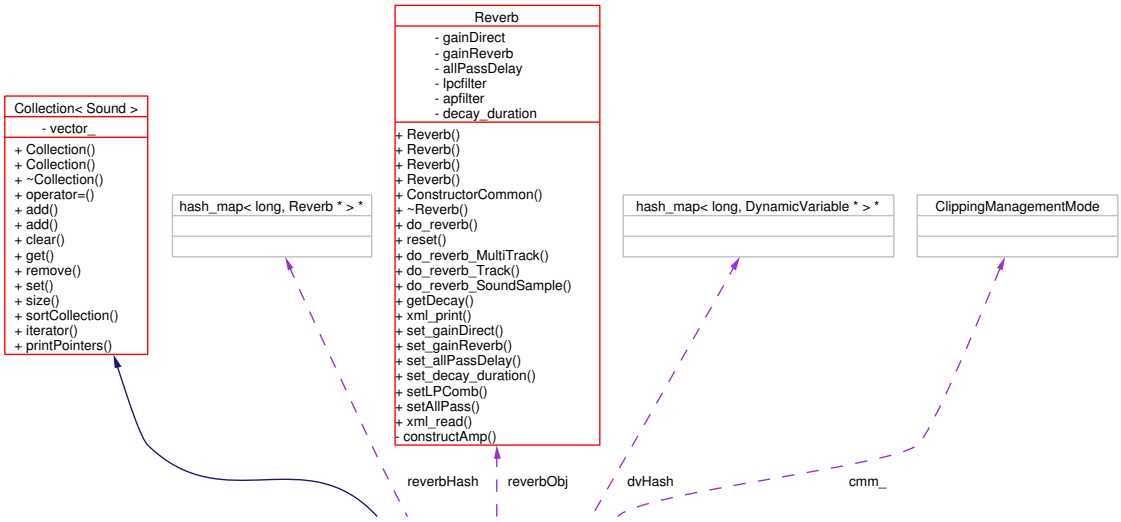
8.40 Score Class Reference

`#include <Score.h>`

Inheritance diagram for Score:



Collaboration diagram for Score:



Public Types

- enum ClippingManagementMode {
  NONE,
  CLIP,
  SCALE,
  CHANNEL_SCALE,
  ANTICLIP,
  CHANNEL_ANTICLIP }

Public Member Functions

- Score ()

- MultiTrack * render (int numChannels, m_rate_type samplingRate=DEFAULT_SAMPLING_RATE)
- MultiTrack * render (int numChannels, m_rate_type samplingRate, int nThreads)
- void setClippingManagementMode (ClippingManagementMode mode)
- ClippingManagementMode getClippingManagementMode ()
- void use_reverb (Reverb *newReverbObj)
- void xml_print ()
- void xml_print (ofstream &xmlOutput)
- void xml_print (const char *xmlOutputPath)
- void xml_read (XmlReader::xmltag *scoretag)

Static Public Member Functions

- void manageClipping (MultiTrack *mt, ClippingManagementMode mode)

Public Attributes

- hash_map< long, Reverb * > * reverbHash
- hash_map< long, DynamicVariable * > * dvHash

Static Private Member Functions

- void clip (MultiTrack *mt)
- void scale (MultiTrack *mt)
- void channelScale (MultiTrack *mt)
- void anticlip (MultiTrack *mt)
- void channelAnticlip (MultiTrack *mt)

Private Attributes

- ClippingManagementMode cmm_
- Reverb * reverbObj

8.40.1 Detailed Description

A Score simply consists of a collection of Sounds. In addition to this, it provides functionality for managing clipping in a piece.

Author:
Braden Kowitz

Definition at line 71 of file Score.h.

8.40.2 Member Enumeration Documentation

8.40.2.1 enum Score::ClippingManagementMode

This sets the clipping management mode for this score. This post-process is run after the score is rendered.

Enumeration values:

NONE • No clipping management is taken at all.
- This lets the composer render once, then try different post-processes, saving some time.

CLIP • Any value over 1 or under -1 is clipped to these limits.

SCALE • The max amplitude value is found in the entire score (all tracks).
- Each track is then scaled by 1/maxAmplitude.

CHANNEL_SCALE • For each track, a max amplitude value if found.
- This track is then scaled by 1/maxAmplitude

ANTICLIP • If the total amplitude accross tracks at any given time is greater than 1, then that sample is scaled on all tracks by 1/totalAmplitude.

CHANNEL_ANTICLIP • For each channel, and each sample
- if a sample has an amplitude greater than 1, then that sample is scaled by 1/amplitude.

Definition at line 125 of file Score.h.

Referenced by getClippingManagementMode(), and xml_read().

8.40.3 Constructor & Destructor Documentation

8.40.3.1 Score::Score ()

This is the default constructor. It sets the ClippingManagementMode to NONE.

Definition at line 60 of file Score.cpp.

References reverbObj.

8.40.4 Member Function Documentation

8.40.4.1 void Score::anticlip (MultiTrack * *mt*) [static, private]

This private function unclips the MultiTrack.

Parameters:

mt The MultiTrack to unclip

Definition at line 390 of file Score.cpp.

References Collection< Track * >::get(), Track::getAmp(), Track::getWave(), m_sample_count_type, m_sample_type, and Collection< Track * >::size().

Referenced by manageClipping().

8.40.4.2 void Score::channelAnticlip (MultiTrack * *mt*) [static, private]

This private function unclips the channels in a MultiTrack.

Parameters:
mt The MultiTrack to unclip

Definition at line 430 of file Score.cpp.

References Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), Track::getWave(), m_sample_count_type, and Collection< Track * >::size().

Referenced by manageClipping().

8.40.4.3 void Score::channelScale (MultiTrack * *mt*) [static, private]

This private function scales the channels in the MultiTrack.

Parameters:
mt The MultiTrack to scale

Definition at line 352 of file Score.cpp.

References Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), Track::getWave(), m_sample_count_type, m_sample_type, and Collection< Track * >::size().

Referenced by manageClipping().

8.40.4.4 void Score::clip (MultiTrack * *mt*) [static, private]

This private function clips the MultiTrack.

Parameters:
mt The MultiTrack to clip

Definition at line 284 of file Score.cpp.

References Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), Track::getWave(), m_sample_count_type, and Collection< Track * >::size().

Referenced by manageClipping().

8.40.4.5 Score::ClippingManagementMode Score::getClippingManagementMode ()

This function gets the current ClippingManagementMode for this score.

Returns:
The ClippingManagementMode

Definition at line 251 of file Score.cpp.

References ClippingManagementMode, and cmm_.

Referenced by xml_print().

8.40.4.6 void Score::manageClipping (MultiTrack * *mt*, ClippingManagementMode *mode*) [static]

This function manages clipping on a MultiTrack object with the specified mode. This is performed automatically when a score is rendered, but is available for the user to render once, then post-process many times.

Parameters:
mt The MultiTrack to clip

mode The ClippingManagementMode

Definition at line 258 of file Score.cpp.

References anticlip(), ANTICLIP, CHANNEL_ANTICLIP, CHANNEL_SCALE, channelAnticlip(), channelScale(), clip(), CLIP, NONE, scale(), and SCALE.

Referenced by render().

8.40.4.7 MultiTrack * Score::render (int *numChannels*, m_rate_type *samplingRate*, int *nThreads*)

This function:

- Renders each sound in this Score using a specified number of threads for parallel rendering
- Composites the sounds into a MultiTrack object
- Post-Processes the sound for clipping management

Note:
The caller must delete this object. Also, the caller must specify a value for samplingRate (unlike single-threaded render)

Parameters:
numChannels The number of channels to render

samplingRate The sampling rate

nThreads The number of threads to use when rendering

Returns:
a MultiTrack object

Definition at line 132 of file Score.cpp.

References Collection< T >::add(), cmm_, MultiTrack::composite(), Reverb::do_reverb_MultiTrack(), threadlist_entry::done, threadlist_entry::finishCondition, Collection< T >::get(), Reverb::getDecay(), ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), Sound::getTotalDuration(), Iterator< T >::hasNext(), Collection< Sound >::iterator(), threadlist_entry::listMutex, m_rate_type, m_sample_count_type, m_time_type, manageClipping(), multithreaded_render_worker(), Iterator< T >::next(), threadlist_entry::numChannels, threadlist_entry::resultTrack, reverbObj, threadlist_entry::samplingRate, Collection< Sound >::size(), threadlist_entry::snd, START_TIME, and threadlist_entry::thread.

8.40.4.8 MultiTrack * Score::render (int *numChannels*, m_rate_type *samplingRate* = DEFAULT_SAMPLING_RATE)

This function:

- Renders each sound in this Score.
- Composites the sounds into a MultiTrack object
- Post-Processes the sound for clipping management

Note:
The caller must delete this object.

Parameters:
numChannels The number of channels to render

samplingRate The sampling rate which defaults to DEFAULT_SAMPLING_RATE

Returns:
A MultiTrack object

Definition at line 67 of file Score.cpp.

References cmm_, MultiTrack::composite(), Reverb::do_reverb_MultiTrack(), Reverb::getDecay(), ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), Sound::getTotalDuration(), Iterator< T >::hasNext(), Collection< Sound >::iterator(), m_rate_type, m_sample_count_type, m_time_type, manageClipping(), Iterator< T >::next(), Sound::render(), reverbObj, and START_TIME.

8.40.4.9 void Score::scale (MultiTrack * *mt*) [static, private]

This private function scales the MultiTrack.

Parameters:
mt The MultiTrack to scale

Definition at line 308 of file Score.cpp.

References Collection< Track * >::get(), Track::getAmp(), SoundSample::getSampleCount(), Track::getWave(), m_sample_count_type, m_sample_type, and Collection< Track * >::size().

Referenced by manageClipping().

8.40.4.10 void Score::setClippingManagementMode (ClippingManagementMode *mode*)

This function sets the ClippingManagementMode for this score.

Parameters:
mode The ClippingManagementMode to set

Definition at line 245 of file Score.cpp.

References cmm_.

Referenced by xml_read().

8.40.4.11 void Score::use_reverb (Reverb * *newReverbObj*)

This function performs reverb in the render() method.

Parameters:
newReverbObj The Reverb object

Definition at line 273 of file Score.cpp.

References reverbObj.

Referenced by xml_read().

8.40.4.12 void Score::xml_print (const char * *xmlOutputPath*)

Deprecated

Definition at line 567 of file Score.cpp.

References xml_print().

8.40.4.13 void Score::xml_print (ofstream & *xmlOutput*)

Deprecated

Definition at line 524 of file Score.cpp.

References getClippingManagementMode(), Iterator< T >::hasNext(), Collection< Sound >::iterator(), Iterator< T >::next(), reverbObj, and Sound::xml_print().

8.40.4.14 void Score::xml_print ()

Deprecated

This outputs an XML representation of the object to STDOUT

Definition at line 580 of file Score.cpp.

Referenced by xml_print().

8.40.4.15 void Score::xml_read (XmlReader::xmltag * *scoretag*)

Deprecated

Definition at line 456 of file Score.cpp.

References Collection< Sound >::add(), XmlReader::xmltag::children, ClippingManagementMode, DynamicVariable::create_dv_from_xml(), dvHash, XmlReader::xmltag::findChildParamValue(), XmlReader::xmltag::getParamValue(), XmlReader::xmltag::name, reverbHash, setClippingManagementMode(), use_reverb(), Sound::xml_read(), and Reverb::xml_read().

8.40.5 Member Data Documentation

8.40.5.1 ClippingManagementMode Score::cmm_ [private]

Definition at line 221 of file Score.h.

Referenced by getClippingManagementMode(), render(), and setClipping-ManagementMode().

8.40.5.2 hash_map<long, DynamicVariable *>* Score::dvHash

Definition at line 218 of file Score.h.

Referenced by xml_read().

8.40.5.3 hash_map<long, Reverb *>* Score::reverbHash

Definition at line 217 of file Score.h.

Referenced by xml_read().

8.40.5.4 Reverb* Score::reverbObj [private]

Definition at line 223 of file Score.h.

Referenced by render(), Score(), use_reverb(), and xml_print().

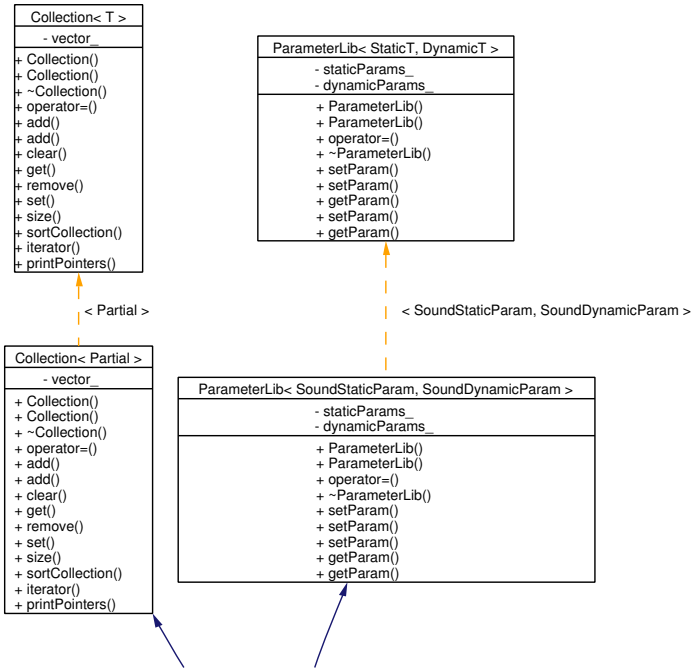
The documentation for this class was generated from the following files:

- Score.h
- Score.cpp

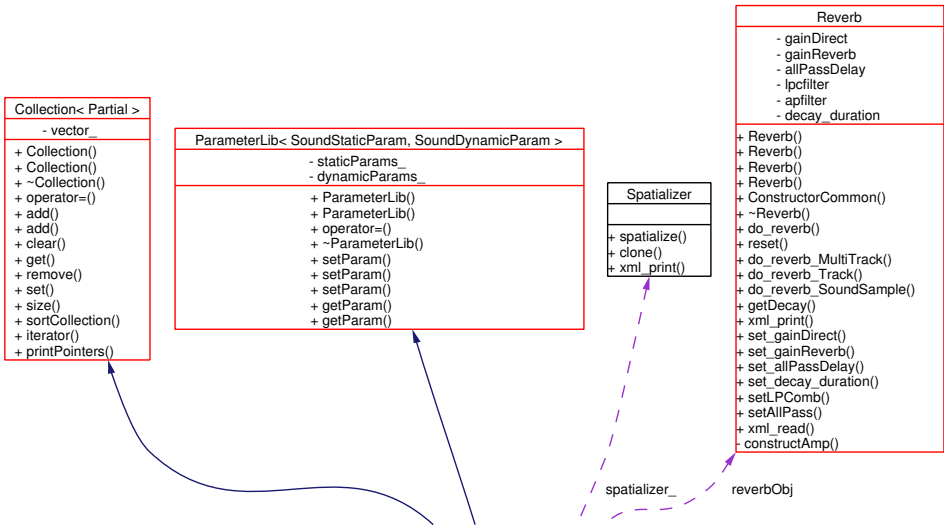
8.41 Sound Class Reference

`#include <Sound.h>`

Inheritance diagram for Sound:



Collaboration diagram for Sound:



Public Member Functions

- Sound ()
- Sound (int numPartials, m_value_type baseFreq)
- void setPartialParam (PartialStaticParam p, m_value_type v)
- void setPartialParam (PartialDynamicParam p, DynamicVariable &v)
- void setPartialParam (PartialDynamicParam p, m_value_type v)
- void getPartialParamEnv (PartialDynamicParam p)
- MultiTrack * render (int numChannels, m_rate_type samplingRate=DEFAULT_SAMPLING_RATE)
- void setSpatializer (Spatializer &s)
- void use_reverb (Reverb *newReverbObj)
- void xml_print (ofstream &xmlOutput, list< Reverb * > &revObjs, list< DynamicVariable * > &dynObjs)
- void xml_read (XmlReader::xmltag *soundtag, hash_map< long, Reverb * > *reverbHash, hash_map< long, DynamicVariable * > *dvHash)
- m_time_type getTotalDuration (void)

Private Member Functions

- void setup_detuning_env (ExponentialInterpolator *env)

Private Attributes

- Spatializer * spatializer_
- Reverb * reverbObj

8.41.1 Detailed Description

Conceptually, A Sound is made up from a set of partials.

Author:
Braden Kowitz

Definition at line 142 of file Sound.h.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 Sound::Sound ()

This is a default constructor that sets a few basic parameters

- DURATION = 1.0
- START_TIME = 0.0
- LOUDNESS = 100 (out of 255) Sones
- LOUDNESS_RATE = 10 Hz

Definition at line 36 of file Sound.cpp.

References DETUNE_DIRECTION, DETUNE_FUNDAMENTAL, DETUNE_SPREAD, DETUNE_VELOCITY, DURATION, LOUDNESS, LOUDNESS_RATE, reverbObj, ParameterLib< SoundStaticParam, SoundDynamicParam >::setParam(), spatializer_, and START_TIME.

8.41.2.2 Sound::Sound (int *numPartials*, m_value_type *baseFreq*)

This is a constructor that creates a Sound object with proper partials already created.

Parameters:
numPartials The number of partials in the Sound

baseFreq The base freqency of the Sound

Definition at line 54 of file Sound.cpp.

References Collection< Partial >::add(), DETUNE_DIRECTION, DETUNE_FUNDAMENTAL, DETUNE_SPREAD, DETUNE_VELOCITY, DURATION, FREQUENCY, LOUDNESS, LOUDNESS_RATE, m_value_type, PARTIAL_NUM, RELATIVE_AMPLITUDE, reverbObj, ParameterLib< SoundStaticParam, SoundDynamicParam >::setParam(), ParameterLib< PartialStaticParam, PartialDynamicParam >::setParam(), spatializer_, and START_TIME.

8.41.3 Member Function Documentation

8.41.3.1 void Sound::getPartialParamEnv (PartialDynamicParam *p*)

This function iterates through the partials, calling getParam on them.

Parameters:

p The PartialDynamicParam to get the envelope which was used on it

8.41.3.2 m_time_type Sound::getTotalDuration (void)

This returns the total duration of the sound. If there is no reverb on this sound or any of its partials, this will return exactly the value of getParam(DURATION). If any partial has reverb, though, that partial will need extra time for the reverb to die out. Likewise, if the sound has reverb, that will also take time to die out.

Returns:

The total duration of the sound including reverb

Definition at line 363 of file Sound.cpp.

References DURATION, Reverb::getDecay(), ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), Iterator< T >::hasNext(), Collection< Partial >::iterator(), m_time_type, Iterator< T >::next(), and reverbObj.

Referenced by render(), and Score::render().

8.41.3.3 MultiTrack * Sound::render (int *numChannels*, m_rate_type *samplingRate* = DEFAULT_SAMPLING_RATE)

This returns a MultiTrack object of the rendered partial.

Parameters:

numChannels The number of channels in the MultiTrack

samplingRate The sampling rate

Returns:
A MultiTrack

Note:
The object must be deleted by the user calling the function.

Definition at line 149 of file Sound.cpp.

References Loudness::calculate(), Track::composite(), DETUNE_FUNDAMENTAL, DETUNING_ENV, Reverb::do_reverb_Track(), DURATION, ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), getTotalDuration(), Iterator< T >::hasNext(), Collection< Partial >::iterator(), LOUDNESS, LOUDNESS_RATE, m_rate_type, m_sample_count_type, m_time_type, Iterator< T >::next(), reverbObj, setup_detuning_env(), Collection< Partial >::size(), Spatializer::spatialize(), and spatializer_.

Referenced by multithreaded_render_worker(), and Score::render().

8.41.3.4 void Sound::setPartialParam (PartialDynamicParam *p*, m_value_type *v*)

This function iterates through the partials, calling setParam on them.

Parameters:
p The PartialDynamicParam to set
v The value to set the parameter to

Definition at line 109 of file Sound.cpp.

References FREQUENCY, Iterator< T >::hasNext(), Collection< Partial >::iterator(), m_value_type, and Iterator< T >::next().

8.41.3.5 void Sound::setPartialParam (PartialDynamicParam *p*, DynamicVariable & *v*)

This function iterates through the partials, calling setParam on them.

Parameters:
p The PartialDynamicParam to set
v The value to set the parameter to

Todo
Change FREQUENCY param like old MOSS

Definition at line 101 of file Sound.cpp.

References Iterator< T >::hasNext(), Collection< Partial >::iterator(), and Iterator< T >::next().

8.41.3.6 void Sound::setPartialParam (PartialStaticParam *p*, m_value_type *v*)

This function iterates through the partials, calling setParam on them.

Parameters:

p The PartialStaticParam to set

v The value to set the parameter to

Definition at line 93 of file Sound.cpp.

References Iterator< T >::hasNext(), Collection< Partial >::iterator(), m_value_type, and Iterator< T >::next().

8.41.3.7 void Sound::setSpatializer (Spatializer & *s*)

This function sets the Spatializer used for this sound. The given parameter is copied into this object's memory.

Parameters:

s The Spatializer to use

Definition at line 306 of file Sound.cpp.

References Spatializer::clone(), and spatializer_.

8.41.3.8 void Sound::setup_detuning_env (ExponentialInterpolator * *env*) [private]

This function creates a detuning envelope.

Parameters:

env The envelope

Definition at line 319 of file Sound.cpp.

References Interpolator::addEntry(), DETUNE_DIRECTION, DETUNE_SPREAD, DETUNE_VELOCITY, and ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam().

Referenced by render().

8.41.3.9 void Sound::use_reverb (Reverb * *newReverbObj*)

This function performs reverb in the render() method

Parameters:
newReverbObj The Reverb object

Definition at line 313 of file Sound.cpp.

References reverbObj.

Referenced by xml_read().

8.41.3.10 void Sound::xml_print (ofstream & *xmlOutput*, list< Reverb * > & *revObjs*, list< DynamicVariable * > & *dynObjs*)

Deprecated
This outputs an XML representation of the object to STDOUT

Definition at line 382 of file Sound.cpp.

References DETUNE_DIRECTION, DETUNE_FUNDAMENTAL, DETUNE_SPREAD, DETUNE_VELOCITY, DURATION, ParameterLib< SoundStaticParam, SoundDynamicParam >::getParam(), Iterator< T >::hasNext(), Collection< Partial >::iterator(), LOUDNESS, LOUDNESS_RATE, Iterator< T >::next(), reverbObj, spatializer_, START_TIME, and Spatializer::xml_print().

Referenced by Score::xml_print().

8.41.3.11 void Sound::xml_read (XmlReader::xmltag * *soundtag*, hash_map< long, Reverb * > * *reverbHash*, hash_map< long, DynamicVariable * > * *dvHash*)

Deprecated

Definition at line 422 of file Sound.cpp.

References Collection< Partial >::add(), XmlReader::xmltag::children, DETUNE_DIRECTION, DETUNE_FUNDAMENTAL, DETUNE_SPREAD, DETUNE_VELOCITY, DURATION, XmlReader::xmltag::findChildParamValue(), LOUDNESS, LOUDNESS_RATE, XmlReader::xmltag::name, ParameterLib< SoundStaticParam, SoundDynamicParam >::setParam(), START_TIME, use_reverb(), and Partial::xml_read().

Referenced by Score::xml_read().

8.41.4 Member Data Documentation

8.41.4.1 Reverb* Sound::reverbObj [private]

Pointer to a reverb object that will apply reverb to this sound

Definition at line 254 of file Sound.h.

Referenced by getTotalDuration(), render(), Sound(), use_reverb(), and xml_print().

8.41.4.2 Spatializer* Sound::spatializer_ [private]

Definition at line 147 of file Sound.h.

Referenced by render(), setSpatializer(), Sound(), and xml_print().

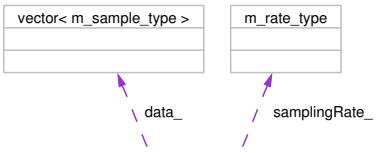
The documentation for this class was generated from the following files:

- Sound.h
- Sound.cpp

8.42 SoundSample Class Reference

`#include <SoundSample.h>`

Collaboration diagram for SoundSample:



Public Member Functions

- SoundSample (m_sample_count_type sampleCount, m_rate_type samplingRate=DEFAULT_SAMPLING_RATE, bool zeroData=false)
- SoundSample (const SoundSample &ss)
- SoundSample & operator= (const SoundSample &ss)
- ~SoundSample ()
- void setSamplingRate (m_rate_type samplingRate)
- m_rate_type getSamplingRate ()
- m_sample_count_type getSampleCount ()
- m_sample_type & operator[ ] (m_sample_count_type index)
- void composite (SoundSample &ss, m_time_type startTime)
- void scale (m_value_type factor)

Private Attributes

- m_rate_type samplingRate_
- vector< m_sample_type > data_

8.42.1 Detailed Description

This is a sample of sound with a fixed number of individual samples. For the sake of speed, this class has no range checking. Be careful when using it as it may cause a bus error.

Author:
Braden Kowitz

Definition at line 46 of file SoundSample.h.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 SoundSample::SoundSample (m_sample_count_type *sampleCount*, m_rate_type *samplingRate* = DEFAULT_SAMPLING_RATE, bool *zeroData* = false)

This is a constructor which takes the number of samples and the sampling rate. It will optionally zero all the data if requested.

Parameters:
sampleCount The number of samples

samplingRate The sampling rate

zeroData Whether to zero the data (default is false)

Definition at line 52 of file SoundSample.cpp.

References data_, m_rate_type, and m_sample_count_type.

8.42.2.2 SoundSample::SoundSample (const SoundSample & *ss*)

This is a copy constructor.

Parameters:
ss The SoundSample to copy

Definition at line 67 of file SoundSample.cpp.

8.42.2.3 SoundSample::~SoundSample ()

This is the destructor.

Definition at line 84 of file SoundSample.cpp.

8.42.3 Member Function Documentation

8.42.3.1 void SoundSample::composite (SoundSample & *ss*, m_time_type *startTime*)

This function composites another SoundSample on top of this object. Both must have the same samplingRate.

Parameters:

ss The SoundSample to composite

startTime When to start applying the SoundSample

Definition at line 116 of file SoundSample.cpp.

References data_, m_sample_count_type, m_time_type, MIN_CLIP_WARNING, and samplingRate_.

Referenced by Track::composite().

8.42.3.2 m_sample_count_type SoundSample::getSampleCount ()

Each SoundSample object is created with a specified number of samples. This function returns that number.

Returns:

The current number of samples

Definition at line 105 of file SoundSample.cpp.

References data_, and m_sample_count_type.

Referenced by Score::channelAnticlip(), Score::channelScale(), Score::clip(), Reverb::constructAmp(), Filter::do_filter_SoundSample(), Reverb::do_reverb_SoundSample(), Score::scale(), Pan::spatialize(), and MultiPan::spatialize().

8.42.3.3 m_rate_type SoundSample::getSamplingRate ()

This function returns the sampling rate for this SoundSample.

Returns:

The current sampling rate

Definition at line 99 of file SoundSample.cpp.

References m_rate_type, and samplingRate_.

Referenced by Reverb::constructAmp(), Filter::do_filter_SoundSample(), Reverb::do_reverb_SoundSample(), Pan::spatialize(), and MultiPan::spatialize().

8.42.3.4 SoundSample & SoundSample::operator= (const SoundSample & *ss*)

This is an overloaded assignment operator.

Parameters:
ss The SoundSample to assign

Definition at line 73 of file SoundSample.cpp.

References data_, and samplingRate_.

8.42.3.5]

m_sample_type & SoundSample::operator[ ] (m_sample_count_type *index*)

This overloaded operator provides element access to this structure. Valid ranges are from 0 to (sampleCount-1). For efficency, this performs no range checking: be careful!

Parameters:
index Which sample to access

Note:
This should be inline, but I'm having problems making it work.

Definition at line 111 of file SoundSample.cpp.

References data_, m_sample_count_type, and m_sample_type.

8.42.3.6 void SoundSample::scale (m_value_type *factor*)

This function scales every value in this SoundSample by a given factor.

Parameters:
factor The scaling factor

Definition at line 162 of file SoundSample.cpp.

References data_, and m_value_type.

Referenced by Track::scale().

8.42.3.7 void SoundSample::setSamplingRate (m_rate_type *samplingRate*)

Each sound sample stores its sampling rate. This is good information to know for composition and writing out to file. This function sets the sampling rate.

Parameters:

samplingRate The new sampling rate

Definition at line 93 of file SoundSample.cpp.

References m_rate_type, and samplingRate_.

8.42.4 Member Data Documentation

8.42.4.1 vector<m_sample_type> SoundSample::data_ [private]

Definition at line 132 of file SoundSample.h.

Referenced by composite(), getSampleCount(), operator=(), operator[ ](), scale(), and SoundSample().

8.42.4.2 m_rate_type SoundSample::samplingRate_ [private]

Definition at line 130 of file SoundSample.h.

Referenced by composite(), getSamplingRate(), operator=(), and setSamplingRate().

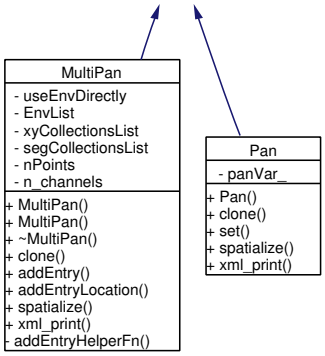
The documentation for this class was generated from the following files:

- SoundSample.h
- SoundSample.cpp

8.43 Spatializer Class Reference

`#include <Spatializer.h>`

Inheritance diagram for Spatializer:



Public Member Functions

- virtual MultiTrack * spatialize (Track &t, int numTracks)
- virtual Spatializer * clone ()
- virtual void xml_print (ofstream &xmlOutput)

8.43.1 Detailed Description

A Spatializer is a definition of an object that will spatalize (move around in space; this is a generalization of panning) a single sound over multiple channels. The default action of this spatializer will evenly distribute the sound over the requested number of channels. Each Spatializer must be dynamic enough to allow spatialization to a number of different channels specified at run-time.

Author:
Braden Kowitz

Definition at line 48 of file Spatializer.h.

8.43.2 Member Function Documentation

8.43.2.1 Spatializer * Spatializer::clone () [virtual]

This function creates an exact duplicate of this Spatializer.

Reimplemented in MultiPan, and Pan.

Definition at line 61 of file Spatializer.cpp.

Referenced by Sound::setSpatializer().

8.43.2.2 MultiTrack * Spatializer::spatialize (Track & *t*, int *numTracks*) [virtual]

This will take a single Track object, and spatialize it to a MultiTrack object with numTracks tracks. The default behevior is to average the sound evenly accross all tracks.

Parameters:

t The Track to spatialize

numTracks The number of Tracks to create

Returns:

a MultiTrack

Reimplemented in MultiPan, and Pan.

Definition at line 37 of file Spatializer.cpp.

References Collection< Track * >::add(), and Track::scale().

Referenced by Sound::render().

8.43.2.3 void Spatializer::xml_print (ofstream & *xmlOutput*) [virtual]

Deprecated

Reimplemented in MultiPan, and Pan.

Definition at line 66 of file Spatializer.cpp.

Referenced by Sound::xml_print().

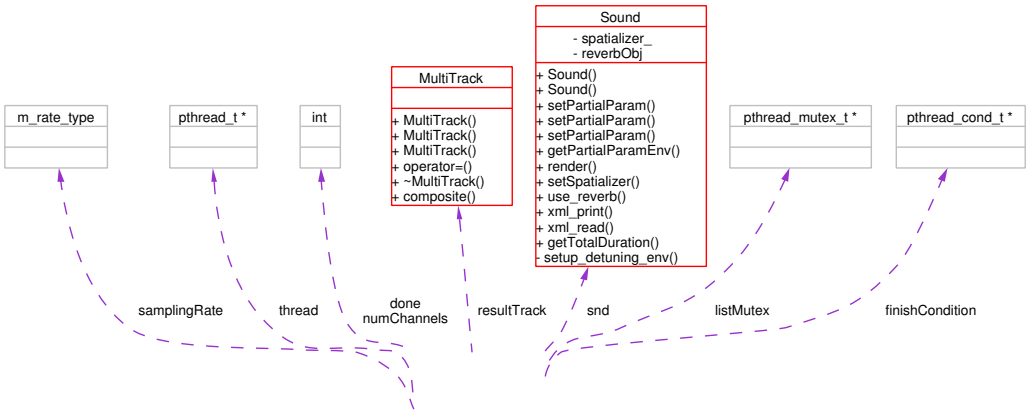
The documentation for this class was generated from the following files:

- Spatializer.h
- Spatializer.cpp

8.44 threadlist_entry Struct Reference

`#include <Score.h>`

Collaboration diagram for threadlist_entry:



Public Attributes

- Sound * snd
- MultiTrack * resultTrack
- int done
- pthread_t * thread
- pthread_mutex_t * listMutex
- pthread_cond_t * finishCondition
- int numChannels
- m_rate_type samplingRate

8.44.1 Member Data Documentation

8.44.1.1 int threadlist_entry::done

Definition at line 50 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.2 pthread_cond_t* threadlist_entry::finishCondition

Definition at line 56 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.3 pthread_mutex_t* threadlist_entry::listMutex

Definition at line 54 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.4 int threadlist_entry::numChannels

Definition at line 59 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.5 MultiTrack* threadlist_entry::resultTrack

Definition at line 48 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.6 m_rate_type threadlist_entry::samplingRate

Definition at line 60 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.7 Sound* threadlist_entry::snd

Definition at line 46 of file Score.h.

Referenced by multithreaded_render_worker(), and Score::render().

8.44.1.8 pthread_t* threadlist_entry::thread

Definition at line 52 of file Score.h.

Referenced by Score::render().

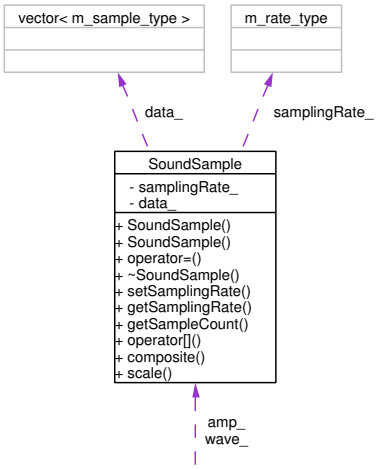
The documentation for this struct was generated from the following file:

- Score.h

8.45 Track Class Reference

`#include <Track.h>`

Collaboration diagram for Track:



Public Member Functions

- Track (SoundSample *wave, SoundSample *amp=0)
- Track (m_sample_count_type sampleCount, m_rate_type samplingRate=DEFAULT_SAMPLING_RATE, bool zeroData=false)
- Track (const Track &t)
- Track & operator= (const Track &t)
- ~Track ()
- SoundSample & getWave ()
- bool hasAmp ()
- SoundSample & getAmp ()
- void composite (Track &t, m_time_type startTime=0.0)
- void scale (m_value_type factor)

Private Attributes

- SoundSample * wave_
- SoundSample * amp_

8.45.1 Detailed Description

A track contains a SoundSample object with sound and an optional SoundSample member that contains Amplitude data. When the track object is deleted, it also deletes the wave and amp objects.

Author:
Braden Kowitz

Definition at line 42 of file Track.h.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 Track::Track (SoundSample * *wave*, SoundSample * *amp* = 0)

This is a constructor that creates a new Track object. The passed in wave and amp objects will be deleted when the Track object is deleted. Take care.

Parameters:
wave The SoundSample that represents the wave

amp The SoundSample that represents the amplitude

Definition at line 37 of file Track.cpp.

8.45.2.2 Track::Track (m_sample_count_type *sampleCount*, m_rate_type *samplingRate* = DEFAULT_SAMPLING_RATE, bool *zeroData* = false)

This is a constructor that creates a empty track with a set number of samples. It optionally sets a preferred sample rate, and zeros out the data if needed.

Parameters:
sampleCount The number of samples to create

samplingRate The sampling rate

zeroData Whether to zero the data (default is false)

Definition at line 43 of file Track.cpp.

References m_rate_type, and m_sample_count_type.

8.45.2.3 Track::Track (const Track & *t*)

This is a copy constructor.

Parameters:

t The track to copy

Definition at line 52 of file Track.cpp.

References amp_, and wave_.

8.45.2.4 Track::~Track ()

This is the destructor.

Definition at line 84 of file Track.cpp.

References amp_, and wave_.

8.45.3 Member Function Documentation

8.45.3.1 void Track::composite (Track & *t*, m_time_type *startTime* = 0.0)

This function composites another wave on top of this wave. It also composites the amp values if both tracks contain amp. Specifing startTime (seconds) will shift the passed in track before composition.

Parameters:

t The Track to composite

startTime the time offset to start compositing (default is 0.0)

Definition at line 120 of file Track.cpp.

References amp_, SoundSample::composite(), hasAmp(), m_time_type, and wave_.

Referenced by Sound::render().

8.45.3.2 SoundSample & Track::getAmp ()

This function returns the amplitude SoundSample for this track. If this object does not have an amplitude track the function will return a new SoundSample object with zero entries.

Returns:

The SoundSample that represents the amplitude

Note:
Always call hasAmp() first!

Todo
This causes a memory leak!

Definition at line 106 of file Track.cpp.

References amp_.

Referenced by Score::anticlip(), Score::channelAnticlip(), Score::channelScale(), Score::clip(), Score::scale(), Pan::spatialize(), and MultiPan::spatialize().

8.45.3.3 SoundSample & Track::getWave ()

This function returns a reference to the wave object in this track.

Returns:
A SoundSample that describes the wave

Definition at line 94 of file Track.cpp.

References wave_.

Referenced by Score::anticlip(), Score::channelAnticlip(), Score::channelScale(), Score::clip(), Filter::do_filter_Track(), Reverb::do_reverb_MultiTrack(), Reverb::do_reverb_Track(), Score::scale(), Pan::spatialize(), MultiPan::spatialize(), and AuWriter::write().

8.45.3.4 bool Track::hasAmp ()

This function indicates whether this track has an amplitude SoundSample.

Return values:
true There is an amplitude SoundSample

false There is not an amplitue SoundSample

Definition at line 100 of file Track.cpp.

References amp_.

Referenced by composite().

8.45.3.5 Track & Track::operator= (const Track & *t*)

This is an overloaded assignment operator.

Parameters:
t The Track to assign

Definition at line 66 of file Track.cpp.

References amp_, and wave_.

8.45.3.6 void Track::scale (m_value_type *factor*)

This function scales an entire track by a factor.

Parameters:
factor The scaling factor

Definition at line 130 of file Track.cpp.

References amp_, m_value_type, SoundSample::scale(), and wave_.

Referenced by Spatializer::spatialize().

8.45.4 Member Data Documentation

8.45.4.1 SoundSample* Track::amp_ [private]

Definition at line 133 of file Track.h.

Referenced by composite(), getAmp(), hasAmp(), operator=(), scale(), Track(), and ~Track().

8.45.4.2 SoundSample* Track::wave_ [private]

Definition at line 132 of file Track.h.

Referenced by composite(), getWave(), operator=(), scale(), Track(), and ~Track().

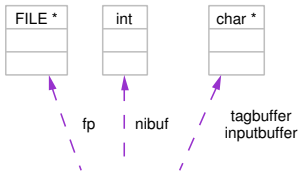
The documentation for this class was generated from the following files:

- Track.h
- Track.cpp

8.46 XmlReader Class Reference

`#include <XmlReader.h>`

Collaboration diagram for XmlReader:



Public Member Functions

- XmlReader ()
- ~XmlReader ()
- bool openFile (char *file)
- bool closeFile ()
- bool readTag (xmltag *tag)
- xmltagset * readXMLDocument ()
- void readXMLDocument (xmltag *)

Protected Member Functions

- bool fillTagBuffer ()
- void dewhitespace (char *c)

Protected Attributes

- FILE * fp
- char * inputbuffer
- char * tagbuffer
- int nibuf

8.46.1 Constructor & Destructor Documentation

8.46.1.1 XmlReader::XmlReader ()

Definition at line 237 of file XmlReader.cpp.

References fp, inputbuffer, nibuf, tagbuffer, and XML_BUFFER_SIZE.

8.46.1.2 XmlReader::~XmlReader ()

Definition at line 248 of file XmlReader.cpp.

References closeFile(), inputbuffer, and tagbuffer.

8.46.2 Member Function Documentation

8.46.2.1 bool XmlReader::closeFile ()

Closes the XML file

Definition at line 270 of file XmlReader.cpp.

References fp.

Referenced by ~XmlReader().

8.46.2.2 void XmlReader::dewhitespace (char * *c*) [protected]

Internal function to remove excess whitespace

Definition at line 283 of file XmlReader.cpp.

Referenced by fillTagBuffer().

8.46.2.3 bool XmlReader::fillTagBuffer () [protected]

Called by readTag, this fill acutally get the tag out of the file

Definition at line 313 of file XmlReader.cpp.

References dewhitespace(), fp, inputbuffer, nibuf, tagbuffer, and XML_BUFFER_SIZE.

Referenced by readTag().

8.46.2.4 bool XmlReader::openFile (char * *file*)

Opens an XML file for reading

Definition at line 257 of file XmlReader.cpp.

References fp.

8.46.2.5 bool XmlReader::readTag (xmltag * *tag*)

The guts of the object, this gets the next tag from the file.

Definition at line 392 of file XmlReader.cpp.

References XmlReader::xmltag::destroyTag(), fillTagBuffer(), XmlReader::xmltag::isClosing, XmlReader::tagparam::next, XmlReader::xmltag::params, XmlReader::xmltag::setName(), and tagbuffer.

Referenced by readXMLDocument(), and DynamicVariableSequence::xml_read().

8.46.2.6 void XmlReader::readXMLDocument (xmltag *)

Definition at line 484 of file XmlReader.cpp.

References XmlReader::xmltagset::add(), XmlReader::xmltag::children, XmlReader::xmltag::isClosing, XmlReader::xmltag::name, readTag(), and readXMLDocument().

8.46.2.7 XmlReader::xmltagset * XmlReader::readXMLDocument ()

Definition at line 473 of file XmlReader.cpp.

References XmlReader::xmltag::children, and XmlReader::xmltag::name.

Referenced by readXMLDocument().

8.46.3 Member Data Documentation

8.46.3.1 FILE* XmlReader::fp [protected]

Definition at line 178 of file XmlReader.h.

Referenced by closeFile(), fillTagBuffer(), openFile(), and XmlReader().

8.46.3.2 char* XmlReader::inputbuffer [protected]

Definition at line 179 of file XmlReader.h.

Referenced by fillTagBuffer(), XmlReader(), and ~XmlReader().

8.46.3.3 int XmlReader::nibuf [protected]

Definition at line 181 of file XmlReader.h.

Referenced by fillTagBuffer(), and XmlReader().

8.46.3.4 char* XmlReader::tagbuffer [protected]

Definition at line 180 of file XmlReader.h.

Referenced by fillTagBuffer(), readTag(), XmlReader(), and ~XmlReader().

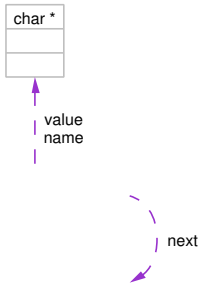
The documentation for this class was generated from the following files:

- XmlReader.h
- XmlReader.cpp

8.47 XmlReader::tagparam Class Reference

`#include <XmlReader.h>`

Collaboration diagram for XmlReader::tagparam:



Public Member Functions

- tagparam (char *n, char *v)
- ~tagparam ()

Public Attributes

- char * name
- char * value
- tagparam * next

Friends

- class XmlReader

8.47.1 Detailed Description

tagparam class A linked list of tag parameters I suppose to be more "lasslike" this should probably just be a Collection.

Definition at line 55 of file XmlReader.h.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 XmlReader::tagparam::tagparam (char * *n*, char * *v*)

Initializes object with set values

Definition at line 33 of file XmlReader.cpp.

References name, next, and value.

8.47.2.2 XmlReader::tagparam::~tagparam ()

Definition at line 49 of file XmlReader.cpp.

References name, next, and value.

8.47.3 Friends And Related Function Documentation

8.47.3.1 friend class XmlReader [friend]

Definition at line 57 of file XmlReader.h.

8.47.4 Member Data Documentation

8.47.4.1 char* XmlReader::tagparam::name

Definition at line 63 of file XmlReader.h.

Referenced by XmlReader::xmltag::findParam(), tagparam(), and ~tagparam().

8.47.4.2 tagparam* XmlReader::tagparam::next

Definition at line 66 of file XmlReader.h.

Referenced by XmlReader::xmltag::destroyTag(), XmlReader::xmltag::findParam(), XmlReader::readTag(), tagparam(), and ~tagparam().

8.47.4.3 char* XmlReader::tagparam::value

Definition at line 64 of file XmlReader.h.

Referenced by XmlReader::xmltag::getParamValue(), tagparam(), and ~tagparam().

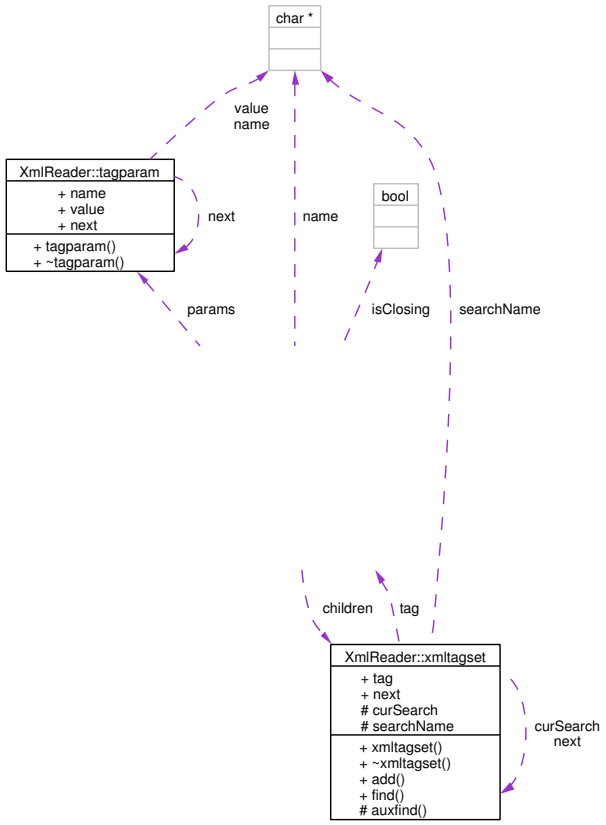
The documentation for this class was generated from the following files:

- XmlReader.h
- XmlReader.cpp

8.48 XmlReader::xmltag Class Reference

`#include <XmlReader.h>`

Collaboration diagram for XmlReader::xmltag:



Public Member Functions

- xmltag ()
- ~xmltag ()
- void setName (char *in)
- char * getParamValue (char *pname)
- bool isParamDefined (char *pname)
- char * findChildParamValue (char *childName, char *paramName)

Public Attributes

- char * name
- tagparam * params
- bool isClosing
- xmltagset * children

Protected Member Functions

- tagparam * findParam (char *pname)
- void destroyTag ()

Friends

- class XmlReader

8.48.1 Detailed Description

xmltag class This is the object generated by the xml reader to represent tags that are read in from the file.

Definition at line 76 of file XmlReader.h.

8.48.2 Constructor & Destructor Documentation

8.48.2.1 XmlReader::xmltag::xmltag ()

Definition at line 70 of file XmlReader.cpp.

References children, isClosing, name, and params.

8.48.2.2 XmlReader::xmltag::~xmltag ()

Definition at line 79 of file XmlReader.cpp.

References destroyTag().

8.48.3 Member Function Documentation

8.48.3.1 void XmlReader::xmltag::destroyTag () [protected]

This frees memeory and clears values inside the tag object

Definition at line 119 of file XmlReader.cpp.

References children, isClosing, name, XmlReader::tagparam::next, and params.

Referenced by XmlReader::readTag(), and ~xmltag().

8.48.3.2 char * XmlReader::xmltag::findChildParamValue (char * *childName*, char * *paramName*)

first does a find for an immediate child tag of given name, then searches that tag for a parameter value. Repeatable if no other find calls are made on children list of this tag in between.

Definition at line 143 of file XmlReader.cpp.

References children, XmlReader::xmltagset::find(), and getParamValue().

Referenced by Partial::auxLoadParam(), DynamicVariable::create_dv_from_xml(), Sound::xml_read(), Score::xml_read(), Reverb::xml_read(), Partial::xml_read(), LPCombFilter::xml_read(), Interpolator::xml_read(), Envelope::xml_read(), Constant::xml_read(), and AllPassFilter::xml_read().

8.48.3.3 XmlReader::tagparam * XmlReader::xmltag::findParam (char * *pname*) [protected]

Definition at line 85 of file XmlReader.cpp.

References XmlReader::tagparam::name, XmlReader::tagparam::next, and params.

Referenced by getParamValue(), and isParamDefined().

8.48.3.4 char * XmlReader::xmltag::getParamValue (char * *pname*)

gets the value of a parameter by name returns NULL if not defined.

Definition at line 99 of file XmlReader.cpp.

References findParam(), and XmlReader::tagparam::value.

Referenced by findChildParamValue(), Score::xml_read(), Interpolator::xml_read(), Envelope::xml_read(), and DynamicVariableSequence::xml_read().

8.48.3.5 bool XmlReader::xmltag::isParamDefined (char * *pname*)

checks to see if a certian parameter is defined in the XML.

Definition at line 109 of file XmlReader.cpp.

References findParam().

8.48.3.6 void XmlReader::xmltag::setName (char * *in*)

Sets the name of the tag

Definition at line 63 of file XmlReader.cpp.

References name.

Referenced by XmlReader::readTag().

8.48.4 Friends And Related Function Documentation

8.48.4.1 friend class XmlReader [friend]

Definition at line 78 of file XmlReader.h.

8.48.5 Member Data Documentation

8.48.5.1 xmltagset* XmlReader::xmltag::children

Definition at line 86 of file XmlReader.h.

Referenced by destroyTag(), findChildParamValue(), XmlReader::readXMLDocument(), Sound::xml_read(), Score::xml_read(), Reverb::xml_read(), Partial::xml_read(), Interpolator::xml_read(), Envelope::xml_read(), and xmltag().

8.48.5.2 bool XmlReader::xmltag::isClosing

Definition at line 82 of file XmlReader.h.

Referenced by destroyTag(), XmlReader::readTag(), XmlReader::readXMLDocument(), DynamicVariableSequence::xml_read(), and xmltag().

8.48.5.3 char* XmlReader::xmltag::name

Definition at line 80 of file XmlReader.h.

Referenced by XmlReader::xmltagset::auxfind(), destroyTag(), XmlReader::readXMLDocument(), setName(), Sound::xml_read(), Score::xml_read(), Reverb::xml_read(), Partial::xml_read(), DynamicVariableSequence::xml_read(), and xmltag().

8.48.5.4 tagparam* XmlReader::xmltag::params

Definition at line 81 of file XmlReader.h.

Referenced by destroyTag(), findParam(), XmlReader::readTag(), and xmltag().

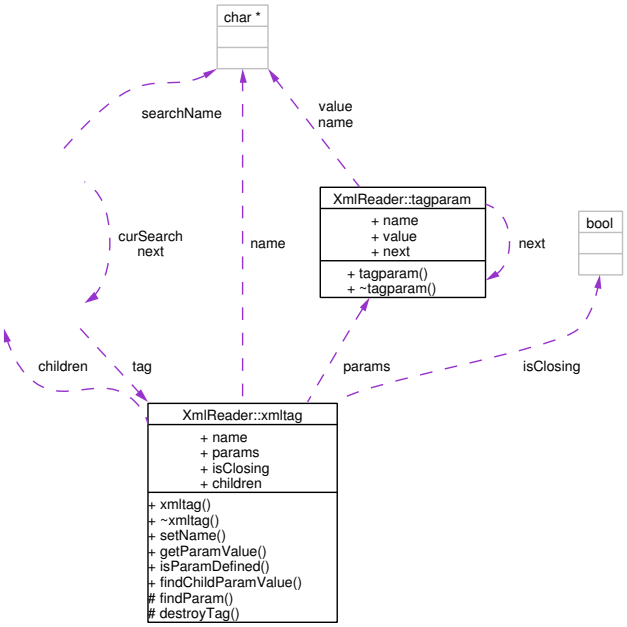
The documentation for this class was generated from the following files:

- XmlReader.h
- XmlReader.cpp

8.49 XmlReader::xmltagset Class Reference

`#include <XmlReader.h>`

Collaboration diagram for XmlReader::xmltagset:



Public Member Functions

- xmltagset ()
- ~xmltagset ()
- void add (xmltag *itag)
- xmltag * find (char *name)

Public Attributes

- xmltag * tag
- xmltagset * next

Protected Member Functions

- xmltag * auxfind (xmltagset *set, char *name)

Protected Attributes

- xmltagset * curSearch
- char * searchName

Friends

- class XmlReader

8.49.1 Constructor & Destructor Documentation

8.49.1.1 XmlReader::xmltagset::xmltagset ()

Definition at line 153 of file XmlReader.cpp.

References curSearch, next, searchName, and tag.

8.49.1.2 XmlReader::xmltagset::~xmltagset ()

Definition at line 162 of file XmlReader.cpp.

References next, and tag.

8.49.2 Member Function Documentation

8.49.2.1 void XmlReader::xmltagset::add (xmltag * *itag*)

Definition at line 174 of file XmlReader.cpp.

References next, and tag.

Referenced by XmlReader::readXMLDocument().

8.49.2.2 XmlReader::xmltag * XmlReader::xmltagset::auxfind (xmltagset * *set*, char * *name*) [protected]

Definition at line 220 of file XmlReader.cpp.

References curSearch, XmlReader::xmltag::name, next, and tag.

Referenced by find().

8.49.2.3 XmlReader::xmltag * XmlReader::xmltagset::find (char * *name*)

Definition at line 190 of file XmlReader.cpp.

References auxfind(), curSearch, searchName, and tag.

Referenced by XmlReader::xmltag::findChildParamValue().

8.49.3 Friends And Related Function Documentation

8.49.3.1 friend class XmlReader [friend]

Definition at line 127 of file XmlReader.h.

8.49.4 Member Data Documentation

8.49.4.1 xmltagset* XmlReader::xmltagset::curSearch [protected]

Definition at line 137 of file XmlReader.h.

Referenced by auxfind(), find(), and xmltagset().

8.49.4.2 xmltagset* XmlReader::xmltagset::next

Definition at line 132 of file XmlReader.h.

Referenced by add(), auxfind(), Partial::xml_read(), xmltagset(), and ~xmltagset().

8.49.4.3 char* XmlReader::xmltagset::searchName [protected]

Definition at line 138 of file XmlReader.h.

Referenced by find(), and xmltagset().

8.49.4.4 xmltag* XmlReader::xmltagset::tag

Definition at line 131 of file XmlReader.h.

Referenced by add(), auxfind(), find(), Partial::xml_read(), xmltagset(), and ~xmltagset().

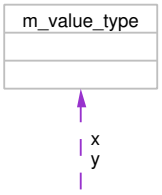
The documentation for this class was generated from the following files:

- XmlReader.h
- XmlReader.cpp

8.50 xy_point Struct Reference

`#include <Types.h>`

Collaboration diagram for xy_point:



Public Attributes

- m_value_type x

  x value

- m_value_type y

  y value

8.50.1 Detailed Description

This is used to make it easier to specify a single point for a Envelope.

Definition at line 72 of file Types.h.

8.50.2 Member Data Documentation

8.50.2.1 m_value_type xy_point::x

x value

Definition at line 75 of file Types.h.

Referenced by MultiPan::addEntryHelperFn(), Envelope::addPoint(), DynamicVariableSequence::AddToShape(), Envelope::Envelope(), Envelope::getPoint(),

Envelope::getPoints(), EnvelopeLibrary::loadLibrary(), DynamicVariableSequence::Print(), DynamicVariableSequence::scale(), Envelope::setPoint(), and DynamicVariableSequence::xml_print().

8.50.2.2 m_value_type xy_point::y

y value

Definition at line 77 of file Types.h.

Referenced by MultiPan::addEntryHelperFn(), DynamicVariableSequence::addInterpolators(), Envelope::addPoint(), Envelope::Envelope(), DynamicVariableSequence::getMaxValue(), Envelope::getPoint(), Envelope::getPoints(), Envelope::getValue(), DynamicVariableSequence::getValue(), EnvelopeLibrary::loadLibrary(), DynamicVariableSequence::Print(), DynamicVariableSequence::scale(), Envelope::setPoint(), and DynamicVariableSequence::xml_print().

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

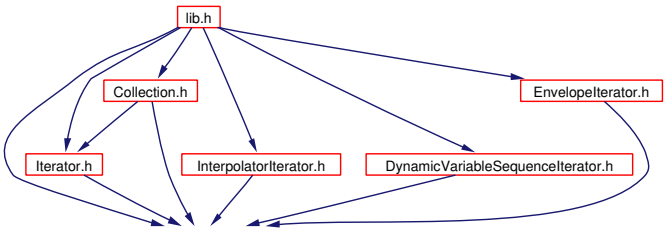
- Types.h

Chapter 9

LASS File Documentation

9.1 AbstractIterator.h File Reference

This graph shows which files directly or indirectly include this file:



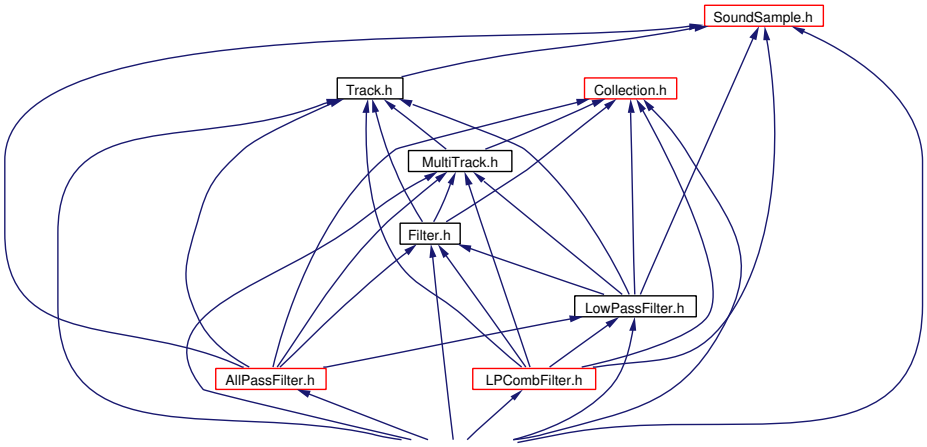
Classes

- class AbstractIterator

9.2 AllPassFilter.cpp File Reference

```
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "Filter.h"
#include "LPCombFilter.h"
#include "LowPassFilter.h"
#include "AllPassFilter.h"
```

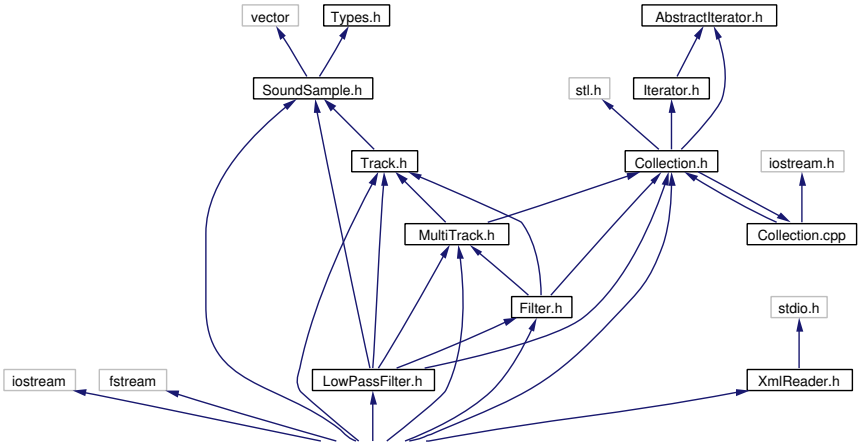
Include dependency graph for AllPassFilter.cpp:



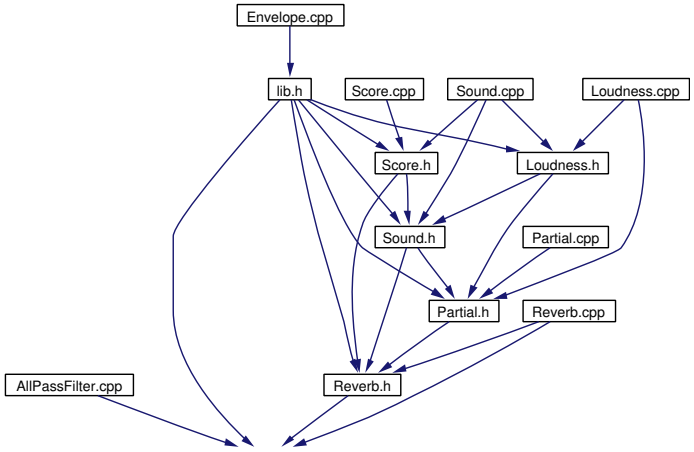
9.3 AllPassFilter.h File Reference

```
#include <iostream>
#include <fstream>
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "LowPassFilter.h"
#include "Filter.h"
#include "XmlReader.h"
```

Include dependency graph for AllPassFilter.h:



This graph shows which files directly or indirectly include this file:



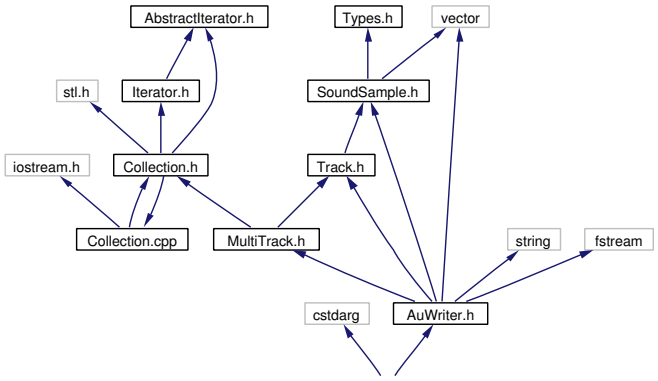
Classes

- class AllPassFilter

9.4 AuWriter.cpp File Reference

```
#include <cstdarg>
#include "AuWriter.h"
```

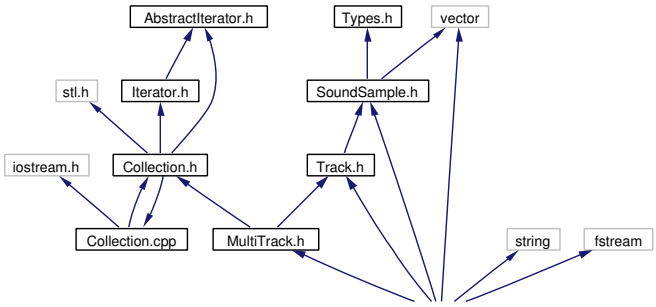
Include dependency graph for AuWriter.cpp:



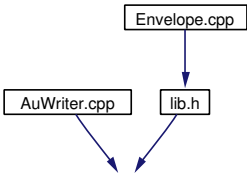
9.5 AuWriter.h File Reference

```
#include "SoundSample.h"
#include "Track.h"
#include "MultiTrack.h"
#include <string>
#include <vector>
#include <fstream>
```

Include dependency graph for AuWriter.h:



This graph shows which files directly or indirectly include this file:



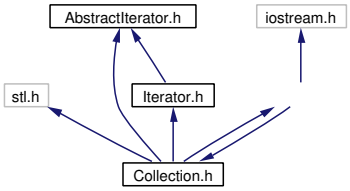
Classes

- class AuWriter

9.6 Collection.cpp File Reference

```
#include <iostream.h>
#include "Collection.h"
```

Include dependency graph for Collection.cpp:

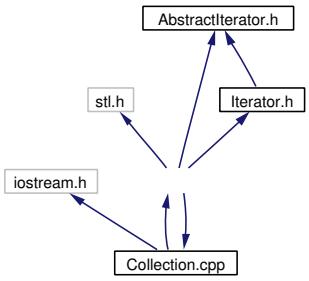


This graph shows which files directly or indirectly include this file:

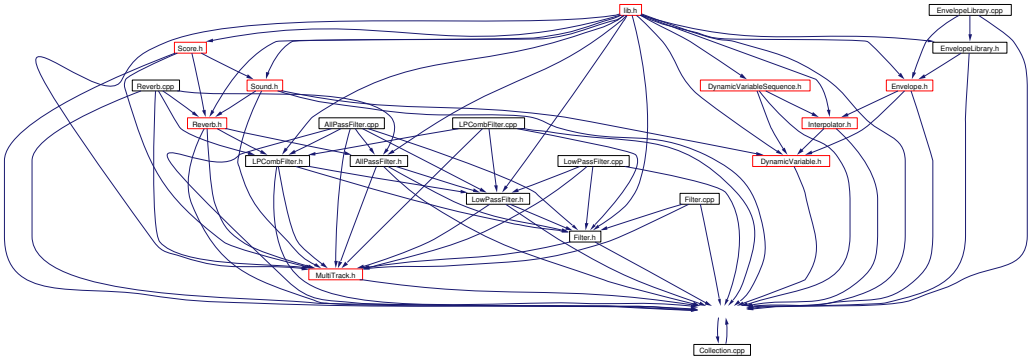
9.7 Collection.h File Reference

```
#include <stl.h>
#include "Iterator.h"
#include "AbstractIterator.h"
#include "Collection.cpp"
```

Include dependency graph for Collection.h:



This graph shows which files directly or indirectly include this file:



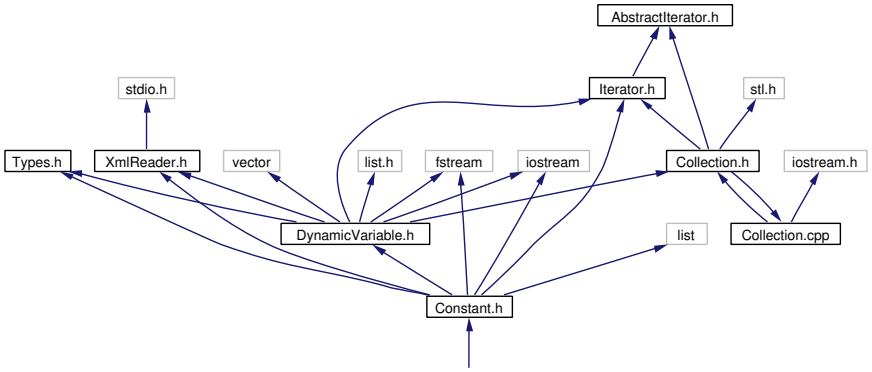
Classes

- class Collection
- class Collection::CollectionIterator

9.8 Constant.cpp File Reference

```
#include "Constant.h"
```

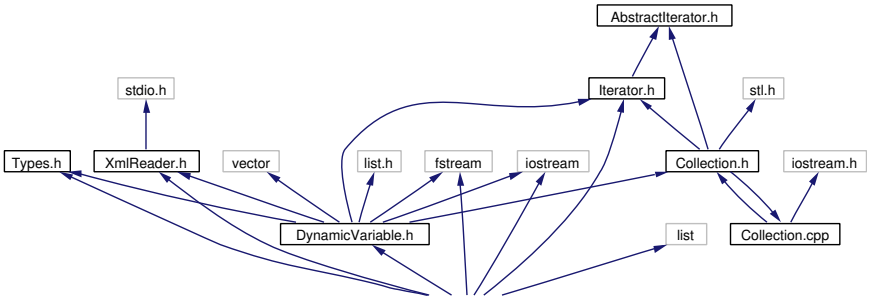
Include dependency graph for Constant.cpp:



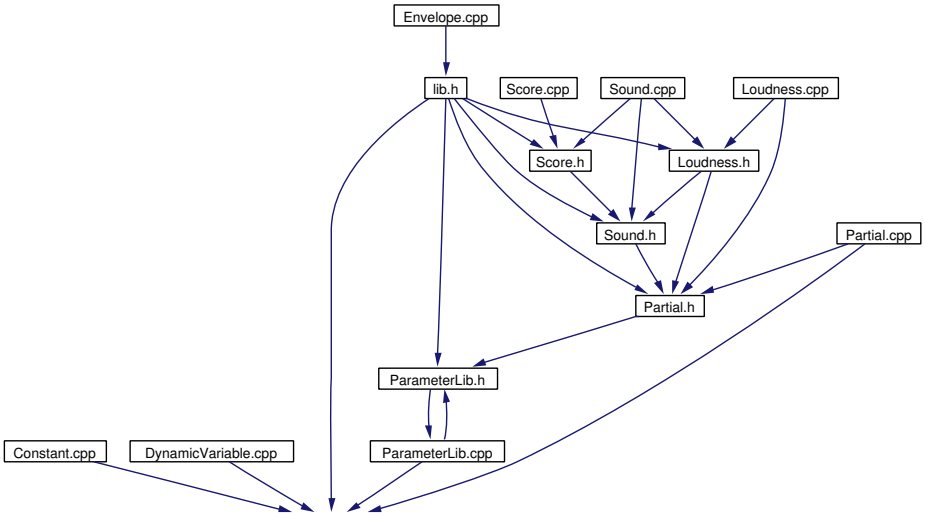
9.9 Constant.h File Reference

```
#include <fstream>
#include <iostream>
#include <list>
#include "Types.h"
#include "DynamicVariable.h"
#include "Iterator.h"
#include "XmlReader.h"
```

Include dependency graph for Constant.h:



This graph shows which files directly or indirectly include this file:



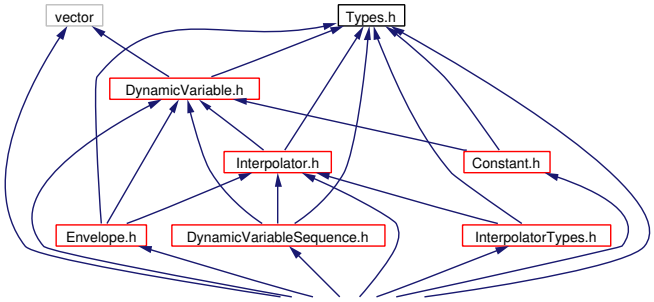
Classes

- class Constant
- class Constant::ConstantIterator

9.10 DynamicVariable.cpp File Reference

```
#include "DynamicVariable.h"
#include "Types.h"
#include <vector>
#include "InterpolatorTypes.h"
#include "Constant.h"
#include "DynamicVariableSequence.h"
#include "Interpolator.h"
#include "Envelope.h"
```

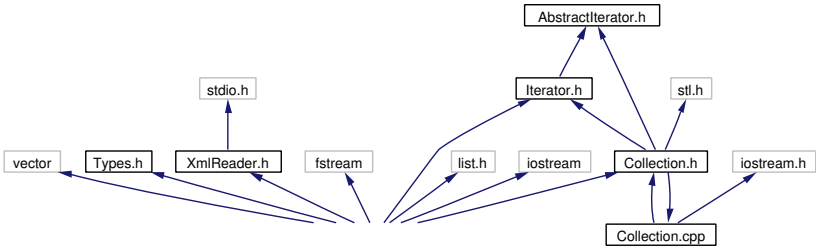
Include dependency graph for DynamicVariable.cpp:



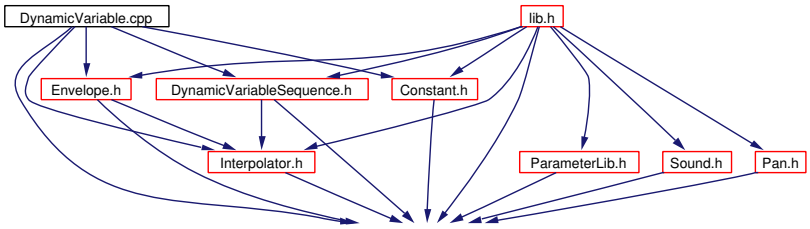
9.11 DynamicVariable.h File Reference

```
#include <fstream>
#include <iostream>
#include <vector>
#include "Types.h"
#include "Iterator.h"
#include "XmlReader.h"
#include "Collection.h"
#include <list.h>
```

Include dependency graph for DynamicVariable.h:



This graph shows which files directly or indirectly include this file:



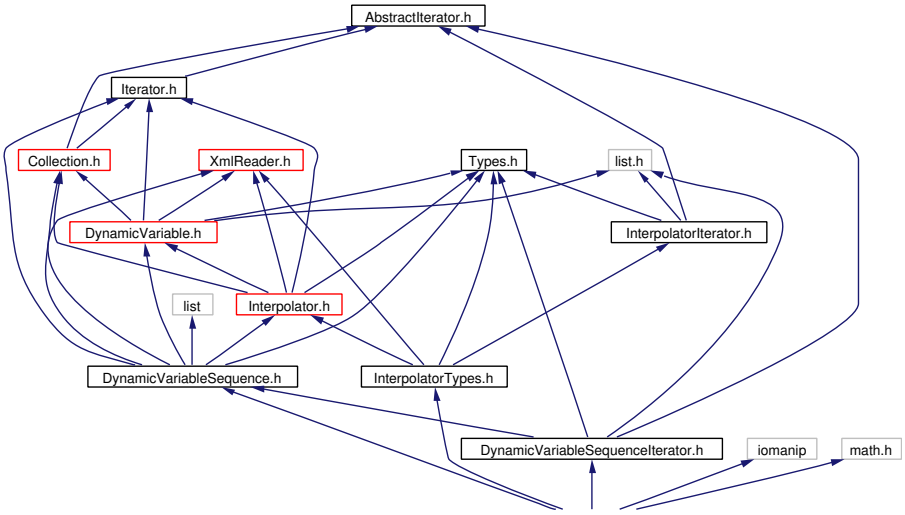
Classes

- class DynamicVariable

9.12 DynamicVariableSequence.cpp File Reference

```
#include "DynamicVariableSequence.h"
#include "DynamicVariableSequenceIterator.h"
#include "InterpolatorTypes.h"
#include <iomanip>
#include <math.h>
```

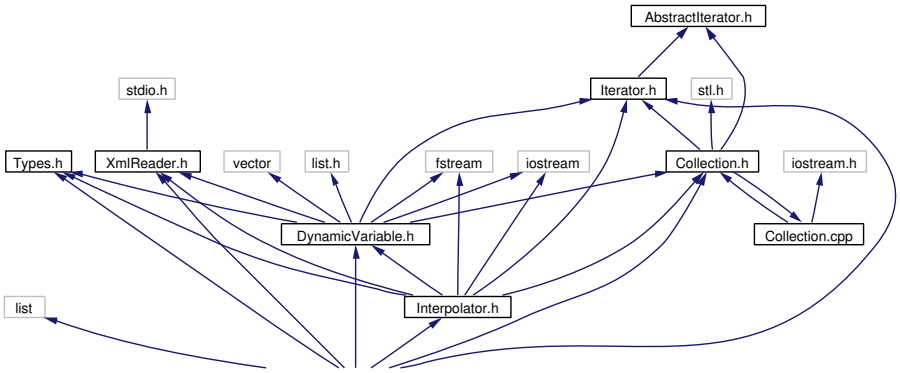
Include dependency graph for DynamicVariableSequence.cpp:



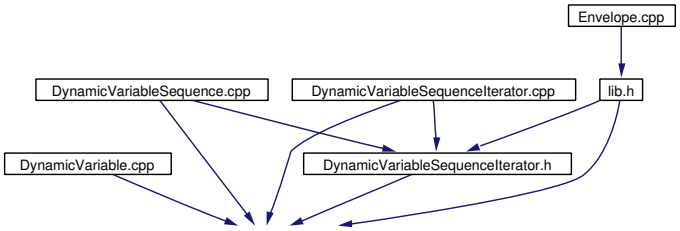
9.13 DynamicVariableSequence.h File Reference

```
#include <list>
#include "Types.h"
#include "Interpolator.h"
#include "Iterator.h"
#include "Collection.h"
#include "DynamicVariable.h"
#include "XmlReader.h"
```

Include dependency graph for DynamicVariableSequence.h:



This graph shows which files directly or indirectly include this file:



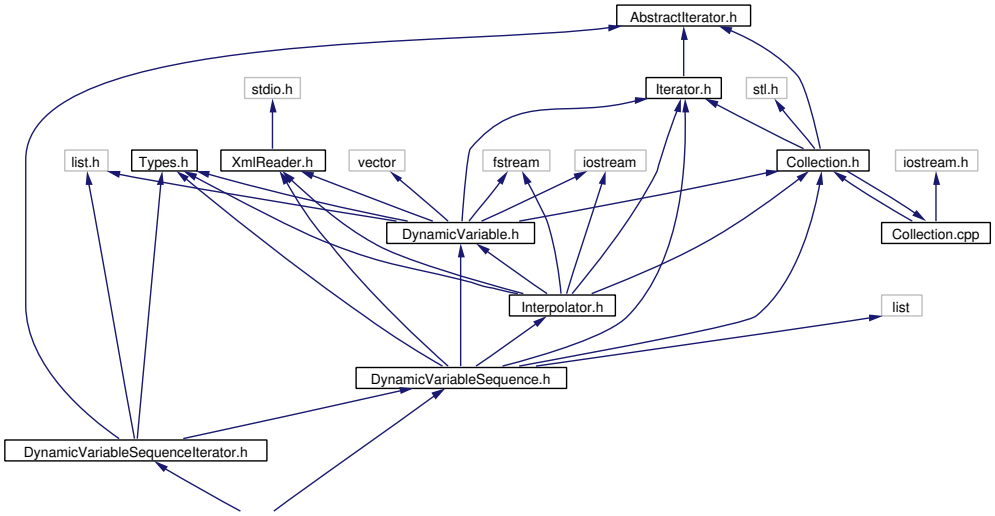
Classes

- class DynamicVariableSequence

9.14 DynamicVariableSequenceIterator.cpp File Reference

```
#include "DynamicVariableSequenceIterator.h"
#include "DynamicVariableSequence.h"
```

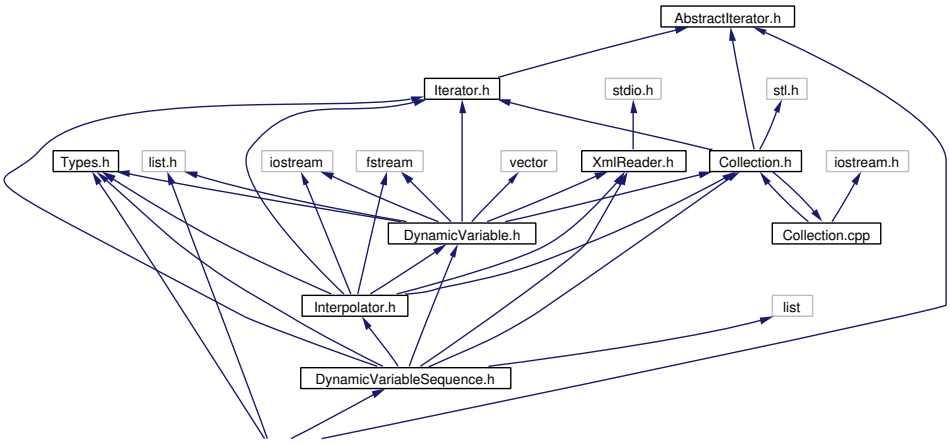
Include dependency graph for DynamicVariableSequenceIterator.cpp:



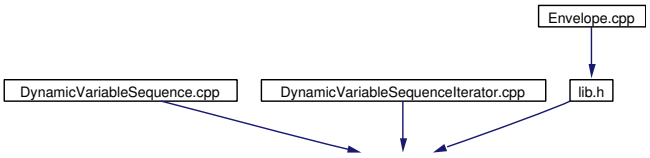
9.15 DynamicVariableSequenceIterator.h File Reference

```
#include "Types.h"
#include "AbstractIterator.h"
#include <list.h>
#include "DynamicVariableSequence.h"
```

Include dependency graph for DynamicVariableSequenceIterator.h:



This graph shows which files directly or indirectly include this file:



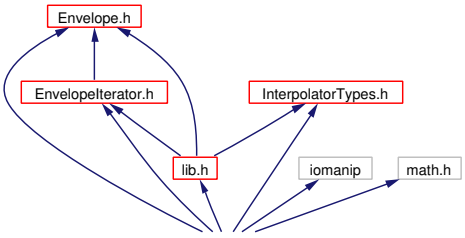
Classes

- class DynamicVariableSequenceIterator

9.16 Envelope.cpp File Reference

```
#include "Envelope.h"
#include "EnvelopeIterator.h"
#include "InterpolatorTypes.h"
#include "lib.h"
#include <iomanip>
#include <math.h>
```

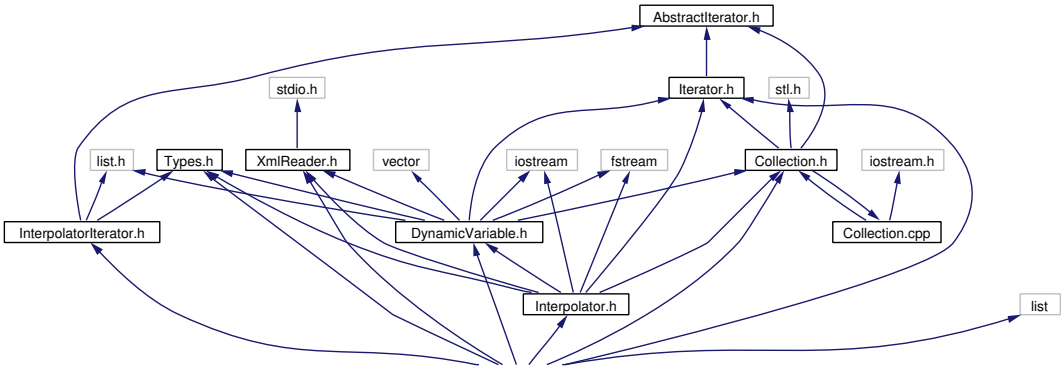
Include dependency graph for Envelope.cpp:



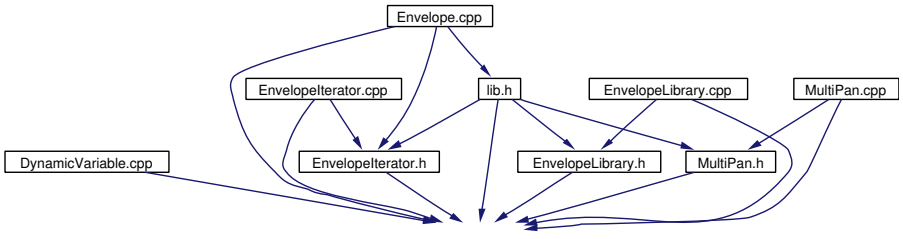
9.17 Envelope.h File Reference

```
#include "XmlReader.h"
#include <list>
#include "Types.h"
#include "Interpolator.h"
#include "Iterator.h"
#include "Collection.h"
#include "DynamicVariable.h"
#include "InterpolatorIterator.h"
```

Include dependency graph for Envelope.h:



This graph shows which files directly or indirectly include this file:



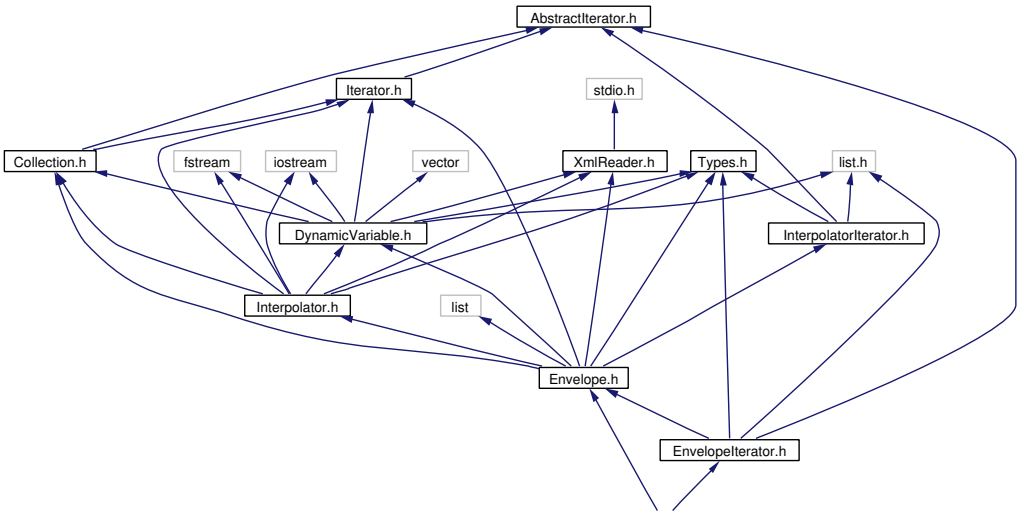
Classes

- class Envelope

9.18 EnvelopeIterator.cpp File Reference

```
#include "EnvelopeIterator.h"
#include "Envelope.h"
```

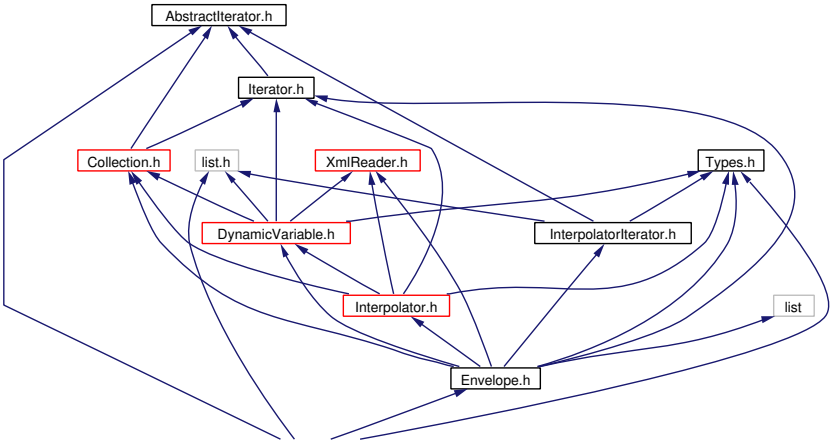
Include dependency graph for EnvelopeIterator.cpp:



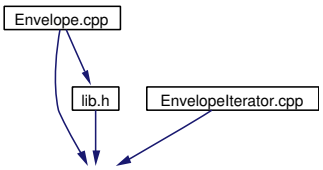
9.19 EnvelopeIterator.h File Reference

```
#include "Types.h"
#include "AbstractIterator.h"
#include <list.h>
#include "Envelope.h"
```

Include dependency graph for EnvelopeIterator.h:



This graph shows which files directly or indirectly include this file:



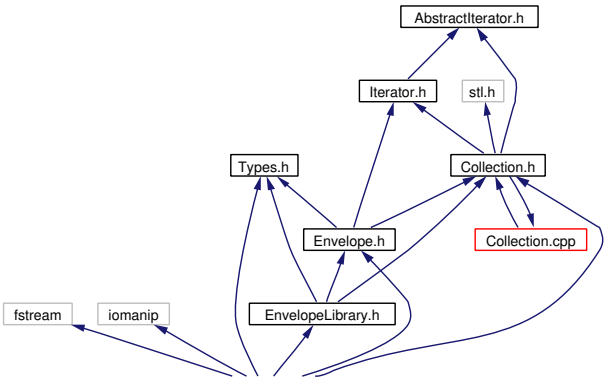
Classes

- class EnvelopeIterator

9.20 EnvelopeLibrary.cpp File Reference

```
#include <fstream>
#include <iomanip>
#include "Types.h"
#include "Collection.h"
#include "EnvelopeLibrary.h"
#include "Envelope.h"
```

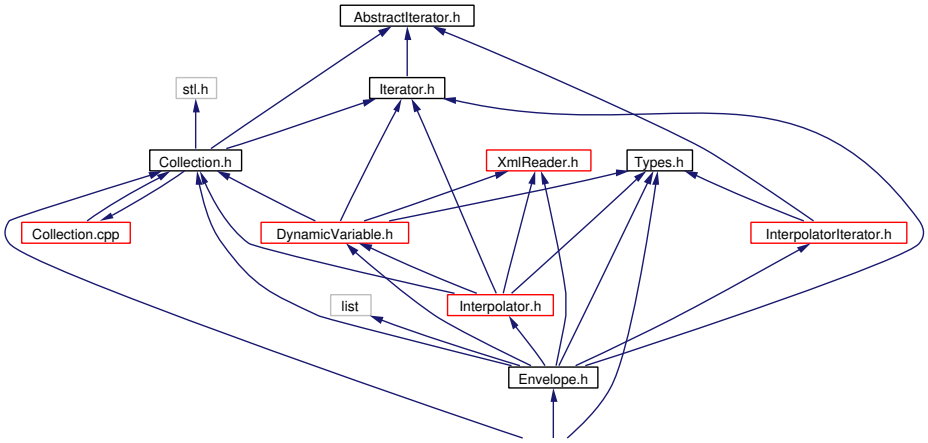
Include dependency graph for EnvelopeLibrary.cpp:



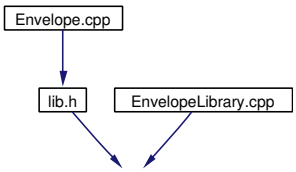
9.21 EnvelopeLibrary.h File Reference

```
#include "Types.h"
#include "Collection.h"
#include "Envelope.h"
```

Include dependency graph for EnvelopeLibrary.h:



This graph shows which files directly or indirectly include this file:



Classes

- class EnvelopeLibrary

9.22 extra-docs.txt File Reference

9.22.1 Detailed Description

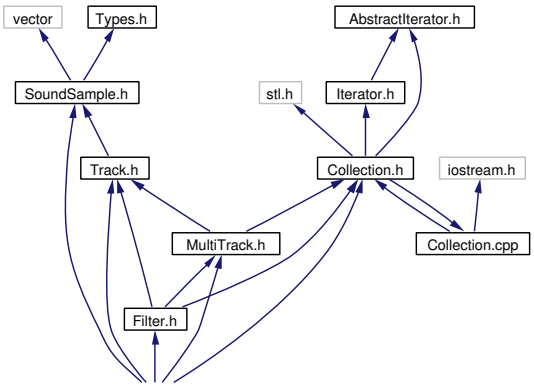
Contains some extra documentation to parse into doxygen.

Definition in file extra-docs.txt.

9.23 Filter.cpp File Reference

```
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "Filter.h"
```

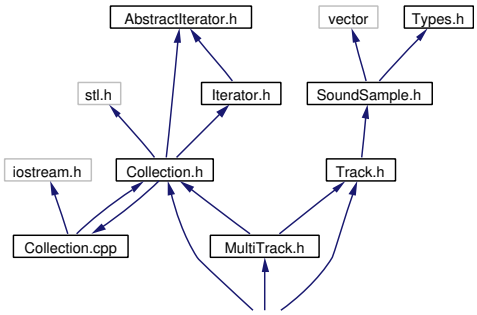
Include dependency graph for Filter.cpp:



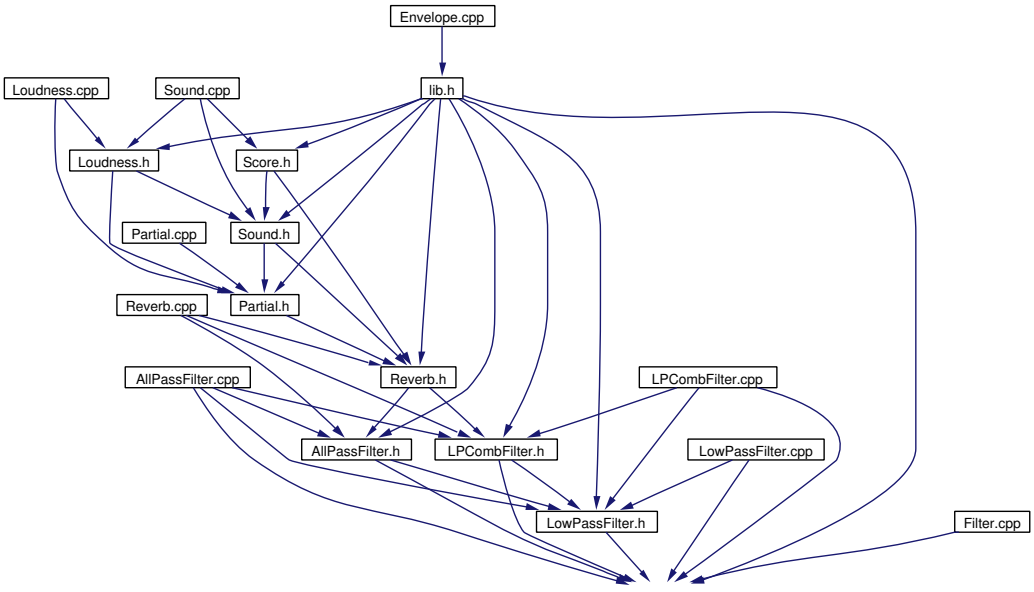
9.24 Filter.h File Reference

```
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
```

Include dependency graph for Filter.h:



This graph shows which files directly or indirectly include this file:



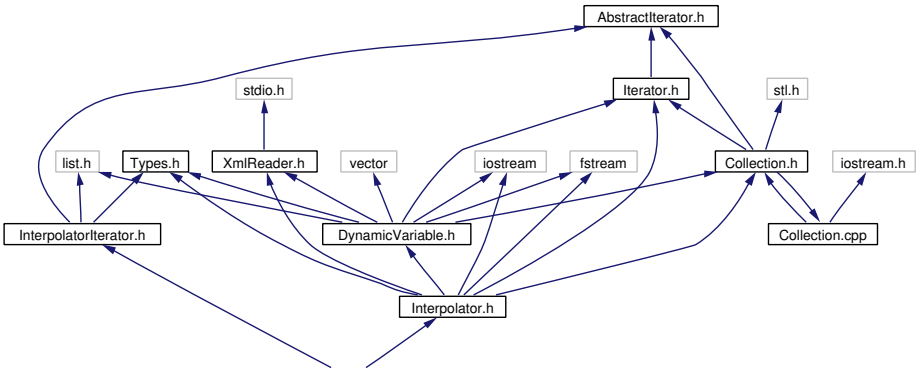
Classes

- class Filter
- class Filter::hist_queue

9.25 Interpolator.cpp File Reference

```
#include "Interpolator.h"
#include "InterpolatorIterator.h"
```

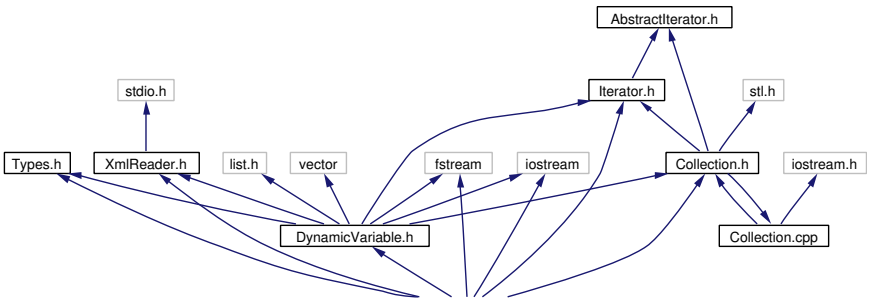
Include dependency graph for Interpolator.cpp:



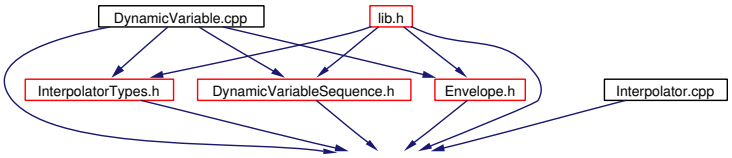
9.26 Interpolator.h File Reference

```
#include <fstream>
#include <iostream>
#include "Types.h"
#include "DynamicVariable.h"
#include "Iterator.h"
#include "Collection.h"
#include "XmlReader.h"
```

Include dependency graph for Interpolator.h:



This graph shows which files directly or indirectly include this file:



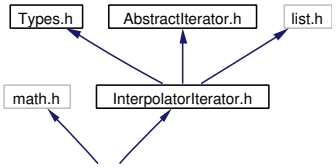
Classes

- class Interpolator
- class InterpolatorEntry

9.27 InterpolatorIterator.cpp File Reference

```
#include <math.h>
#include "InterpolatorIterator.h"
```

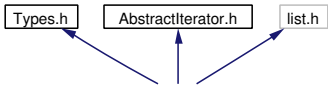
Include dependency graph for InterpolatorIterator.cpp:



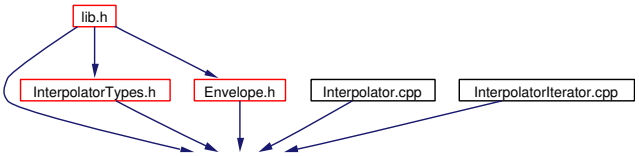
9.28 InterpolatorIterator.h File Reference

```
#include "Types.h"
#include "AbstractIterator.h"
#include <list.h>
```

Include dependency graph for InterpolatorIterator.h:



This graph shows which files directly or indirectly include this file:



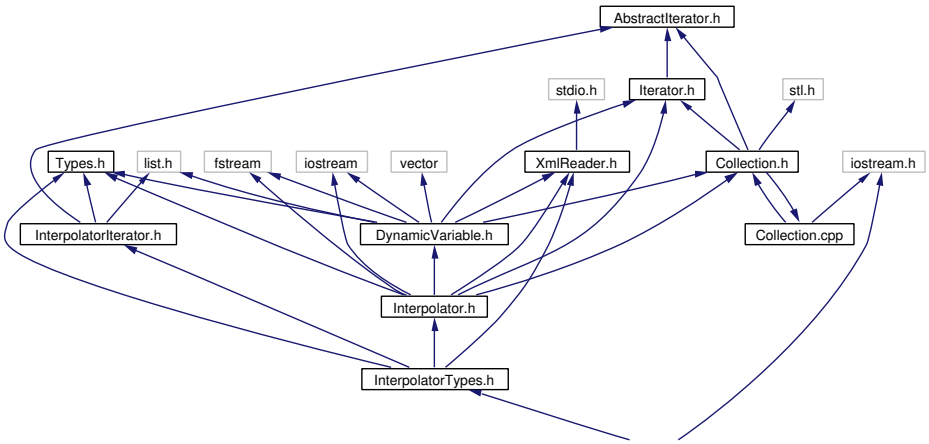
Classes

- class CubicSplineInterpolatorIterator
- class ExponentialInterpolatorIterator
- class InterpolatorIterator
- class InterpolatorIterator::Entry
- class LinearInterpolatorIterator

9.29 InterpolatorTypes.cpp File Reference

```
#include "InterpolatorTypes.h"
#include <iostream.h>
```

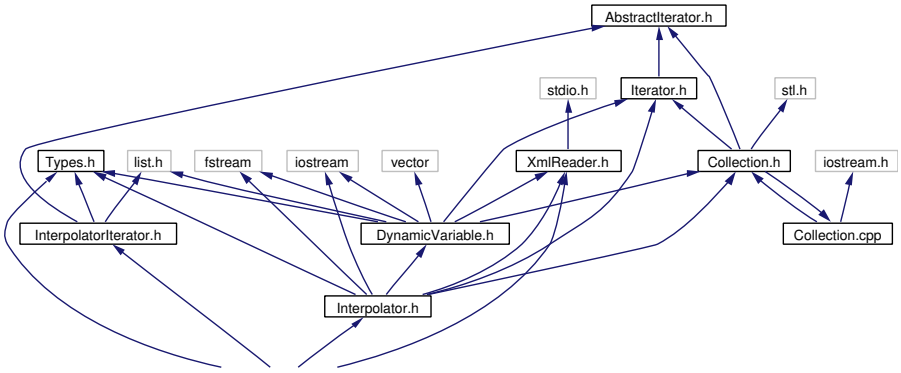
Include dependency graph for InterpolatorTypes.cpp:



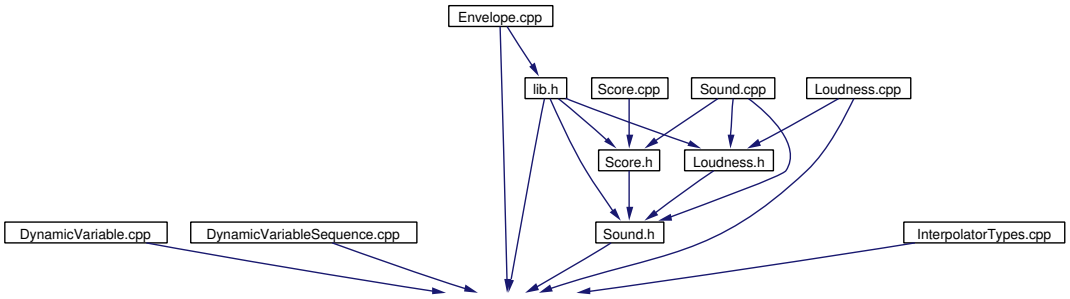
9.30 InterpolatorTypes.h File Reference

```
#include "Types.h"
#include "InterpolatorIterator.h"
#include "Interpolator.h"
#include "XmlReader.h"
```

Include dependency graph for InterpolatorTypes.h:



This graph shows which files directly or indirectly include this file:



Classes

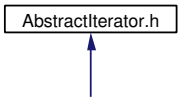
- class CubicSplineInterpolator
- class ExponentialInterpolator

- class LinearInterpolator

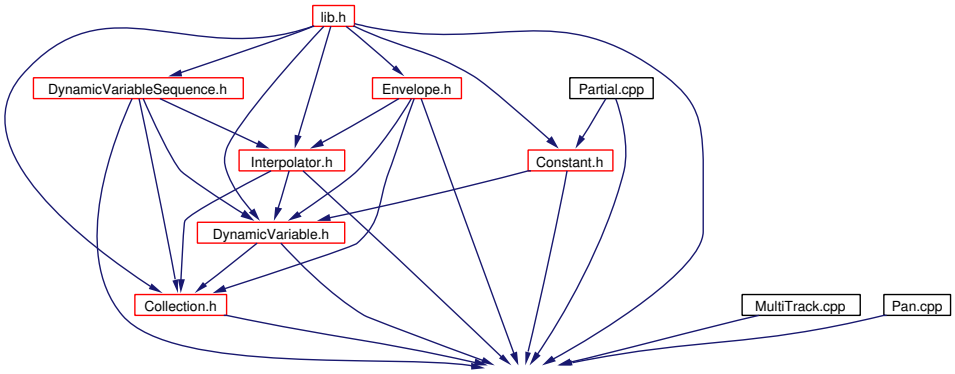
9.31 Iterator.h File Reference

```
#include "AbstractIterator.h"
```

Include dependency graph for Iterator.h:



This graph shows which files directly or indirectly include this file:



Classes

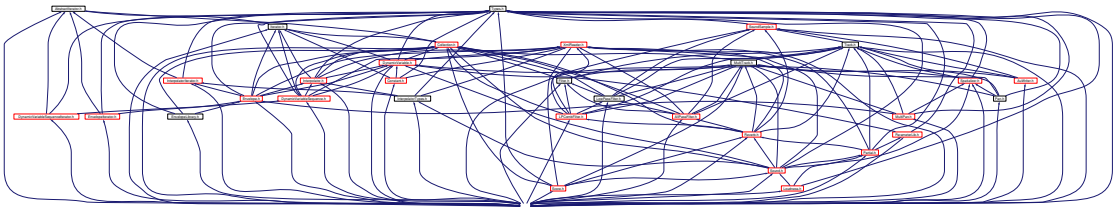
- class Iterator

9.32 lib.h File Reference

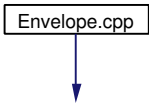
```
#include "AbstractIterator.h"
#include "AllPassFilter.h"
#include "AuWriter.h"
#include "Collection.h"
#include "Constant.h"
#include "DynamicVariable.h"
#include "DynamicVariableSequence.h"
#include "DynamicVariableSequenceIterator.h"
#include "Envelope.h"
#include "EnvelopeIterator.h"
#include "EnvelopeLibrary.h"
#include "Filter.h"
#include "Interpolator.h"
#include "InterpolatorIterator.h"
#include "InterpolatorTypes.h"
#include "Iterator.h"
#include "LPCombFilter.h"
#include "Loudness.h"
#include "LowPassFilter.h"
#include "MultiPan.h"
#include "MultiTrack.h"
#include "Pan.h"
#include "ParameterLib.h"
#include "Partial.h"
#include "Reverb.h"
#include "Score.h"
#include "Sound.h"
#include "SoundSample.h"
#include "Spatializer.h"
```

```
#include "Track.h"
#include "Types.h"
#include "XmlReader.h"
```

Include dependency graph for lib.h:



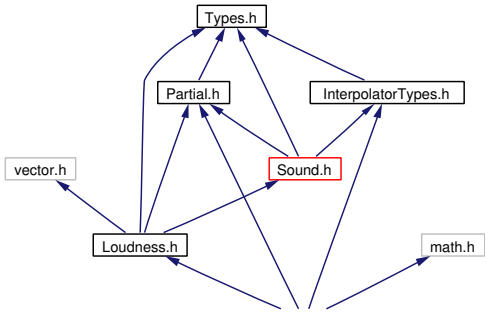
This graph shows which files directly or indirectly include this file:



9.33 Loudness.cpp File Reference

```
#include "Loudness.h"
#include <math.h>
#include "InterpolatorTypes.h"
#include "Partial.h"
```

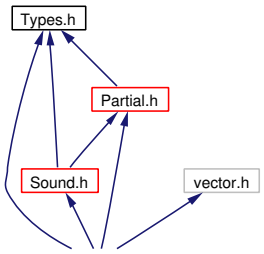
Include dependency graph for Loudness.cpp:



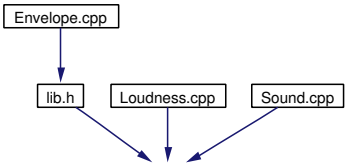
9.34 Loudness.h File Reference

```
#include "Types.h"
#include "Sound.h"
#include "Partial.h"
#include <vector.h>
```

Include dependency graph for Loudness.h:



This graph shows which files directly or indirectly include this file:



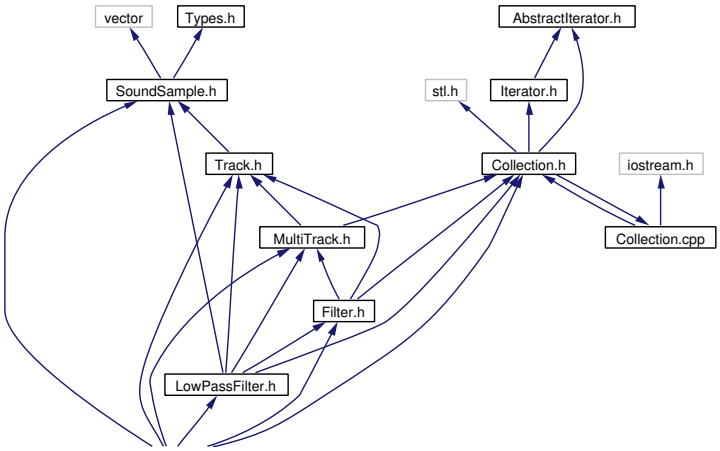
Classes

- class Loudness
- class Loudness::CriticalBand
- class Loudness::PartialSnapshot

9.35 LowPassFilter.cpp File Reference

```
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "Filter.h"
#include "LowPassFilter.h"
```

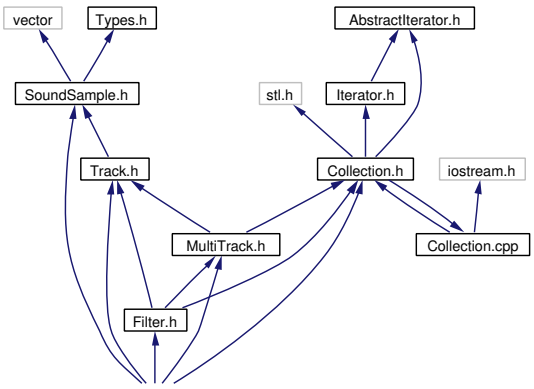
Include dependency graph for LowPassFilter.cpp:



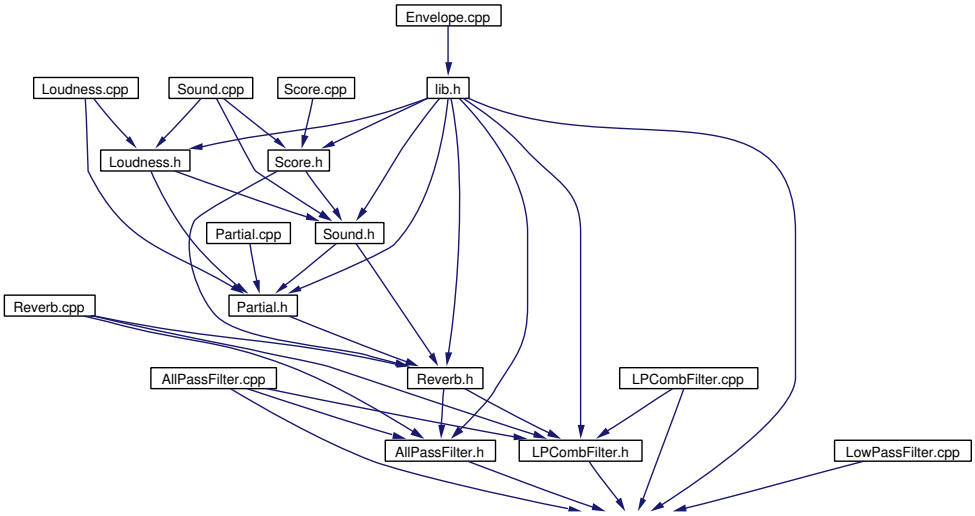
9.36 LowPassFilter.h File Reference

```
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "Filter.h"
```

Include dependency graph for LowPassFilter.h:



This graph shows which files directly or indirectly include this file:



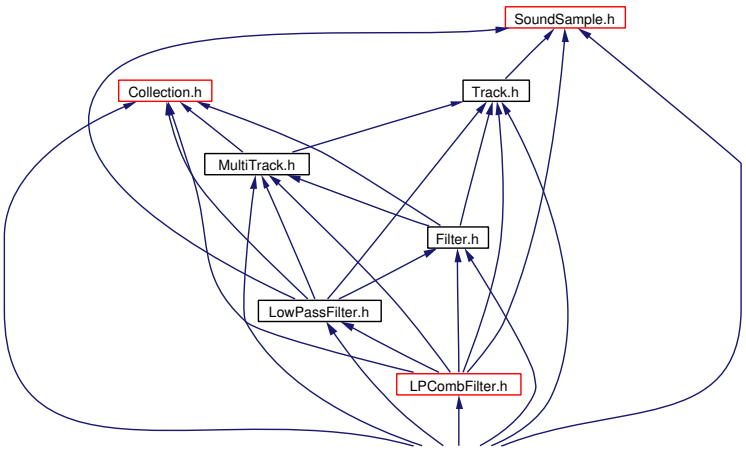
Classes

- class LowPassFilter

9.37 LPCombFilter.cpp File Reference

```
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "Filter.h"
#include "LPCombFilter.h"
#include "LowPassFilter.h"
```

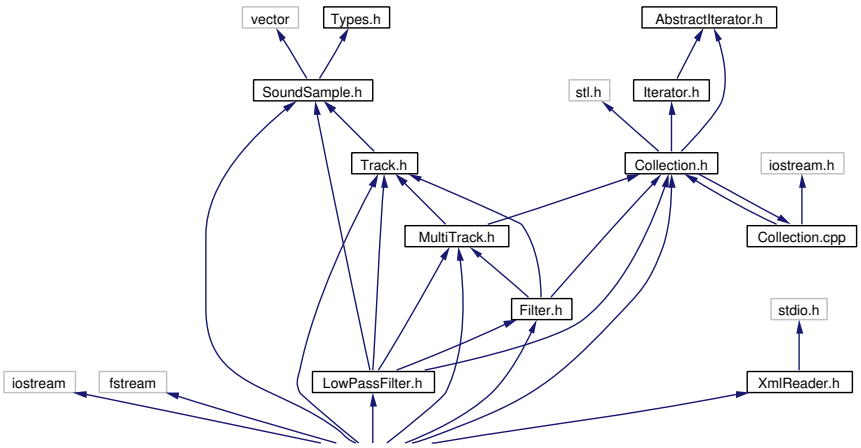
Include dependency graph for LPCombFilter.cpp:



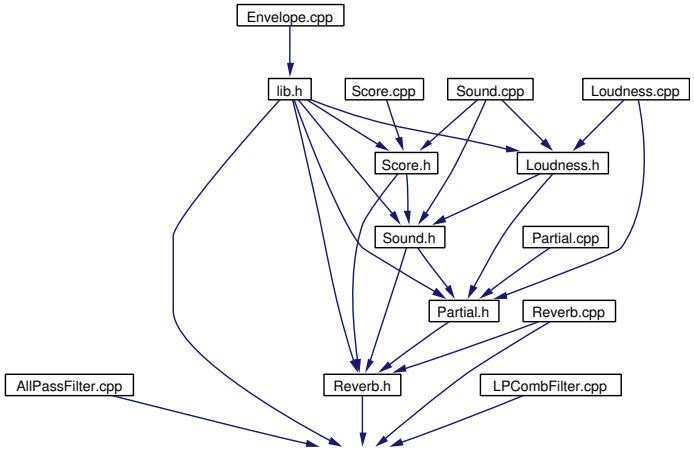
9.38 LPCombFilter.h File Reference

```
#include <iostream>
#include <fstream>
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "LowPassFilter.h"
#include "Filter.h"
#include "XmlReader.h"
```

Include dependency graph for LPCombFilter.h:



This graph shows which files directly or indirectly include this file:



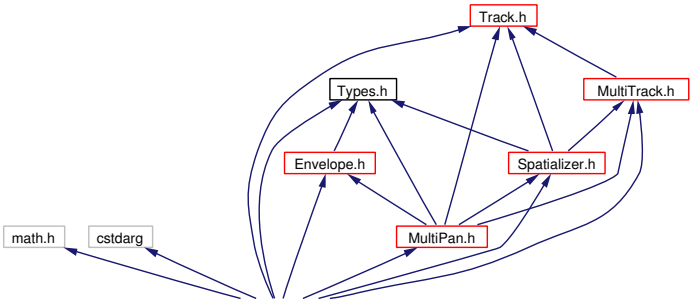
Classes

- class LPCombFilter

9.39 MultiPan.cpp File Reference

```
#include <math.h>
#include <cstdarg>
#include "Types.h"
#include "Spatializer.h"
#include "MultiTrack.h"
#include "Track.h"
#include "Envelope.h"
#include "MultiPan.h"
```

Include dependency graph for MultiPan.cpp:



Defines

- #define JBL_SQRD(x) ((x)∗(x))

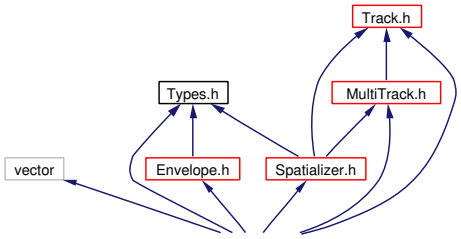
9.39.1 Define Documentation

9.39.1.1 #define JBL_SQRD(x) ((x)∗(x))

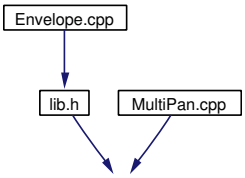
9.40 MultiPan.h File Reference

```
#include <vector>
#include "Types.h"
#include "Spatializer.h"
#include "MultiTrack.h"
#include "Track.h"
#include "Envelope.h"
```

Include dependency graph for MultiPan.h:



This graph shows which files directly or indirectly include this file:



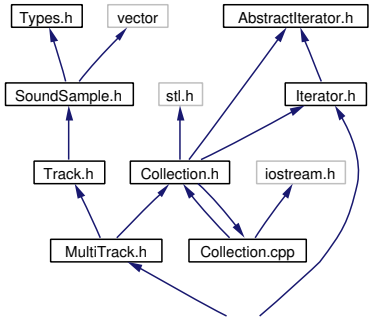
Classes

- class MultiPan

9.41 MultiTrack.cpp File Reference

```
#include "MultiTrack.h"
#include "Iterator.h"
```

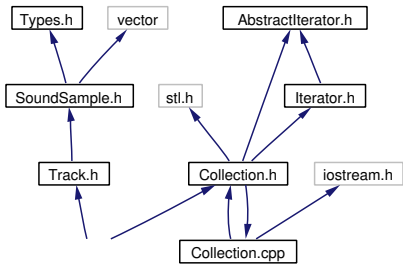
Include dependency graph for MultiTrack.cpp:



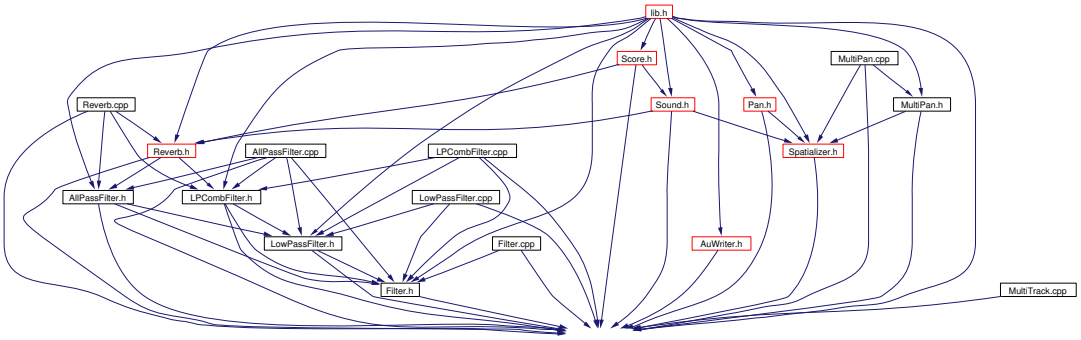
9.42 MultiTrack.h File Reference

```
#include "Track.h"
#include "Collection.h"
```

Include dependency graph for MultiTrack.h:



This graph shows which files directly or indirectly include this file:



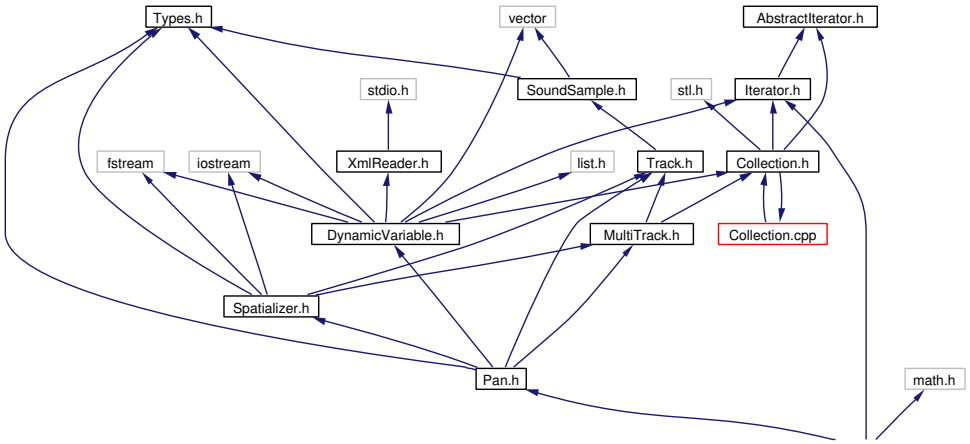
Classes

- class MultiTrack

9.43 Pan.cpp File Reference

```
#include "Pan.h"
#include "Iterator.h"
#include <math.h>
```

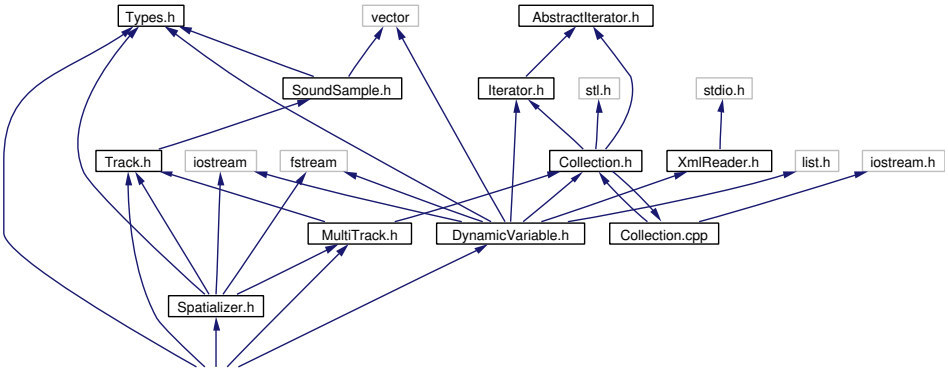
Include dependency graph for Pan.cpp:



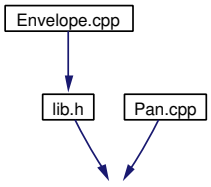
9.44 Pan.h File Reference

```
#include "Types.h"
#include "Spatializer.h"
#include "MultiTrack.h"
#include "Track.h"
#include "DynamicVariable.h"
```

Include dependency graph for Pan.h:



This graph shows which files directly or indirectly include this file:



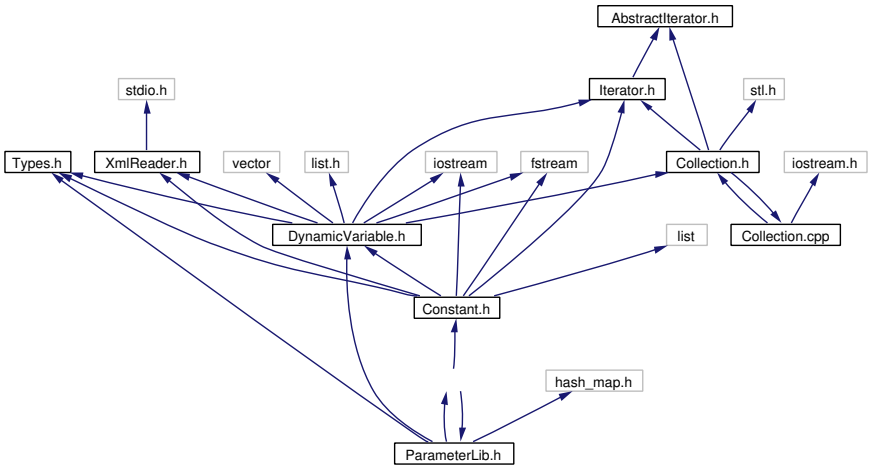
Classes

- class Pan

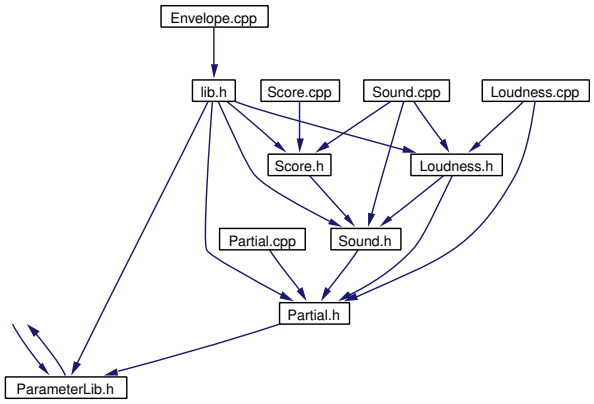
9.45 ParameterLib.cpp File Reference

```
#include "ParameterLib.h"
#include "Constant.h"
```

Include dependency graph for ParameterLib.cpp:



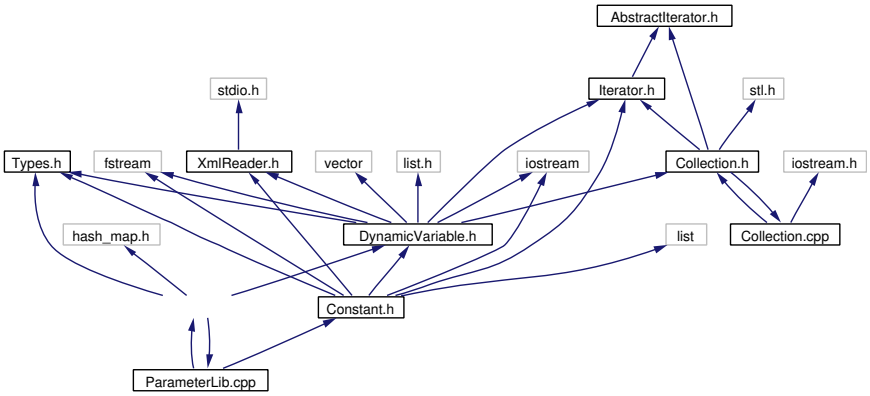
This graph shows which files directly or indirectly include this file:



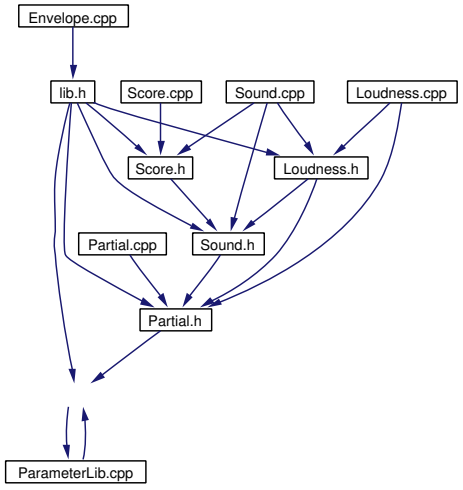
9.46 ParameterLib.h File Reference

```
#include "Types.h"
#include "DynamicVariable.h"
#include <hash_map.h>
#include "ParameterLib.cpp"
```

Include dependency graph for ParameterLib.h:



This graph shows which files directly or indirectly include this file:



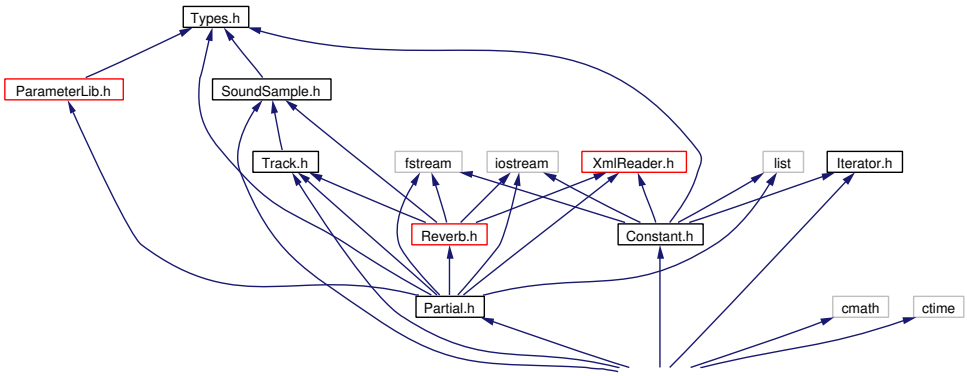
Classes

- class ParameterLib

9.47 Partial.cpp File Reference

```
#include "Partial.h"
#include <cmath>
#include <ctime>
#include "SoundSample.h"
#include "Constant.h"
#include "Track.h"
#include "Iterator.h"
```

Include dependency graph for Partial.cpp:



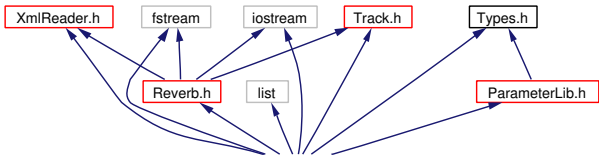
Namespaces

- namespace std

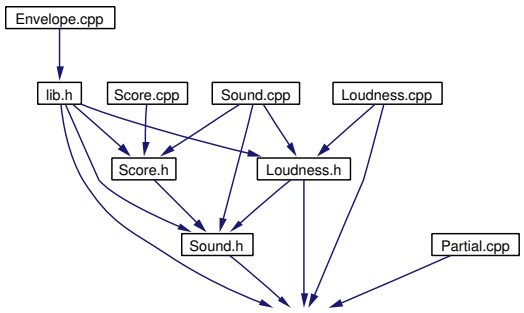
9.48 Partial.h File Reference

```
#include "XmlReader.h"
#include <fstream>
#include <iostream>
#include <list>
#include "Types.h"
#include "ParameterLib.h"
#include "Track.h"
#include "Reverb.h"
```

Include dependency graph for Partial.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Partial

Enumerations

- enum PartialStaticParam {
  RELATIVE_AMPLITUDE,
  PARTIAL_NUM,
  WAVE_TYPE }
- enum PartialDynamicParam {
  FREQUENCY,
  WAVE_SHAPE,
  TREMOLO_AMP,
  TREMOLO_RATE,
  VIBRATO_AMP,
  VIBRATO_RATE,
  PHASE,
  LOUDNESS_SCALAR,
  FREQ_ENV,
  DETUNING_ENV,
  AMPTRANS_AMP_ENV,
  AMPTRANS_RATE_ENV,
  FREQTRANS_AMP_ENV,
  FREQTRANS_RATE_ENV,
  AMPTRANS_WIDTH,
  FREQTRANS_WIDTH }

9.48.1 Detailed Description

Defines a Partial object, as well as the static and dynamic parameters that pertain to said object.

Definition in file Partial.h.

9.48.2 Enumeration Type Documentation

9.48.2.1 enum PartialDynamicParam

Defines the Dynamic parameters that can be set for a Partial object.

- FREQUENCY

  - The pitch at which a partial is heard.
  - Set in Hz. (no bounds checking)

- WAVE_SHAPE
  - How the partial changes it's amplitude over time.
  - Envelope should start and end at y=0; otherwise "clicks" and "pops" will be created

- TREMOLO_AMP
  - The amplitude of tremolo (size of the effect).
  - Given as a scaling factor to amplitude.

- TREMOLO_RATE
  - Given in Hz. (see vibrato rate)

- VIBRATO_AMP
  - The amplitude of vibrato (size of the effect).
  - Given as a scaling factor to frequency.

- VIBRATO_RATE
  - Given in Hz. (6 Hz is a "normal" vibrato)

- FREQUENCY_DEVIATION ****Commmented out (i.e. not used anywhere, but can be put back in)*****
  - Specifies how randomly scaled the frequencies of this partial will be.
  - Range [0 - 1]
  - Could be specified as a value or as an envelope which will affect the frequency. Exactly the same effect can be obtained by using GLISSANDO_ENV or DETUNING_ENV. See comment for glissando envelope.

- LOUDNESS_SCALAR
  - Not to be set by users.
  - This variable is set by the Loudness routines.
  - The scaling factor is then taken into account at every sample.

- GLISSANDO_ENV *******Commented Out (i.e. not used anywhere, but can be put back in) *****
  - This is a glissando envelope. It is multiplied by the
  - frequency at every point. Thus, if you leave it at 1.0 (The default, it will not affect anything). If you leave it at 1.0 for most of the sound, then interpolate it down to 0.5, the partial will trail off (by an octave down) at the end of the sound.

  - The y values for this envelope can be any positive number. a value of 0 will kill the sound (0 frequency), and too high of a value could make the frequency inaudible or above Nyquist (no bounds checking).

- DETUNING_ENV ******** (Commmented Out) *********
  - used to gradually tune or detune a partial. It's an envelope, and acts just like a glissando envelope, but more general.
- AMPTRANS_AMP_ENV
  - used to control the value of amplitude transients
  - the value of the amp envelope is a maximum transient modifier
  - this value is multiplied by a random percentage and then used to modify the amplitude as follows: If the value after the percentage is applied is 0.7, then the amplitude will be 1.7 or 0.3 times its original value.
- AMPTRANS_RATE_ENV
  - used to control the rate of amplitude transients
  - the value of the rate envelope is the percentage chance of a transient occuring at that particular time
- FREQTRANS_AMP_ENV
  - used to control the amplitude of frequency transients
  - see AMPTRANS_AMP_ENV for an explanation
- FREQTRANS_RATE_ENV
  - used to control the rate of frequency transients
  - see AMPTRANS_RATE_ENV for an explanation
- AMPTRANS_WIDTH
  - the width of any amplitude transient in samples
  - defaults to 1103, or 0.025 seconds.
- FREQTRANS_WIDTH -the width of any frequency transient in samples
  - defaults to 1103, or 0.025 seconds.

Enumeration values:

FREQUENCY

WAVE_SHAPE

TREMOLO_AMP

TREMOLO_RATE

VIBRATO_AMP

VIBRATO_RATE
PHASE
LOUDNESS_SCALAR
FREQ_ENV
DETUNING_ENV
AMPTRANS_AMP_ENV
AMPTRANS_RATE_ENV
FREQTRANS_AMP_ENV
FREQTRANS_RATE_ENV
AMPTRANS_WIDTH
FREQTRANS_WIDTH

Definition at line 134 of file Partial.h.

9.48.2.2 enum PartialStaticParam

Defines the static parameters that can be set for a Partial object.

- RELATIVE_AMPLITUDE
  - Used by Loudness to balance the amplitude of partials.
  - Set these values in any range you like.
- PARTIAL_NUM
  - Defines the number partial.
  - 0 is the lowest partial.
  - Currently only used for Frequency Deviation calculations.
- WAVE_TYPE
  - Specify type of wave

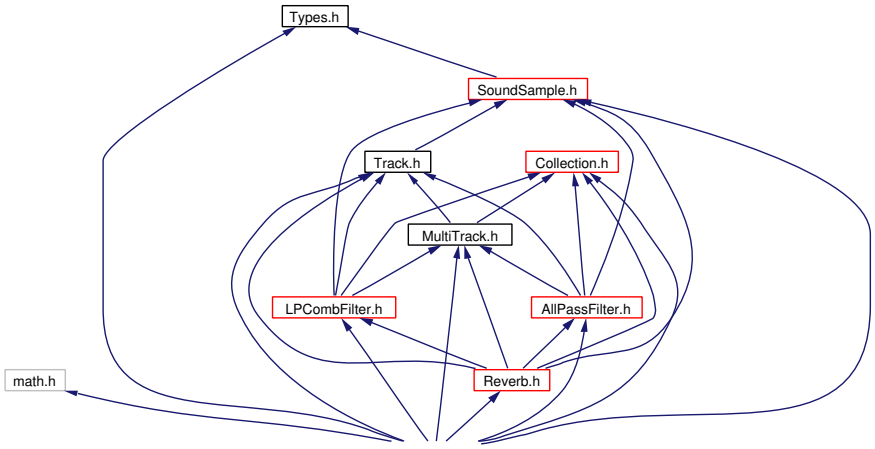
Enumeration values:
RELATIVE_AMPLITUDE
PARTIAL_NUM
WAVE_TYPE

Definition at line 58 of file Partial.h.

9.49 Reverb.cpp File Reference

```
#include <math.h>
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "LPCombFilter.h"
#include "AllPassFilter.h"
#include "Reverb.h"
#include "Types.h"
```

Include dependency graph for Reverb.cpp:



Defines

- #define max(x, y) ((x) > (y) ? (x) : (y))

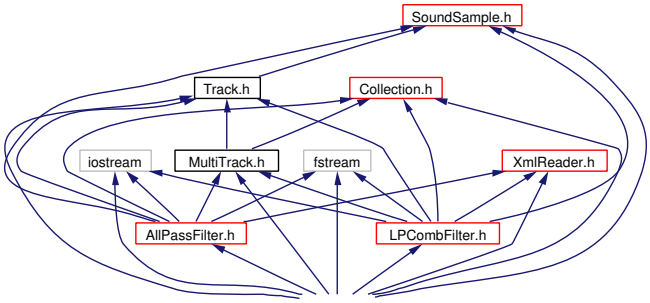
9.49.1 Define Documentation

9.49.1.1 #define max(x, y) ((x) > (y) ? (x) : (y))

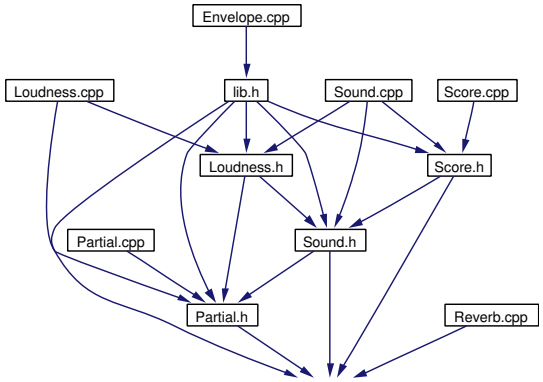
9.50 Reverb.h File Reference

```
#include <iostream>
#include <fstream>
#include "SoundSample.h"
#include "Collection.h"
#include "Track.h"
#include "MultiTrack.h"
#include "LPCombFilter.h"
#include "AllPassFilter.h"
#include "XmlReader.h"
```

Include dependency graph for Reverb.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Reverb

Defines

- #define REVERB_NUM_COMB_FILTERS 6

9.50.1 Define Documentation

9.50.1.1 #define REVERB_NUM_COMB_FILTERS 6

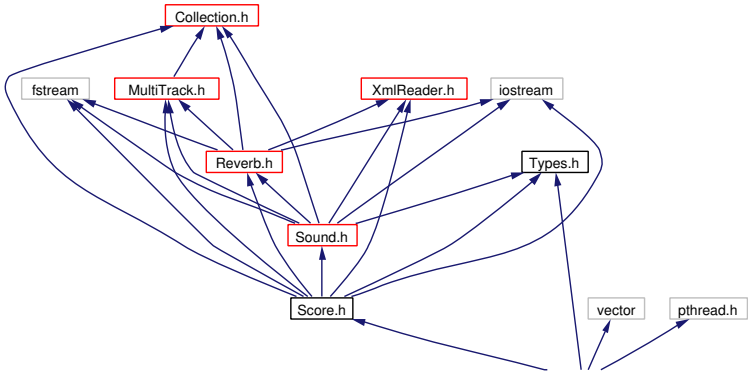
Definition at line 44 of file Reverb.h.

Referenced by Reverb::ConstructorCommon(), Reverb::do_reverb(), Reverb::reset(), Reverb::Reverb(), Reverb::xml_print(), and Reverb::∼Reverb().

9.51 Score.cpp File Reference

```
#include "Score.h"
#include "Types.h"
#include <vector>
#include <pthread.h>
```

Include dependency graph for Score.cpp:



Functions

- void * multithreaded_render_worker (void *vtle)

9.51.1 Function Documentation

9.51.1.1 void* multithreaded_render_worker (void * *vtle*)

Definition at line 39 of file Score.cpp.

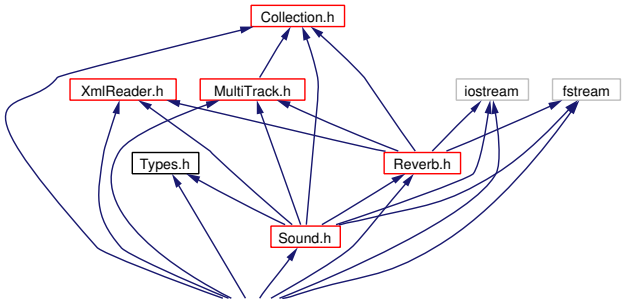
References threadlist_entry::done, threadlist_entry::finishCondition, threadlist_entry::listMutex, threadlist_entry::numChannels, Sound::render(), threadlist_entry::resultTrack, threadlist_entry::samplingRate, and threadlist_entry::snd.

Referenced by Score::render().

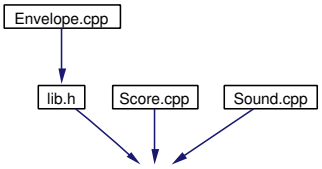
9.52 Score.h File Reference

```
#include "XmlReader.h"
#include "Types.h"
#include "Collection.h"
#include "MultiTrack.h"
#include "Sound.h"
#include "Reverb.h"
#include <iostream>
#include <fstream>
```

Include dependency graph for Score.h:



This graph shows which files directly or indirectly include this file:



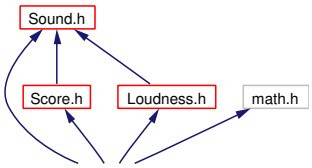
Classes

- class Score
- struct threadlist_entry

9.53 Sound.cpp File Reference

```
#include "Sound.h"
#include "Score.h"
#include <math.h>
#include "Loudness.h"
```

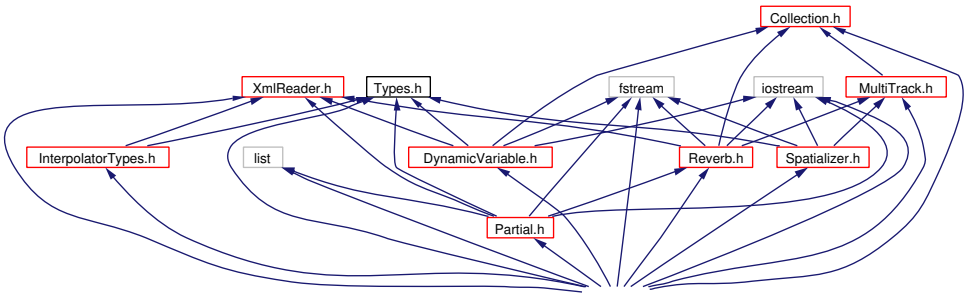
Include dependency graph for Sound.cpp:



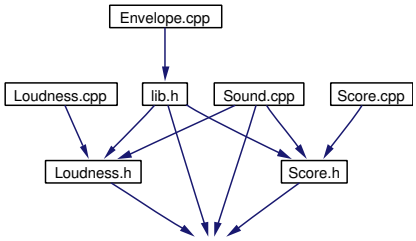
9.54 Sound.h File Reference

```
#include "XmlReader.h"
#include <list>
#include "Types.h"
#include "Partial.h"
#include "MultiTrack.h"
#include "Collection.h"
#include "DynamicVariable.h"
#include "Spatializer.h"
#include "Reverb.h"
#include "InterpolatorTypes.h"
#include <iostream>
#include <fstream>
```

Include dependency graph for Sound.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Sound

Enumerations

- enum SoundStaticParam {
  START_TIME,
  DURATION,
  LOUDNESS,
  LOUDNESS_RATE,
  DETUNE_SPREAD,
  DETUNE_DIRECTION,
  DETUNE_VELOCITY,
  DETUNE_FUNDAMENTAL }
- enum SoundDynamicParam

9.54.1 Detailed Description

Defines a Sound object, as well as the static and dynamic parameters that pertain to said object.

Definition in file Sound.h.

9.54.2 Enumeration Type Documentation

9.54.2.1 enum SoundDynamicParam

Defines the DynamicParameters that can be set for a Sound object.

Definition at line 134 of file Sound.h.

9.54.2.2 enum SoundStaticParam

Enumeration values:

START_TIME • When placed in a Score, the sound will start at this time.

DURATION • The sound lasts this long (in seconds)

LOUDNESS • Describes how loud the sound should be perceived at its loudest point.
- Given in Sones, the valid range is [0,255]

LOUDNESS_RATE • Loudness does not have to be calculated EVERY sample.
- This specifies how often loudness is calculated.
- the default is 10Hz.

DETUNE_SPREAD • Detuning describes the effect of letting all partials exponentially converge from random points to their intended frequencies, or to diverge in the reverse fashion
- Detuning spread refers to the the variance in frequences at the divergent end of the sound (the beginning for convergence (tuning), the end for divergence (detuning))
- A value of 0.0 effectively disables detuning
- A value of, say, 0.3 causes the detuned frequencies to fall within the range [(1.0-0.3)∗Freq, (1.0+0.3)∗Freq]. In other words, the SPREAD value is a percent that corresponds to a range in which the partial will (randomly) fall at the max. detuning portion of the sound.

DETUNE_DIRECTION • -1.0 means do detuning (divergence)
- +1.0 means do tuning (convergence)

DETUNE_VELOCITY • Values over [-1.0, +1.0]
- For VELOCITY=0.5, the transition will be linear from start to end. For Velocity = -1.0, the transition will be nearly instantaneous and immediate (and exponentially interpolated). For velocity +1.0, the transition will be instantaneous, but will occur at the extreme end of the sound (And will be exponentially interpolated)
- If y is the amount of tuning and x is time, think of -1.0 as a vertical line on the left (at the beginning) and then a flat horizontal line. For +1.0, the horizontal line is at the beginning (on the left), and there is a sharp transition at the right end (at the end).

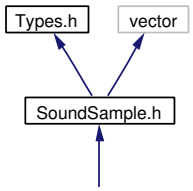
DETUNE_FUNDAMENTAL • Positive values = TRUE (tune/detune the fundamental)
- Negative Values = FALSE (do not tune/detune ...)

Definition at line 118 of file Sound.h.

9.55 SoundSample.cpp File Reference

```
#include "SoundSample.h"
```

Include dependency graph for SoundSample.cpp:



Defines

- #define MIN_CLIP_WARNING 10

9.55.1 Define Documentation

9.55.1.1 #define MIN_CLIP_WARNING 10

Because m_time_type is a float, there can be a slight round-off error that causes different calculations to produce different numbers of samples for a given Track/SoundSample. We would like to warn the user if he tries to composite two sections of vastly different lengths, but we need to have some acceptable error to account for round-off errors inherent in the program and not caused by users

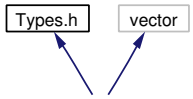
Definition at line 43 of file SoundSample.cpp.

Referenced by SoundSample::composite().

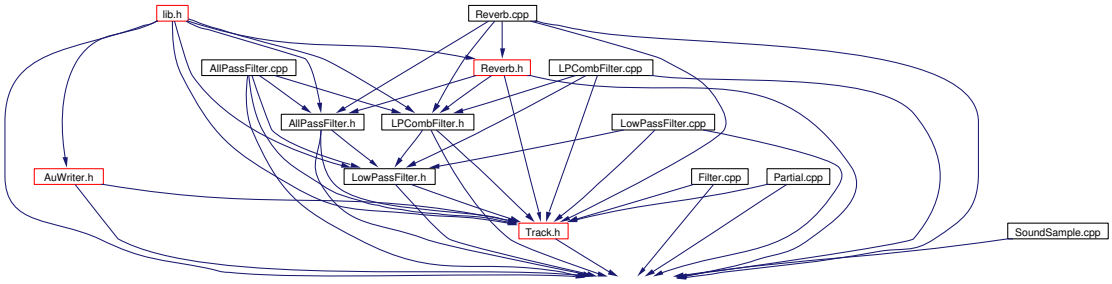
9.56 SoundSample.h File Reference

```
#include "Types.h"
#include <vector>
```

Include dependency graph for SoundSample.h:



This graph shows which files directly or indirectly include this file:



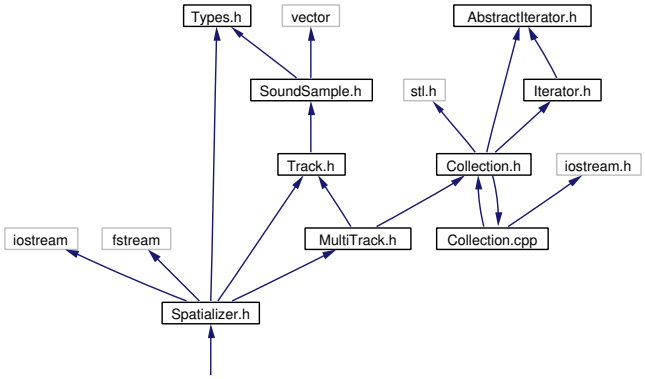
Classes

- class SoundSample

9.57 Spatializer.cpp File Reference

```
#include "Spatializer.h"
```

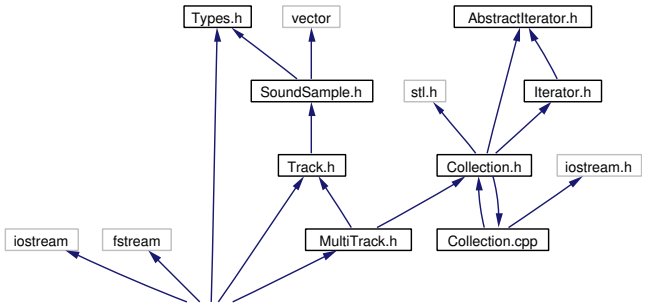
Include dependency graph for Spatializer.cpp:



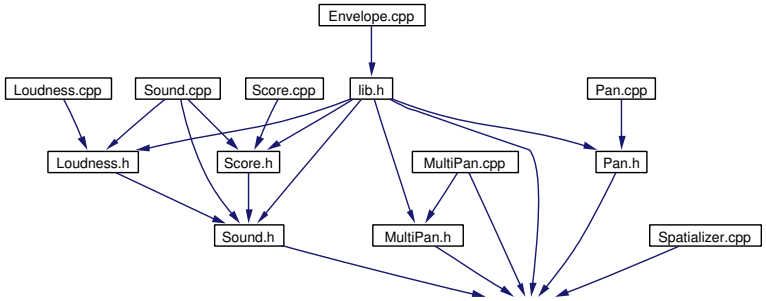
9.58 Spatializer.h File Reference

```
#include <iostream>
#include <fstream>
#include "Types.h"
#include "MultiTrack.h"
#include "Track.h"
```

Include dependency graph for Spatializer.h:



This graph shows which files directly or indirectly include this file:



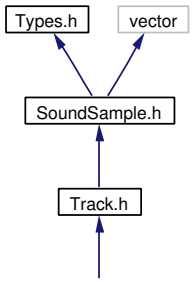
Classes

- class Spatializer

9.59 Track.cpp File Reference

```
#include "Track.h"
```

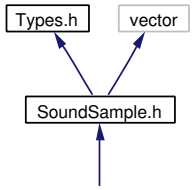
Include dependency graph for Track.cpp:



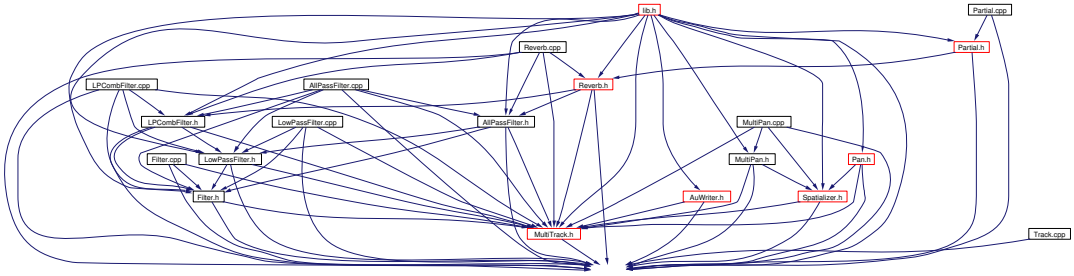
9.60 Track.h File Reference

```
#include "SoundSample.h"
```

Include dependency graph for Track.h:



This graph shows which files directly or indirectly include this file:

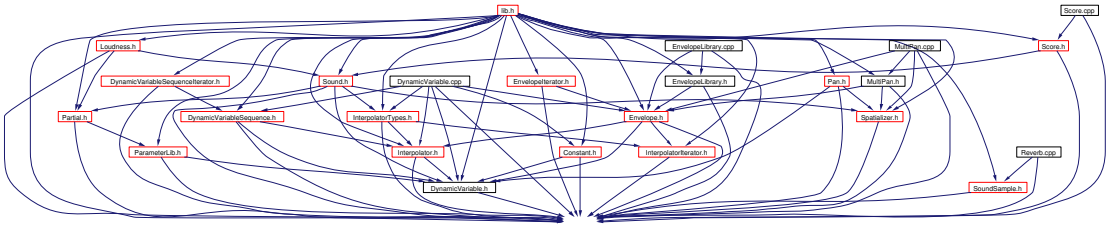


Classes

- class Track

9.61 Types.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct env_seg
- struct envelope_segment
- struct xy_point

Typedefs

- typedef float m_sample_type

  Specifies a type for an individual sample value.

- typedef long m_sample_count_type

  Specifies a sample count.

- typedef float m_time_type

  Specifies a type for an individual time value.

- typedef float m_value_type

  Specifies a value type (used for frequency and amplitude).

- typedef unsigned int m_rate_type

  Specifies a rate for playback.

Enumerations

- enum stretch_type {

  FIXED,

FLEXIBLE }

- enum interpolation_type {

  EXPONENTIAL,

  CUBIC_SPLINE,

  LINEAR }

Variables

- const m_rate_type DEFAULT_SAMPLING_RATE = 44100

  When no sampling rate is specified, uses 44.1kHz.

- const m_rate_type DEFAULT_LOUDNESS_RATE = 10

  Loudness can be calculated at much slower rates. The default rate is 10Hz.

9.61.1 Detailed Description

This file is included to define basic filetypes for the application. In this manner, we can easily change from float-sound to double-sound by editing one line of code.

Author:
Braden Kowitz

Definition in file Types.h.

9.61.2 Typedef Documentation

9.61.2.1 typedef unsigned int m_rate_type

Specifies a rate for playback.

Definition at line 53 of file Types.h.

Referenced by Envelope::addInterpolators(), DynamicVariableSequence::addInterpolators(), Loudness::calculate(), Reverb::constructAmp(), Reverb::ConstructorCommon(), SoundSample::getSamplingRate(), DynamicVariable::getSamplingRate(), MultiTrack::MultiTrack(), Sound::render(), Score::render(), Partial::render(), Reverb::Reverb(), SoundSample::setSamplingRate(), DynamicVariable::setSamplingRate(), SoundSample::SoundSample(), Pan::spatialize(), MultiPan::spatialize(), Track::Track(), and AuWriter::write().

9.61.2.2 typedef long m_sample_count_type

Specifies a sample count.

Definition at line 44 of file Types.h.

Referenced by Score::anticlip(), InterpolatorIterator::append(), Loudness::calculate(), Score::channelAnticlip(), Score::channelScale(), Score::clip(), SoundSample::composite(), Constant::ConstantIterator::ConstantIterator(), Reverb::constructAmp(), InterpolatorIterator::Entry::Entry(), SoundSample::getSampleCount(), DynamicVariable::getSampleCount(), MultiTrack::MultiTrack(), SoundSample::operator[ ](), Sound::render(), Score::render(), Partial::render(), Score::scale(), SoundSample::SoundSample(), Pan::spatialize(), MultiPan::spatialize(), Track::Track(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), and AuWriter::write().

9.61.2.3 typedef float m_sample_type

Specifies a type for an individual sample value.

Definition at line 41 of file Types.h.

Referenced by Score::anticlip(), Score::channelScale(), Reverb::constructAmp(), LPCombFilter::do_filter(), LowPassFilter::do_filter(), AllPassFilter::do_filter(), Reverb::do_reverb(), SoundSample::operator[ ](), Score::scale(), and AuWriter::write().

9.61.2.4 typedef float m_time_type

Specifies a type for an individual time value.

Definition at line 47 of file Types.h.

Referenced by Interpolator::addEntry(), DynamicVariableSequence::addInterpolators(), DynamicVariableSequence::AddToShape(), InterpolatorIterator::append(), Loudness::calculate(), Track::composite(), SoundSample::composite(), MultiTrack::composite(), Reverb::ConstructorCommon(), InterpolatorIterator::Entry::Entry(), Envelope::generateLengths(), DynamicVariableSequence::generateTimes(), DynamicVariable::getDuration(), DynamicVariableSequence::getSegmentTime(), Sound::getTotalDuration(), Partial::getTotalDuration(), DynamicVariableSequence::getValue(), InterpolatorEntry::InterpolatorEntry(), EnvelopeLibrary::loadLibrary(), CubicSplineInterpolatorIterator::next(), ExponentialInterpolatorIterator::next(), Sound::render(), Score::render(), Partial::render(), DynamicVariable::setDuration(), DynamicVariableSequence::setSegmentTime(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), LinearInterpolator::valueIterator(), and AuWriter::write().

9.61.2.5 typedef float m_value_type

Specifies a value type (used for frequency and amplitude).

Definition at line 50 of file Types.h.

Referenced by Interpolator::addEntry(), Envelope::addInterpolators(), Envelope::AddToShape(), InterpolatorIterator::append(), Loudness::calculate(), Constant::Constant(), Constant::ConstantIterator::ConstantIterator(), Loudness::criticalBandIndex(), Envelope::DefineShape(), InterpolatorIterator::Entry::Entry(), Loudness::CriticalBand::getBandGamma(), Interpolator::getMaxValue(), Envelope::getMaxValue(), DynamicVariableSequence::getMaxValue(), Constant::getMaxValue(), Loudness::PartialSnapshot::getScalingFactor(), Envelope::getSegmentLength(), Envelope::getValue(), DynamicVariableSequence::getValue(), Constant::getValue(), InterpolatorEntry::InterpolatorEntry(), CubicSplineInterpolatorIterator::next(), ExponentialInterpolatorIterator::next(), LinearInterpolatorIterator::next(), EnvelopeIterator::next(), DynamicVariableSequenceIterator::next(), Loudness::PartialSnapshot::PartialSnapshot(), Partial::pmod(), Partial::render(), Track::scale(), SoundSample::scale(), Interpolator::scale(), Envelope::scale(), DynamicVariableSequence::scale(), Constant::scale(), ParameterLib< StaticT, DynamicT >::setParam(), Sound::setPartialParam(), Envelope::setSegmentLength(), Constant::setValue(), Sound::Sound(), Pan::spatialize(), MultiPan::spatialize(), CubicSplineInterpolator::valueIterator(), ExponentialInterpolator::valueIterator(), and LinearInterpolator::valueIterator().

9.61.3 Enumeration Type Documentation

9.61.3.1 enum interpolation_type

This is used in EnvelopeEntry to specify what kind of interpolation the entry should have.

Enumeration values:

EXPONENTIAL

CUBIC_SPLINE

LINEAR

Definition at line 66 of file Types.h.

Referenced by CubicSplineInterpolator::getType(), ExponentialInterpolator::getType(), LinearInterpolator::getType(), EnvelopeLibrary::loadLibrary(), Envelope::setSegmentInterpolationType(), and DynamicVariableSequence::setSegmentInterpolationType().

9.61.3.2 enum stretch_type

This is used in EnvelopeEntry to specify whether the entry should be played for a fixed amount of time or a percentage of total time.

Enumeration values:

FIXED

FLEXIBLE

Definition at line 59 of file Types.h.

Referenced by Envelope::getSegmentLengthType(), DynamicVariableSequence::getSegmentTimeType(), and EnvelopeLibrary::loadLibrary().

9.61.4 Variable Documentation

9.61.4.1 const m_rate_type DEFAULT_LOUDNESS_RATE = 10 [static]

Loudness can be calculated at much slower rates. The default rate is 10Hz.

Definition at line 124 of file Types.h.

9.61.4.2 const m_rate_type DEFAULT_SAMPLING_RATE = 44100 [static]

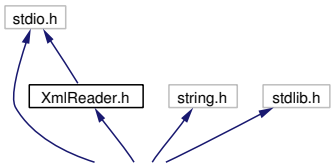
When no sampling rate is specified, uses 44.1kHz.

Definition at line 121 of file Types.h.

9.62 XmlReader.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "XmlReader.h"
```

Include dependency graph for XmlReader.cpp:

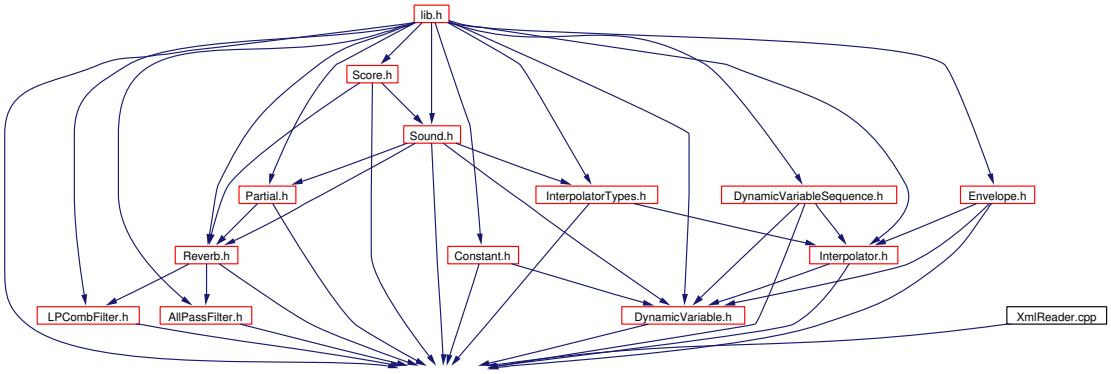


9.63 XmlReader.h File Reference

`#include <stdio.h>`

Include dependency graph for XmlReader.h:

This graph shows which files directly or indirectly include this file:



Classes

- class XmlReader
- class XmlReader::tagparam
- class XmlReader::xmltag
- class XmlReader::xmltagset

Defines

- #define XML_BUFFER_SIZE 1024
- #define XML_DEBUG

9.63.1 Detailed Description

Deprecated

Defines the XmlReader object and it's related objects.

Definition in file XmlReader.h.

9.63.2 Define Documentation

9.63.2.1 #define XML_BUFFER_SIZE 1024

Definition at line 28 of file XmlReader.h.

Referenced by XmlReader::fillTagBuffer(), and XmlReader::XmlReader().

9.63.2.2 #define XML_DEBUG

Definition at line 29 of file XmlReader.h.

Chapter 10

LASS Page Documentation

10.1 Deprecated List

Member AllPassFilter::xml_print(ofstream &xmlOutput) This outputs an XML representation of the object to STDOUT.

Member AllPassFilter::xml_read(XmlReader::xmltag *apftag) Reads some xml and sets the gain and delay.

Member Constant::xml_print(ofstream &xmlOutput)

Member Constant::xml_print(ofstream &xmlOutput, list< DynamicVariable * > &dynObjs) This outputs an XML representation of the object to STDOUT

Member Constant::xml_read(XmlReader::xmltag *xml) This sets the duration, sampling rate, and value if they they are contained in the xml.

Member DynamicVariable::create_dv_from_xml(XmlReader::xmltag *dvtag) This create a DynamicVariable from xml

Member DynamicVariable::xml_print(ofstream &xmlOutput)=0 This outputs an XML representation of the object to STDOUT

Member DynamicVariable::xml_print(ofstream &xmlOutput, list< DynamicVariable * > &dynObjs)=0 This outputs an XML representation of the object to STDOUT

Member DynamicVariableSequence::xml_print(ofstream &xmlOutput) This outputs an XML representation of the object to STDOUT

Member DynamicVariableSequence::xml_print(ofstream &xmlOutput, list< DynamicVariable * > &dyn This outputs an XML representation of the object to STDOUT

Member DynamicVariableSequence::xml_read(XmlReader *xml, char *tagname) This reads the DVS from xml

Member Envelope::xml_print(ofstream &xmlOutput) This outputs an XML representation of the object to STDOUT

Member Envelope::xml_print(ofstream &xmlOutput, list< DynamicVariable * > &dynObjs) This outputs an XML representation of the object to STDOUT

Member Envelope::xml_read(XmlReader::xmltag *soundtag)

Member Interpolator::xml_print(ofstream &xmlOutput)

Member Interpolator::xml_print(ofstream &xmlOutput, list< DynamicVariable * > &dynObjs) This outputs an XML representation of the object to STDOUT

Member Interpolator::xml_read(XmlReader::xmltag *soundtag)

Member LowPassFilter::xml_print() This outputs an XML representation of the object to STDOUT

Member LPCombFilter::xml_print(ofstream &xmlOutput) This outputs an XML representation of the object to STDOUT

Member LPCombFilter::xml_read(XmlReader::xmltag *lptag)

Member MultiPan::xml_print(ofstream &xmlOutput)

Member Pan::xml_print(ofstream &xmlOutput)

Member Partial::auxLoadParam(enum PartialDynamicParam param, XmlReader::xmltag *tag, hash_map< long, Dy
Auxillary function to assist in loading dv's from XML

Member Partial::xml_print(ofstream &xmlOutput, list< Reverb * > &revObjs, list< DynamicVariable * > &dynObjs
This outputs an XML representation of the object to STDOUT

Member Partial::xml_read(XmlReader::xmltag *partialtag, hash_map< long, Reverb * > *reverbHash, hash_map< l

Member Reverb::xml_print(ofstream &xmlOutput) This outputs an XML representation of the object to STDOUT

Member Reverb::xml_read(XmlReader::xmltag *reverbtag) Used by XML parsing code to reconstruct a Reverb object. This sets the reverb's internal member data and, in general, shouldn't be used anywhere else.

Member Score::xml_print(const char *xmlOutputPath)

Member Score::xml_print(ofstream &xmlOutput)

Member Score::xml_print() This outputs an XML representation of the object to STDOUT

Member Score::xml_read(XmlReader::xmltag *scoretag)

Member Sound::xml_print(ofstream &xmlOutput, list< Reverb * > &revObjs, list< DynamicVariable * > &dynObjs)
This outputs an XML representation of the object to STDOUT

Member Sound::xml_read(XmlReader::xmltag *soundtag, hash_map< long, Reverb * > *reverbHash, hash_map< lo

Member Spatializer::xml_print(ofstream &xmlOutput)

File XmlReader.h Defines the XmlReader object and it's related objects.

10.2 Todo List

Member AuWriter::WriteIntMsb(ostream &out, long l, int size) Recursive inlining? This is insane. This needs to be reworked.

Member Collection::CollectionIterator::next() What to do when out of bounds?

Member Constant::ConstantIterator::next() What to do on an out of bounds error?

Class envelope_segment Once DVS is completely gone and done for, timeType and timeValue should be deleted!

Member Interpolator::addEntry(m_time_type time, m_value_type value) We should have destructor and assignment and copy.

Member InterpolatorIterator::append(m_time_type t_from, m_time_type t_to, m_value_type v_from, m_v
Perhaps for future versions: void append(const InterpolatorIterator&);

Member MultiTrack::MultiTrack(MultiTrack &mt) The argument should be const.

Member MultiTrack::operator=(MultiTrack &mt) The argument should be const.

Class Pan Add destructor to Spatializer.

Member Sound::setPartialParam(PartialDynamicParam p, DynamicVariable &v)
Change FREQUENCY param like old MOSS

Member Track::getAmp() This causes a memory leak!

Index