Syntopia Reference Manual

0.1

Generated by Doxygen 1.2.17

Tue Sep 17 18:01:33 2002

Contents

Chapter 1

Syntopia Documentation

The documentation for 'Syntopia' is still under development. These pages are just preliminary placeholders.

Feel free to browse the documentation, but dont expect anything until a more mature version is released.

Syntopia consists of a number of subprojects, each residing in its own namespace:

- **SynthCore** (p. 14) is all the DSP routines: generators, filters, envelopes... It also contains the VST interface code.
- **GUIToolkit** (p. 11) is an independent Graphical User Interface Toolkit. It contains wrapper classes for a number of standard Windows API components: Buttons, Labels, Comboboxes, menus, sliders and a couple of more advanced components: (including a double buffered canvas class). The event handling part is still very immature.
- **SynthGUI** (p. 17) is the user interface for the synth, built in - well, **GUIToolkit** (p. 11).
- **Utils** (p. 19) is some helper classes: amongst them some string conversion, and a XML library (mainly for serializing C++ objects)

Chapter 2

Syntopia Namespace Index

2.1 Syntopia Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Syntopia Hierarchical Index

3.1 Syntopia Class Hierarchy

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

Chapter 4

Syntopia Compound Index

4.1 Syntopia Compound List

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

Chapter 5

Syntopia Page Index

5.1 Syntopia Related Pages

Here is a list of all related documentation pages:

Chapter 6

Syntopia Namespace Documentation

6.1 GUIToolkit Namespace Reference

Wrapper classes for Windows API GUI calls. GUIToolkit is a simple Graphical User Interface framework, consisting of wrapper classes for standard windows API calls.

Compounds

- class **Button**

  A simple button object.

- class **ComboBox**

  Combo Box. A selectable drop down list.

- class **ComboBoxItem**

  Items for use in a combo box.

- class **Component**

  Base Class for all Components.

- class **DBCanvas**
- class **divider**

  Divider - a simple static graphic component (similar to a ruler in HTML).

- struct **event**

  The event struct passed to the user supplied event handler.

- class **EventHandler**

  Interface for user implemented event handling. The end-user must implement this interface (IE he must supply a class derived from this) with a overridden HandleEvents method.

- class **GUIControl**

  Mandatory class for administrating every GUI.

- class **Label**

  Simple static label.

- class **MCanvas**

  Double-buffered canvas.

- class **menuItem**

  Items to be used on a context menu.

- class **popupMenu**

  A context menu (usually invoked on right mouse button click).

- class **ScrollBar**

  A standard scroll bar.

- class **TextBox**

  Single- og multi-line text box.

- class **WinForm**

  The main windows class. A containter for all Components.

Typedefs

- typedef std::map< **Component** *, int > **ComponentMap**
- typedef std::map< **Component** *, int >::const_iterator **ComponentMapIter**
- typedef std::map< **Component** *, **EventHandler** * > **ComponentEventMap**
- typedef std::map< **Component** *, **EventHandler** * >::const_iterator **ComponentEventMapIter**
- typedef std::map< HWND, **Component** * > **HWNDMap**
- typedef std::map< HWND, **Component** * >::const_iterator **HWNDMapIter**
- typedef std::map< **ComboBoxItem** *, int > **ComboBoxMap**
- typedef std::map< **ComboBoxItem** *, int >::const_iterator **ComboBoxMapIter**

6.1.1 Detailed Description

Wrapper classes for Windows API GUI calls. GUIToolkit is a simple Graphical User Interface framework, consisting of wrapper classes for standard windows API calls.

The GUI as of now supports the following components (derived from the **Component** (p. 35) class):

- **Button** (p. 31) : well...

- Canvas : generic double-buffered drawing component.

- divider : graphical separator component.

- **Label** (p. 62) : for static text output.

- popUpMenu : for context sensitive right mouse button click menus.
- **ComboBox** (p. 32) : Drop-down combo box.
- **ScrollBar** (p. 97) : It is... a scroll bar
- **TextBox** (p. 109) : Single line / multi line text box.

As of now the GUI framework is a minimalistic framework specifically written for the Syntopia project. The event handling is not yet totally encapsulated (in fact it is VERY preliminary) so a certain knowledge of winAPI events are neccesary. But someday it will be nicely integrated in a slick OO fashion....

All components should be placed on a **WinForm** (p. 117) component. In order to capture events the user must supply a class implementing the interface **EventHandler** (p. 52).

A **GUIControl** (p. 58) object must be instantiated before any use of the GUI framework. Example:

```
GUIControl * myGUIControl = new GUIControl(hInstance, nCmdShow);
```

hInstance and nCmdShow can be passed from the WinMain or the DLLMain args.

Example usage of the framework can be found in the namespace **SynthGUI** (p. 17).

6.2 SynthCore Namespace Reference

The main part of the synth framework.

Compounds

- class **BiFilter**

  BiFilter (p. 25) *is a biquadratic filter implementation.*

- class **BiQuad**

  The **BiQuad** (p. 27) *module implements biquadratic filters.*

- class **coord**

  Simple coordinate class.

- class **DelayLine**

  Simple (non-fractional) delay line.

- class **Dummy**

  Benchmarking module.

- class **EnvData**

  Class for storing envelope, and performing interpolation.

- class **Envelope3**

  Envelope3 (p. 48) *is a generic envelope module.*

- class **Fourier**

  Static class for doing **Fourier** (p. 54) *transformations and constructing simple waveforms.*

- class **FreqTable**

  Freqtable converts midinotes to their corresponding frequency in Hz.

- class **Input**

  Input (p. 61) *is a generic input, used in almost every module.*

- class **Module**

  Base class for all modules.

- class **moduleRegistry**

  Register all modules in this class In order to load **Synth** (p. 101) *presets from an XML file and in order to use the GUI the system needs to 'know' about the modules. It needs a pointer to a constructor so that it can create objects from string representations at run-time. Therefore it is neccesary to register using the 'add' method in the registry.*

- class **Multiplier**

  Multiplier (p. 82)*: IE ring-modulator, VCA or amplitude modulator.*

- class **Output**

  Output (p. 85) *is a generic connector endpoint.*

- class **Polyphony**

  Combines the polyphonic parts into one signal.

- class **Sample**

  Sample (p. 91) *Container.*

- class **SampleMap**

  SampleMap (p. 93) *maps midikeys (0-127) to corresponding samples.*

- class **Sampler**

  The main oscillator class.

- class **Synth**

  The main container for the various modules.

- class **SynthEngine**
- class **SynthVoice**
- class **Tube**

  Tube (p. 112) *is a waveguide based experimental delay/reverb kind of effect.*

- class **WaveShaper**

  Aplies a non-linear transformation to input signal.

Typedefs

- typedef std::vector< **coord** * >::const_iterator **pointsIter**

  coord iterator

- typedef std::list< **Module** * > **ModuleMap**
- typedef std::list< **Module** * >::const_iterator **ModuleMapIter**

Functions

- ostream & **operator**<< (ostream &ostr, **Module** &myMod)

  Modules can be streamed to an ostream.

- ostream & **operator**<< (ostream &ostr, **Module** *myMod)

  Modules pointers can also be streamed to an ostream.

6.2.1 Detailed Description

The main part of the synth framework.

The design is centered around the **Synth** (p. 101) class. This class is a container for:

- A number (determined by the polyphony) of SynthVoices, which again host various Modules.

- A number of pre-processing Modules.

A simple synth design would be:

A primary **SynthVoice** (p. 105) (AKA firstVoice) consisting of: A **Sampler** (p. 94) (A voltage controlled oscillator component) connected to an envelope, which again would be connected to an filter (controlled by another envelope).

This **SynthVoice** (p. 105) would be duplicated to the desired number of polyphonic voices. Notice that the duplication is not entirely trivial since the Modules need to share ressources (IE only the **Sampler** (p. 94) in firstVoice would contain the actual wavedata).

The pre-processing modules in the **Synth** (p. 101) unit would consist of a '**Polyphony** (p. 88)' **Module** (p. 75), mixing together the **SynthVoice** (p. 105), and typically an reverb or delay effect (as for now, the tube **Module** (p. 75) would be a good substitute).

The following rules must be followed if new Modules are added.

- All modules must be derived from the **Module** (p. 75) base class.
- They must implement methods for serializing themselves: IE getXML() and loadXML()
- You must use the **Input** (p. 61) and **Output** (p. 85) class for connecting to other Modules.
- They must implement a second static 'constructor', taking a **SynthVoice** (p. 105) as argument! Notice that I cannot enforce this requirement through a purely virtual method, since there is no such thing as a virtual static class in C++. Example: static **Module** (p. 75) * newInstance(SynthVoice * S) { return new BiQuad(S); }
- They must register themselves by having something like this called at run-time: moduleRegistry::getInstance()->add(typeid(BiQuad).name(),**BiQuad::newInstance** (p. 27));

6.2.2 Typedef Documentation

6.2.2.1 typedef std::list< Module * > SynthCore::ModuleMap

Todo:

Why do I have to define these two times: Why can't the compiler use the definitions in "Synth.h"? I dont get it.

Definition at line 93 of file Synth.h.

6.3 SynthGUI Namespace Reference

Graphical User Interface for Syntopia Project. As of now contains classes for manipulation of:

- Envelopes (**EnvDialog** (p. 46))
- Filters (**FilterDialog** (p. 53))
- Oscillator/Sampler (**OscDialog** (p. 84))

and of course the main UI window: **mainDialog** (p. 63).

Compounds

- class **AInput**
  Combined drop box / slider component.
- class **EnvDialog**
  Class for manipulating the Envelope module.
- class **FilterDialog**
  Class for manipulating the Filter module.
- class **mainDialog**
  The main UI window.
- class **mod**
  mod is the class for visually representing a module.
- class **modInOuts**
  Class for graphical representing in and outputs.
- class **modList**
  Containter for the 'mods' displayed on the main UI window.
- class **OscDialog**
  Dialog window for manipulating Oscillator/Sampler setting.
- class **SlideInput**
  A combined slider and text box input component.

Typedefs

- typedef std::vector< Module * > **ModVector**

6.3.1 Detailed Description

Graphical User Interface for Syntopia Project. As of now contains classes for manipulation of:

- Envelopes (**EnvDialog** (p. 46))

- Filters (**FilterDialog** (p. 53))
- Oscillator/Sampler (**OscDialog** (p. 84))

and of course the main UI window: **mainDialog** (p. 63).

6.3.2 Typedef Documentation

6.3.2.1 typedef std::vector<Module *> SynthGUI::ModVector

If the module is part of a voice (for polyphonic synths) One mod can point to many instances (but of course only one of these: the so called firstVoice is drawn). This is (the type of) the list of modules.

Definition at line 46 of file MainDialog.h.

6.4 Utils Namespace Reference

Conversion and XML routines. Utils consists of simple string conversions routines and a small XML parsing library.

Compounds

- class **AttributeNode**
  Used to represent XML Attributes.
- class **parseError**
  Exception thrown by XMLDoc.parser.
- class **TagNode**
  Used to represent a tag node (I.E.: <html>) .
- class **TextNode**
  Used to represent text nodes.
- class **XMLDoc**
  Class for representing a XML document.
- class **XMLNode**
  The base class for all XMLNodes.
- class **XMLSearch**

Functions

- double **Log2** (double x)
  Returns base 2 logarithm (log2 (x) = log10(x) / log10(2)).
- void **trim** (string &S)
  Trims leading and trailing spaces.
- std::string **newline** ()
- std::string **space** (int indent)
  Returns 'indent' spaces.
- std::string **floatToString** (float f)
  Converts a float to a C++ std::string.
- std::string **intToString** (int i)
  Converts a int to a C++ std::string.
- float **stringToFloat** (string s)
  Converts a C++ std::string to a float.
- int **stringToInt** (string s)
  Converts a C++ std::string to an int.

6.4.1 Detailed Description

Conversion and XML routines. Utils consists of simple string conversions routines and a small XML parsing library.

The XML library is a DOM (document object model) parsing library. It is not very efficient and should be used for small files (<2MB) only.

A simple search routine for traversing the DOM tree is provided. (See **XMLDoc** (p. 119) for example usage).

6.4.2 Function Documentation

6.4.2.1 string Utils::newline ()

Returns a newline For the time being a CR / NL (WIN 32 standard) todo Should be platform independent.

Definition at line 46 of file Utils.cpp.

Referenced by SynthCore::WaveShaper::getXML(), Utils::TextNode::getXML(), Utils::TagNode::getXML(), SynthCore::Tube::getXML(), SynthCore::Synth::getXML(), SynthCore::Sampler::getXML(), SynthCore::Polyphony::getXML(), SynthCore::Module::getXML(), SynthCore::Output::getXML(), SynthCore::Input::getXML(), SynthCore::BiQuad::getXML(), SynthCore::Envelope3::getXML(), and Utils::XMLDoc::parseFile().

Chapter 7

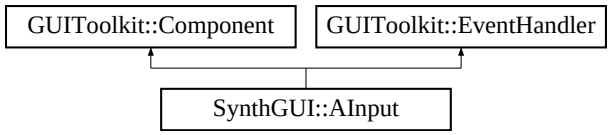
Syntopia Class Documentation

7.1 SynthGUI::AInput Class Reference

Combined drop box / slider component.

`#include <AInput.h>`

Inheritance diagram for SynthGUI::AInput::



Public Methods

- **AInput** (std::string s, float min, float max, int steps, bool integral)
- **~AInput** ()

  Destructor.

- void **PutInWinForm** (WinForm *WF, int x0, int x1, int y0, int y1)

  Places component on winform.

- void **HandleEvents** (event ev)

  Call this to set eventhandler. (Lot of 'background' code in this!).

- float **getValue** ()

  Return current value.

- void **setValue** (float f)

  Sets current value.

7.1.1 Detailed Description

Combined drop box / slider component.

Special UI component. A drop down box allows the user to choose between different input methods: drop down, slider or text box input Used to select mappable parameters in the Synth GUI.

Definition at line 39 of file AInput.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 SynthGUI::AInput::AInput (std::string *s*, float *min*, float *max*, int *steps*, bool *integral*)

's' is the caption of the module. The allowed interval is [min;max] quantized in 'steps' steps. 'integral' can be set to true, if only integral values are allowed.

Definition at line 12 of file AInput.cpp.

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

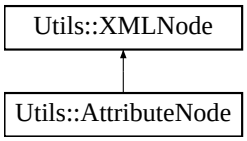
- AInput.h
- AInput.cpp

7.2 Utils::AttributeNode Class Reference

Used to represent XML Attributes.

`#include <Utils.h>`

Inheritance diagram for Utils::AttributeNode::



Public Methods

- **AttributeNode** ()
  Constructor.
- **AttributeNode** (string name, string data)
  Init node with text = s.
- ~**AttributeNode** ()
  Destructor.
- string **printString** ()
  Debug info : name , pointer, path.
- string **getXML** (int indent)
  Used when constructing a XML text representation.
- string **pathName** ()
  Used in **path***()* (p. 121) *to print path.*

Public Attributes

- string **attributeName**
  The name of the attribute (IE: <TagName AttributeName="AttributeData">).
- string **attributeData**
  The value of the attribute (IE: <TagName AttributeName="AttributeData">).

7.2.1 Detailed Description

Used to represent XML Attributes.

Definition at line 193 of file Utils.h.

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

- Utils.h
- Utils.cpp

7.3 SynthCore::BiFilter Class Reference

BiFilter (p. 25) is a biquadratic filter implementation.

`#include <sndutils.h>`

Public Methods

- **BiFilter** ()
  Constructor.
- ~**BiFilter** ()
  Destructor;.
- float **read** (float input)
  Actual Code;.
- void **bypassFreqAndQ** (float freq, float Q)
  Dummy (p. 41) *filter. Does not alter input.*
- void **lowpassFreqAndQ** (float freq, float Q)
  Calculates Low Pass filter coefficients.
- void **notchFreqAndQ** (float freq, float Q)
  Calculates notch (a steep valley) filter coefficients.
- void **bandpass2FreqAndQ** (float freq, float Q)
  Calculates bandpass (constant 0 dB peak gain) filter coefficients.
- void **bandpass1FreqAndQ** (float freq, float Q)
  Calculates Low Pass (constant skirt gain, peak gain = Q) filter coefficients.
- void **highpassFreqAndQ** (float freq, float Q)
  Calculates High Pass filter coefficients.
- void **allpassFreqAndQ** (float freq, float Q)
  Calculates all pass filter coefficients.
- void **highShelfFreqAndQAndA** (float freq, float Q, float A)
  Calculates high shelf filter coefficients.
- void **lowShelfFreqAndQAndA** (float freq, float Q, float A)
  Calculates low shelf filter coefficients.
- void **peakingEQFreqAndQAndA** (float freq, float Q, float A)
  Calculates peaking EQ filter coefficients.
- float **getFreqResponse** (float wT)
  calculates the frequency response for wT = (frequency∗sampleLength). **Fourier** (p. 54) *transforms filter co-efficients.*

- float **getPhaseResponse** (float wT)

  calculates the phase response for wT = (frequency∗sampleLength). **Fourier** (p. 54) *transforms filter co-efficients.*

- float **getFreqResponse2** (float wT)

  calculates the frequency response for wT = (frequency∗sampleLength). Uses 'brute force' method (read: this is not efficient- use **getFreqResponse***()* (p. 25) *instead)*

- void **getPolesAndZeroes** (float &p1r, float &p1i, float &p2r, float &p2i, float &z1r, float &z1i, float &z2r, float &z2i)

  Return the poles and Zeroes of the filter. All calling parameters are just return values passed by reference.

- void **clone** (BiFilter ∗from)
- **Sample** ∗ **getFreqResponseSample** (int resolution)

  Returns total frequency response in a sample.

- **Sample** ∗ **getPhaseResponseSample** (int resolution)

  Returns total phase response in a sample.

7.3.1 Detailed Description

BiFilter (p. 25) is a biquadratic filter implementation.

BiFilter (p. 25) is a biquadratic filter implementation. Thus allowing up to two poles and zeroes in the Z-plane transform of the transfer-function.

BiFilter (p. 25) is used in the **BiQuad** (p. 27) module.

Definition at line 69 of file sndutils.h.

7.3.2 Member Function Documentation

7.3.2.1 void SynthCore::BiFilter::clone (BiFilter ∗ *from*)

This copies the coefficients from another filter. Useful for filters in series, since there is no reason to calcalute coefficients twice.

Definition at line 314 of file sndutils.cpp.

References a0, a1, a2, b0, b1, and b2.

Referenced by SynthCore::BiQuad::updateFreqAndRez().

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

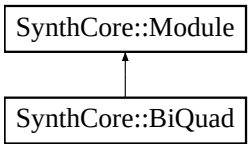
- sndutils.h
- sndutils.cpp

7.4 SynthCore::BiQuad Class Reference

The **BiQuad** (p. 27) module implements biquadratic filters.

`#include <filters.h>`

Inheritance diagram for SynthCore::BiQuad::



Public Types

- enum { **lowpass**, **highpass**, **notch**, **bandpass2**, **bandpass1**, **allpass**, **highShelf**, **lowShelf**, **peakingEQ**, **bypass** }

Public Methods

- virtual std::string **getName** ()
  The name.
- virtual **Module** * **sharedClone** ()
- **BiQuad** (**SynthVoice** *S1)
  Constructor.
- ~**BiQuad** ()
  Destructor;.
- void **update** ()
  Main routine;.
- void **setDry** (float s)
  Dry/Wet percentage.
- void **setFreqAndRez** (float frequence, float resonance)
  Notice resonance only is a resonance for low and high pass filters.
- void **updateFreqAndRez** ()
- std::string **getXML** (int indent=0)
- void **loadXML** (XMLNode *n, bool firstPass)
  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

Static Public Methods

- **Module** * **newInstance** (**SynthVoice** *S1)
  Static constructor (for 'factory' use).

Public Attributes

- bool **HACK**
  HACK.
- float **rez**
  Current resonance (Q-parameter).
- float **freq**
  Current frequency.
- int **filterType**
  Current filter type.
- float **dry**
  Determines throughput (dryness). Obsolete?
- **Output** ∗ **Output1**
  Mono output.
- **Input** ∗ **Input1**
  Mono input.
- **Input** ∗ **FreqInput**
  Frequency **Input** (p. 61).
- **Input** ∗ **RezInput**
- **BiFilter** ∗ **myFilter**
  Filter 1. (for 12 db/Octave).
- **BiFilter** ∗ **myFilter2**
  Filter 2. (for another 12 db/octave).

7.4.1 Detailed Description

The **BiQuad** (p. 27) module implements biquadratic filters.

Thus allowing up to two poles and zeroes in the Z-plane transform of the transfer-function.

Definition at line 32 of file filters.h.

7.4.2 Member Function Documentation

7.4.2.1 std::string SynthCore::BiQuad::getXML (int *indent* = 0) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 74 of file filter.cpp.

References dry, freq, FreqInput, SynthCore::Input::getXML(), SynthCore::Output::getXML(), Input1, Utils::intToString(), Utils::newline(), Output1, rez, RezInput, and Utils::space().

7.4.2.2 void SynthCore::BiQuad::setDry (float *s*)

Dry/Wet percentage.

Todo:
consider the soundness of this.

Definition at line 69 of file filter.cpp.

References dry.

7.4.2.3 Module * SynthCore::BiQuad::sharedClone () [virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented from **SynthCore::Module** (p. 78).

Definition at line 20 of file filter.cpp.

References BiQuad(), dry, filterType, SynthCore::Module::firstVoice, freq, SynthCore::Module::hostVoice, SynthCore::Module::ModID, and rez.

7.4.2.4 void SynthCore::BiQuad::updateFreqAndRez ()

The filter coefficients are not calcuted every samplestep (That would be to time consuming) Call this method to update coefficients.

Definition at line 132 of file filter.cpp.

References SynthCore::BiFilter::allpassFreqAndQ(), SynthCore::BiFilter::bandpass1FreqAndQ(), SynthCore::BiFilter::bandpass2FreqAndQ(), SynthCore::BiFilter::bypassFreqAndQ(), SynthCore::BiFilter::clone(), freq, SynthCore::BiFilter::highpassFreqAndQ(), SynthCore::BiFilter::highShelfFreqAndQAndA(), SynthCore::BiFilter::lowpassFreqAndQ(), SynthCore::BiFilter::lowShelfFreqAndQAndA(), myFilter, myFilter2, SynthCore::BiFilter::notchFreqAndQ(), SynthCore::BiFilter::peakingEQFreqAndQAndA(), and rez.

Referenced by BiQuad(), setFreqAndRez(), and update().

7.4.3 Member Data Documentation

7.4.3.1 bool SynthCore::BiQuad::HACK

HACK.

Todo:
clean this

Definition at line 43 of file filters.h.

Referenced by BiQuad(), and update().

7.4.3.2 Input* SynthCore::BiQuad::RezInput

Resonance (that is: Q factor) **Input** (p. 61). This number is just a parameter used in determing the filter coefficients. For low and high pass filters it is a resonance factor.

Definition at line 73 of file filters.h.

Referenced by BiQuad(), getXML(), loadXML(), update(), and ~BiQuad().

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

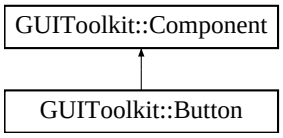
- filters.h
- filter.cpp

7.5 GUIToolkit::Button Class Reference

A simple button object.

`#include <Button.h>`

Inheritance diagram for GUIToolkit::Button::



Public Methods

- **Button** (std::string s)
  Creates a button with the caption s.
- **~Button** ()
  Clean up.
- void **PutInWinForm** (**WinForm** *WF, int x0, int x1, int y0, int y1)
  Places the button on the winform.

7.5.1 Detailed Description

A simple button object.

Definition at line 24 of file Button.h.

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

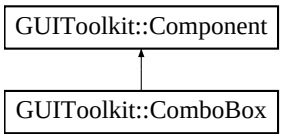
- Button.h
- Button.cpp

7.6 GUIToolkit::ComboBox Class Reference

Combo Box. A selectable drop down list.

`#include <ListBox.h>`

Inheritance diagram for GUIToolkit::ComboBox::



Public Methods

- **ComboBox** ()
  Constructor.
- ∼**ComboBox** ()
  Destructor.
- void **PutInWinForm** (**WinForm** ∗WF, int x0, int y0, int sizex, int sizey)
  Place combobox on **WinForm** (p. 117).
- void **add** (**ComboBoxItem** ∗item)
  Puts 'item' on **ComboBox** (p. 32).
- void **setSelected** (**ComboBoxItem** ∗item)
  Sets the selected item to 'item'.
- void **setSelected** (int index)
  Sets the selected item to index.
- void **show** ()
  Show **ComboBox** (p. 32).
- void **hide** ()
  Hides **ComboBox** (p. 32).
- **ComboBoxItem** ∗ **getSelected** ()
  Returns currently selected item.

7.6.1 Detailed Description

Combo Box. A selectable drop down list.

Definition at line 45 of file ListBox.h.

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

- ListBox.h
- Listbox.cpp

7.7 GUIToolkit::ComboBoxItem Class Reference

Items for use in a combo box.

`#include <ListBox.h>`

Public Methods

- **ComboBoxItem** (std::string s)
  Constructor. 's' is the caption of the item.
- **~ComboBoxItem** ()
  Destructor.

Public Attributes

- std::string **Label**
  Name of **ComboBox** (p. 32) *item.*

Friends

- class **ComboBox**
  ComboBox (p. 32) *may access private members.*

7.7.1 Detailed Description

Items for use in a combo box.

Definition at line 18 of file ListBox.h.

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

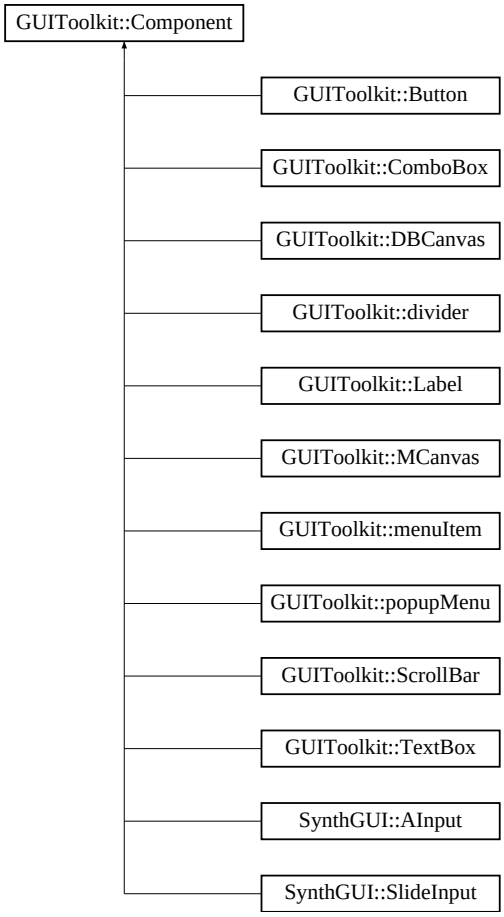
- ListBox.h

7.8 GUIToolkit::Component Class Reference

Base Class for all Components.

`#include <Component.h>`

Inheritance diagram for GUIToolkit::Component::



Public Methods

- **Component** ()

  Constructor.

- virtual ~**Component** ()

  Destructor.

- virtual void **HandleEvents** (**event** e)

  If an user eventhandler is associated with the component, it will be called.

- virtual void **setEventHandler** (**EventHandler** *eh)

  Specify an user event handler.

- virtual HWND **getHWND** ()

  Returns window-handle.

- virtual void **draw** ()

  For components not capable of redrawing themselves.

Protected Attributes

- **EventHandler** ∗ **myEventHandler**
- HMENU **ID**
- **WinForm** ∗ **topWF**

7.8.1 Detailed Description

Base Class for all Components.

Definition at line 25 of file Component.h.

7.8.2 Member Function Documentation

7.8.2.1 virtual void GUIToolkit::Component::draw () [inline, virtual]

For components not capable of redrawing themselves.

Todo:
what uses this?

Reimplemented in **GUIToolkit::MCanvas** (p. 65).

Definition at line 47 of file Component.h.

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

- Component.h
- Component.cpp

7.9 SynthCore::coord Class Reference

Simple coordinate class.

`#include <envelope.h>`

Public Methods

- **coord** ()
- **coord** (float mx, float my)

Public Attributes

- float **x**
- float **y**

7.9.1 Detailed Description

Simple coordinate class.

Definition at line 25 of file envelope.h.

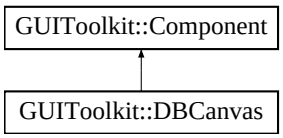
The documentation for this class was generated from the following file:

- envelope.h

7.10 GUIToolkit::DBCanvas Class Reference

`#include <WinForm.h>`

Inheritance diagram for GUIToolkit::DBCanvas::



Public Methods

- **DBCanvas** (**WinForm** *WF, int _x0, int _y0, int _x1, int _y1)
- ~**DBCanvas** ()
- void **Paint** (HDC hdc)
- void **Rectangle** (int x0, int y0, int x1, int y1)
- void **setPixel** (int x, int y, int r, int g, int b)
- void **clear** ()
- void **update** ()

Public Attributes

- int **x0**
- int **y0**
- int **x1**
- int **y1**

7.10.1 Detailed Description

Todo:
Obsolete?

Definition at line 67 of file WinForm.h.

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

- WinForm.h
- WinForm.cpp

7.11 SynthCore::DelayLine Class Reference

Simple (non-fractional) delay line.

`#include <sndutils.h>`

Public Methods

- float **read** (float input)

  Returns value read from delayLine. Stores new input in delay line. Advances read/write pointer.

- void **setDelay** (int D)

  Set length of delay in samples.

- **DelayLine** (int Length)

  Constructor.

- **~DelayLine** ()

  Destructor.

- float **readNBack** (int N)

  Read N samples back in buffer. Does not change read/write pointer.

- float **readNForward** (int N)

  Read N samples forward in buffer. Does not change read/write pointer.

Public Attributes

- float * **bufferStart**
- float * **bufferEnd**
- float * **readPtr**
- float **output1**
- float * **temp**
- int **bufferLength**

  Length of buffer.

7.11.1 Detailed Description

Simple (non-fractional) delay line.

Simple (non-fractional) delay line. Used as a building block in tubes and reverb.

Definition at line 28 of file sndutils.h.

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

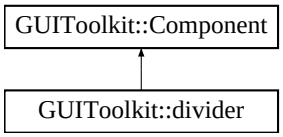
- sndutils.h
- sndutils.cpp

7.12 GUIToolkit::divider Class Reference

Divider - a simple static graphic component (similar to a ruler in HTML).

`#include <divider.h>`

Inheritance diagram for GUIToolkit::divider::



Public Methods

- **divider** ()
  Constructor.
- **~divider** ()
  Destructor.
- void **PutInWinForm** (**WinForm** *WF, int x0, int x1, int y0, int y1)
  Places divider on **WinForm** (p. 117).

7.12.1 Detailed Description

Divider - a simple static graphic component (similar to a ruler in HTML).

Definition at line 24 of file divider.h.

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

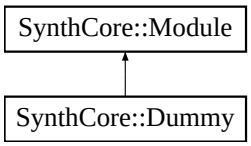
- divider.h
- divider.cpp

7.13 SynthCore::Dummy Class Reference

Benchmarking module.

`#include <Modules.h>`

Inheritance diagram for SynthCore::Dummy::



Public Methods

- virtual std::string **getName** ()
  The name is just a virtual placeholder.
- **Dummy** (**SynthVoice** *S1)
- ~**Dummy** ()
- void **update** ()
- void **dupdate** ()

Public Attributes

- **Output** * **Output1**
- **Input** * **Input1**
- float * **In1**
- float * **Ou1**
- **Output** * **Out**

7.13.1 Detailed Description

Benchmarking module.

Dummy (p. 41) module. Used for benchmarking the overhead of the framework.

Definition at line 262 of file Modules.h.

7.13.2 Member Function Documentation

7.13.2.1 void SynthCore::Dummy::update () `[virtual]`

update() (p. 41) is where the main sound processing in a module takes place. Called every sample-step. It is purely virtual

Implements **SynthCore::Module** (p. 78).

Definition at line 101 of file Modules.cpp.

References SynthCore::Input::ConFrom, and SynthCore::Output::value.

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

- Modules.h
- Modules.cpp

7.14 SynthCore::EnvData Class Reference

Class for storing envelope, and performing interpolation.

`#include <envelope.h>`

Public Types

- enum **loopStateEnum** { **dontLoop**, **sustainedLoop**, **alwaysLoop** }

  The different states. dontLoop = never loops. sustainedLoop = loops as long key is held. alwaysLoop = loops no matter what.

Public Methods

- void **clearEnvelope** ()

  Clears the Envelope.

- void **setADSR** (float A, float D, float S, float R)

  Creates an factory envelope with attack time A, decay time D, sustain level S (between 0 and 1), and Release time R;.

- void **setChoosen** (int j)
- int **getChoosen** ()
- void **getLimits** (float &x0, float &y0, float &x1, float &y1)
- void **interpolateLinear** ()
- void **interpolateSpline** ()

  Cubic spline interpolation.

- void **interpolate** ()

  Calls the chosen interpolation method.

- void **removePoint** (int p)
- void **addPoint** (float x, float y)
- float **getValue** (long &time, bool loop)
- std::string **getXML** (int indent)
- **EnvData** ()
- ~**EnvData** ()

Public Attributes

- bool **endOfEnvelope**

  This flag is set when end of data is reached for the first time.

- std::vector< **coord** * > **points**

  The anchor-points.

- float * **envdata**

  Interpolated float data.

- float **sampleLength**
  The interpolated array corresponds to this sample length.
- int **envsize**
  The length of the interpolated array.
- int **beginLP**
  Loop-region. the value is an index of 'points'-vector.
- int **endLP**
  Loop-region. the value is an index of 'points'-vector.
- int **loopStart**
  Loop-region: the value is an index of 'envdata'-array.
- int **loopEnd**
  Loop-region: the value is an index of 'envdata'-array.
- **loopStateEnum loopState**
  Defines the loop properties of the envelope.
- int **choosen**
  Set to index of graphically selected anchor point.

7.14.1 Detailed Description

Class for storing envelope, and performing interpolation.

EnvData (p. 43) contains a vector ('points') which defines a number anchor-points in the envelope.

The interpolation routines converts these points into a float array ('envdata')

Todo:
The loop point is unused for now.

Definition at line 47 of file envelope.h.

7.14.2 Member Function Documentation

7.14.2.1 float SynthCore::EnvData::getValue (long & *time*, bool *loop*) [inline]

This is what you want to call in order to read the envelope data.

Todo:
This also depends on an interpolation method. Should be fixed.

Definition at line 246 of file envelope.cpp.

References endOfEnvelope, envdata, envsize, loopEnd, loopStart, and sampleLength.

Referenced by SynthCore::Envelope3::update().

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

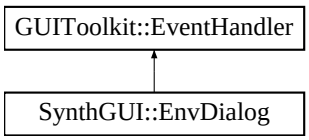
- envelope.h
- envelope.cpp

7.15 SynthGUI::EnvDialog Class Reference

Class for manipulating the Envelope module.

`#include <envDialog.h>`

Inheritance diagram for SynthGUI::EnvDialog::



Public Methods

- **EnvDialog** (**mod** *md)
  Create the dialog for mod 'md'.
- **~EnvDialog** ()
  Destructor.
- void **drawEnv** ()
  Updates envelope graph.
- void **MessageLoop** ()
  Enter infinite loop.
- void **HandleEvents** (event ev)
  Private event handler (called by components on dialog window).
- void **EnvToScreen** (float xx, float yy, int &x, int &y)
  Converts from Envelope coordinate to Screen coordinates.
- void **ScreenToEnv** (int x, int y, float &xx, float &yy)
  Converts from Screen coordinates to Envelope coordinates.

7.15.1 Detailed Description

Class for manipulating the Envelope module.

Definition at line 29 of file envDialog.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 SynthGUI::EnvDialog::~EnvDialog ()

Destructor.

Todo:
create this

Definition at line 222 of file envDialog.cpp.

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

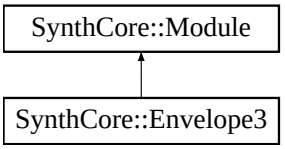
- envDialog.h
- envDialog.cpp

7.16 SynthCore::Envelope3 Class Reference

Envelope3 (p. 48) is a generic envelope module.

`#include <envelope.h>`

Inheritance diagram for SynthCore::Envelope3::



Public Methods

- virtual std::string **getName** ()

  This is the name of the unit.

- void **setDuration** (int calls)

  Sets the duration of the envelope measured in CALLS to the update function.

- virtual **Module** * **sharedClone** ()
- **Envelope3** (**SynthVoice** *s)

  Constructor.

- ~**Envelope3** ()

  Destructor;.

- void **update** ()

  Actual Code takes place here;.

- void **loadXML** (XMLNode *n, bool firstPass)

  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

- std::string **getXML** (int indent)

Static Public Methods

- **Module** * **newInstance** (**SynthVoice** *s)

  Static constructor (for 'factory' use).

Public Attributes

- bool **killVoice**
- **EnvData** * **myData**

  The actual envelope.

- **Output** * **Output1**

Mono **Output** (p. 85).

- **Input** ∗ **Input1**
  Mono **Input** (p. 61).

7.16.1 Detailed Description

Envelope3 (p. 48) is a generic envelope module.

The envelope can be arbitrarily complex.

If the input is connected, the output will be: **Output** (p. 85) = **Input** (p. 61) ∗ envelope otherwise it will be, **Output** (p. 85) = envelope.

Todo:
check the two modes.

Definition at line 129 of file envelope.h.

7.16.2 Member Function Documentation

7.16.2.1 std::string SynthCore::Envelope3::getXML (int *indent*) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 55 of file envelope.cpp.

References SynthCore::Input::getXML(), SynthCore::Output::getXML(), SynthCore::EnvData::getXML(), Input1, Utils::intToString(), killVoice, myData, Utils::newline(), Output1, and Utils::space().

7.16.2.2 Module ∗ SynthCore::Envelope3::sharedClone () [virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented from **SynthCore::Module** (p. 78).

Definition at line 31 of file envelope.cpp.

References Envelope3(), SynthCore::Module::firstVoice, SynthCore::Module::hostVoice, killVoice, SynthCore::Module::ModID, and myData.

7.16.3 Member Data Documentation

7.16.3.1 bool SynthCore::Envelope3::killVoice

Set this to true, if you want the Envelope to kill the current voice, when the end of the envelope is reached. Notice that at least one envelope must kill the voice in order not to use excessive CPU power.

Definition at line 138 of file envelope.h.

Referenced by Envelope3(), getXML(), loadXML(), sharedClone(), and SynthCore::SynthVoice::SynthVoice().

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

- envelope.h
- envelope.cpp

7.17 GUIToolkit::event Struct Reference

The event struct passed to the user supplied event handler.

`#include <Events.h>`

Public Attributes

- HWND **hwnd**
- UINT **msg**
- WPARAM **wParam**
- LPARAM **lParam**
- **Component** * **EventSource**

7.17.1 Detailed Description

The event struct passed to the user supplied event handler.

Definition at line 26 of file Events.h.

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

- Events.h

7.18 GUIToolkit::EventHandler Class Reference

Interface for user implemented event handling. The end-user must implement this interface (IE he must supply a class derived from this) with a overridden HandleEvents method.

`#include <Events.h>`

Inheritance diagram for GUIToolkit::EventHandler::

Public Methods

- virtual void **HandleEvents** (**event** ev)

7.18.1 Detailed Description

Interface for user implemented event handling. The end-user must implement this interface (IE he must supply a class derived from this) with a overridden HandleEvents method.

Todo:
should it be purely virtual?

Definition at line 38 of file Events.h.

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

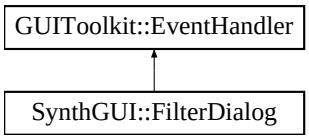
- Events.h

7.19 SynthGUI::FilterDialog Class Reference

Class for manipulating the Filter module.

`#include <FilterDialog.h>`

Inheritance diagram for SynthGUI::FilterDialog::



Public Methods

- **FilterDialog** (**mod** *moduleptr)
  Takes pointer to mod structure.
- **~FilterDialog** ()
  Destructor.
- void **MessageLoop** ()
  Enter infinite loop.
- void **HandleEvents** (event ev)
  Private message handler.

7.19.1 Detailed Description

Class for manipulating the Filter module.

Definition at line 27 of file FilterDialog.h.

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

- FilterDialog.h
- FilterDialog.cpp

7.20 SynthCore::Fourier Class Reference

Static class for doing **Fourier** (p. 54) transformations and constructing simple waveforms.

`#include <sndutils.h>`

Static Public Methods

- void **transform** (float data[], unsigned long nn, int isign)
- **Sample** * **sawtooth** (int sampleLength, int partials, int wrapAround)

  Creates Sawtooth with given number of partials. 'wrapAround' samples are duplicated at the end.

- **Sample** * **square** (int sampleLength, int partials, int wrapAround)

  Creates square with given number of partials. 'wrapAround' samples are duplicated at the end.

- **Sample** * **triangle** (int sampleLength, int partials, int wrapAround)

  Creates triangle with given number of partials. 'wrapAround' samples are duplicated at the end.

- **Sample** * **sine** (int sampleLength, int partials, int wrapAround)
- **Sample** * **wave1** (int sampleLength, int partials, int wrapAround)
- **Sample** * **wave2** (int sampleLength, int partials, int wrapAround)
- **Sample** * **wave3** (int sampleLength, int partials, int wrapAround)
- **Sample** * **wave4** (int sampleLength, int partials, int wrapAround)

7.20.1 Detailed Description

Static class for doing **Fourier** (p. 54) transformations and constructing simple waveforms.

Definition at line 248 of file sndutils.h.

7.20.2 Member Function Documentation

7.20.2.1 Sample * SynthCore::Fourier::sine (int *sampleLength*, int *partials*, int *wrapAround*) [static]

Creates triangle with given number of partials. 'wrapAround' samples are duplicated at the end. Its offcourse overkill to do it this way, but it ment for reference purposes.

Definition at line 666 of file sndutils.cpp.

References SynthCore::Sample::makeWrap(), SynthCore::Sample::samples, and SynthCore::Sample::wavenormalize().

7.20.2.2 void SynthCore::Fourier::transform (float *data*[], unsigned long *nn*, int *isign*) [static]

Fourier (p. 54) transforms data[]. The result is returned in data. isign = 1 for fouriertransform, isign = -1 for inverse fourier transform The length of the data must be a power of 2 (IE: 1024, 2048, ...) Based on implementation in numerical recipes. Implementation changed!

Todo:
clean up wrappers.

Definition at line 527 of file sndutils.cpp.

References Utils::Log2().

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

- sndutils.h
- sndutils.cpp

7.21 SynthCore::FreqTable Class Reference

Freqtable converts midinotes to their corresponding frequency in Hz.

`#include <sndutils.h>`

Public Methods

- **FreqTable** ()
- ~**FreqTable** ()

  Destructor.

- int **getMaxPartialsFromFreq** (float freq)
- int **getMaxPartialsFromMidi** (int midikey)

  Returns max number of partials from a given midikey.

Public Attributes

- float **freqTab** [128]

  The table itself.

7.21.1 Detailed Description

Freqtable converts midinotes to their corresponding frequency in Hz.

Definition at line 282 of file sndutils.h.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 SynthCore::FreqTable::FreqTable () [inline]

The Counstructor calculates to table 440 - 220 - 110 - 55 - 27.5 - 13.75 a-2 has freq 6.875 hz. midinote 0 c-2 Midi Begin 21 a0 27.5000 1745 Key begin 41 f2 512 69 a4 440.0 109 Center Key 108 c8 4186.0 11 Key End 127

Definition at line 299 of file sndutils.h.

References freqTab.

7.21.3 Member Function Documentation

7.21.3.1 int SynthCore::FreqTable::getMaxPartialsFromFreq (float *freq*) [inline]

Returns max number of partials from a given frequency

Todo:
Only correct at 48KHz!

Definition at line 318 of file sndutils.h.

Referenced by getMaxPartialsFromMidi().

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

- sndutils.h

7.22 GUIToolkit::GUIControl Class Reference

Mandatory class for administrating every GUI.

`#include <GUIControl.h>`

Public Methods

- **GUIControl** (HINSTANCE my_hInstance, int my_nCmdShow)
- ~**GUIControl** ()

  Destructor.

- void **MessageLoop** ()

  Call this infinite loop, when UI is setup.

- int **Register** (**Component** *m)

  Every component must register. Done in **Component** (p. 35) *Constructor.*

- std::string **getUniqueName** ()

  To generate a new Windows class a unique name is needed.

- void **RegisterEvents** (**Component** *m, **EventHandler** *e)

  Set **EventHandler** (p. 52) *to* **Component** (p. 35).

- int **RegisterRaw** (**Component** *m)
- void **RegisterManualDraw** (HWND hwnd, **Component** *m)

  @Is this used?

- void **DEBUG** (std::string t)

  Prints to ErrorTextBox;.

- void **DEBUG** (char t[])

  Prints to ErrorTextBox;.

- void **DEBUG** (int t)

  Prints to ErrorTextBox;.

Static Public Methods

- LRESULT CALLBACK **WndProc** (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

  A Static callback routine. Used by windows event handling.

- std::string **my_itos** (int i)

  Conversion.

Public Attributes

- MSG **Msg**
- HINSTANCE **hInstance**

  Passed from WINMAIN.

- int **nCmdShow**

  Passed from WINMAIN.

- HWND **firstWindow**

  When 'firstWindow' is closed, the application terminates.

- **TextBox** ∗ **ErrorTextBox**

  Windows programs usually dont emply a console. Use this for error reporting in conjunction with the DEBUG routines.

Static Public Attributes

- GUIControl ∗ **GUIInstance** = 0

7.22.1 Detailed Description

Mandatory class for administrating every GUI.

The **GUIControl** (p. 58) takes care of eventhandling and keeps track of UI components. Always start by creating an instance of this class.

Definition at line 27 of file GUIControl.h.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 GUIToolkit::GUIControl::GUIControl (HINSTANCE *my_hInstance*, int *my_nCmdShow*)

Pass parameters from WinMain or DLLMain. If these parameters are not available call with my_nCmdShow = SW_SHOWDEFAULT and my_hInstance = GetModuleHandle(NULL). (This could be critical under non XP-environments)

Todo:
add apropriate default constructor.

Definition at line 26 of file GUIControl.cpp.

References firstWindow, GUIInstance, hInstance, and nCmdShow.

7.22.3 Member Function Documentation

7.22.3.1 std::string GUIToolkit::GUIControl::my_itos (int *i*) [static]

Conversion.

Todo:
Obsolete.

Definition at line 19 of file GUIControl.cpp.

Referenced by DEBUG().

7.22.3.2 int GUIToolkit::GUIControl::RegisterRaw (Component * m)

Raw components (that is, DirectX component) must call this

Todo:
is this the way?

Definition at line 219 of file GUIControl.cpp.

7.22.4 Member Data Documentation

7.22.4.1 GUIControl * GUIToolkit::GUIControl::GUIInstance = 0 [static]

We need a static callback routine in order to satisfy windows. But we also need a nice object, that the static routine can access. Therefore we have a static pointer to a dynamic object. This means there can be only ONE instance of **GUIControl** (p. 58) (since it accessed through GUIInstance) Awfull Hack, but neccesary.

Definition at line 17 of file GUIControl.cpp.

Referenced by GUIControl(), and WndProc().

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

- GUIControl.h
- GUIControl.cpp

7.23 SynthCore::Input Class Reference

Input (p. 61) is a generic input, used in almost every module.

`#include <Input.h>`

Public Methods

- **Input** (**Module** *host)
  Constructor with pointer to host module.
- **~Input** ()
  Destructor.
- float **read** ()
  Should be used to read from input.
- void **ConnectFrom** (**Output** *O)
  Always use this to connect the input to a output.
- float **updateRead** ()
  This just updates host, and reads from the host output.
- void **loadXML** (string tagName, XMLNode *n)
  Load values from XMLNode.
- std::string **getXML** (std::string tag, int indent=0)
  Save state to XML string.

Public Attributes

- **Output** * **ConFrom**
  Connected from this output.
- **Module** * **HostModule**
  We live here (set by constructor, but can be changed public):.
- bool **manualUpdate**

7.23.1 Detailed Description

Input (p. 61) is a generic input, used in almost every module.

Definition at line 23 of file Input.h.

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

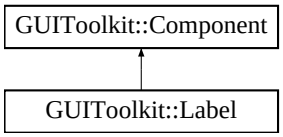
- Input.h
- Input.cpp

7.24 GUIToolkit::Label Class Reference

Simple static label.

`#include <Label.h>`

Inheritance diagram for GUIToolkit::Label::



Public Methods

- **Label** (std::string s)
  Constructor: 's' is the caption of the label.
- **~Label** ()
  Destructor.
- void **PutInWinForm** (**WinForm** *WF, int x0, int x1, int y0, int y1)
  Put label on winform.

7.24.1 Detailed Description

Simple static label.

Definition at line 21 of file Label.h.

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

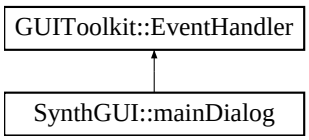
- Label.h
- Label.cpp

7.25 SynthGUI::mainDialog Class Reference

The main UI window.

`#include <MainDialog.h>`

Inheritance diagram for SynthGUI::mainDialog::



Public Methods

- **mainDialog** (Synth *syn)
  Create with a pointer to Synth.
- void **drawmain** ()
  Updates the main window.
- **~mainDialog** ()
  Destructor.
- void **MessageLoop** ()
  Enter infinite loop.
- void **HandleEvents** (event ev)
  Private eventhandler.
- void **drawMod** (int x, int y, int x2, int y2, std::string s, **mod** *md, bool drawLines)
  Draws a single 'mod'.
- int **mouseOverMod** (int x, int y)
  Checks whether mouse over mod.

7.25.1 Detailed Description

The main UI window.

Definition at line 177 of file MainDialog.h.

7.25.2 Member Function Documentation

7.25.2.1 void SynthGUI::mainDialog::drawMod (int *x*, int *y*, int *x2*, int *y2*, std::string *s*, mod * *md*, bool *drawLines*)

Draws a single 'mod'.

Todo:
privatize?

Definition at line 16 of file MainDialog.cpp.

References SynthGUI::modList::chosen, SynthGUI::modList::getMod(), SynthGUI::modList::mods, and SynthGUI::mod::mv.

Referenced by drawmain().

7.25.2.2 int SynthGUI::mainDialog::mouseOverMod (int *x*, int *y*)

Checks whether mouse over mod.

Todo:
privatize?

Definition at line 142 of file MainDialog.cpp.

References SynthGUI::modList::mods.

Referenced by HandleEvents().

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

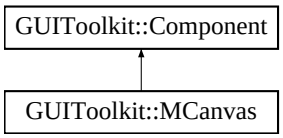
- MainDialog.h
- MainDialog.cpp

7.26 GUIToolkit::MCanvas Class Reference

Double-buffered canvas.

`#include <Canvas.h>`

Inheritance diagram for GUIToolkit::MCanvas::



Public Methods

- **MCanvas** ()

  Constructor.

- ∼**MCanvas** ()

  Destructor.

- void **PutInWinForm** (**WinForm** ∗WF, int x0, int y0, int x1, int y1)

  Place canvas on windform.

- void **HandleEvents** (**event** e)
- HWND **getHWND** ()

  Returns **Window** (p. 116)*-handle.*

- void **draw** ()

  Copies the off-screen buffer to the on screen buffer.

- void **doDraw** ()

  Asks windows to refresh the window.

- void **clear** ()

  Clears off-screen buffer.

- int **getWidth** ()
- int **getHeight** ()
- void **putPixel** (int x, int y, int r, int g, int b)

  Putpixel.

- void **setColor** (int r, int g, int b)

  setColor (it will clean up GDI objects for you)

- void **setColor** (COLORREF c)

  setColor (it will clean up GDI objects for you)

- void **deleteColor** ()

Normally, no need to call this. Used by Setcolor.

- void **setBrush** (int r, int g, int b)
  setBrush
- void **setBrush** (COLORREF c)
  setBrush
- void **deleteBrush** ()
  Normally, no need to call this. Used by setBrush.
- void **print** (int x, int y, std::string s)
  Displays string.
- void **line** (int x, int y, int x2, int y2)
  Line.
- void **circle** (int x, int y, int radius)
  Circle. (x,y) is center, not upper left corner.
- void **rectangle** (int x, int y, int x2, int y2)
  Rectangle.
- void **setBGColor** (int r, int g, int b)
  Sets background color (for text).
- void **setBGColor** (COLORREF c)
  Set background color (for text).
- void **setTextColor** (int r, int g, int b)
  Set text color.
- void **setTextColor** (COLORREF c)
  Set text color.

7.26.1 Detailed Description

Double-buffered canvas.

Definition at line 24 of file Canvas.h.

7.26.2 Member Function Documentation

7.26.2.1 void GUIToolkit::MCanvas::HandleEvents (event *e*) [virtual]

Pass events to user defined event handler (if any) Notice: this also restrict mouse dragging by capturing the cursor in the canvas when LMB is held.

Reimplemented from **GUIToolkit::Component** (p. 35).

Definition at line 149 of file Canvas.cpp.

References GUIToolkit::event::EventSource, GUIToolkit::EventHandler::HandleEvents(), and GUIToolkit::event::msg.

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

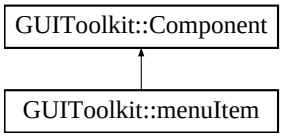
- Canvas.h
- Canvas.cpp

7.27 GUIToolkit::menuItem Class Reference

Items to be used on a context menu.

`#include <menu.h>`

Inheritance diagram for GUIToolkit::menuItem::



Public Methods

- **menuItem** (std::string s)
  Create item with caption 's'.
- ~**menuItem** ()
  Destructor.

Public Attributes

- int **ID**
  Unique ID.
- std::string **Label**
  Label (p. 62).

7.27.1 Detailed Description

Items to be used on a context menu.

Definition at line 17 of file menu.h.

7.27.2 Member Data Documentation

7.27.2.1 int GUIToolkit::menuItem::ID

Unique ID.

Todo:
privatize

Reimplemented from **GUIToolkit::Component** (p. 35).

Definition at line 27 of file menu.h.

Referenced by GUIToolkit::popupMenu::add(), and menuItem().

7.27.2.2 std::string GUIToolkit::menuItem::Label

Label (p. 62).

Todo:
privatize

Definition at line 30 of file menu.h.

Referenced by GUIToolkit::popupMenu::add().

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

- menu.h

7.28 SynthGUI::mod Class Reference

mod is the class for visually representing a module.

`#include <MainDialog.h>`

Public Methods

- **mod** (Module *mm)

  Call with a pointer to module.

- ~**mod** ()

  Destructor.

- void **addModule** (Module *mm)
- void **displayDialog** ()

  This method displays the appropriate dialog box for changing module settings/routing.

Public Attributes

- **ModVector mv**
- int **x**

  Screen coordinate.

- int **y**

  Screen coordinate.

- int **x2**

  Screen coordinate.

- int **y2**

  Screen coordinate.

- std::vector< **modInOuts** * > **InOuts**

  In and outs.

7.28.1 Detailed Description

mod is the class for visually representing a module.

Definition at line 49 of file MainDialog.h.

7.28.2 Member Function Documentation

7.28.2.1 void SynthGUI::mod::addModule (Module * *mm*) [inline]

A 'mod' can point to several modules at once. More modules can added with this method.

Definition at line 79 of file MainDialog.h.

References mv.

7.28.3 Member Data Documentation

7.28.3.1 std::vector<modInOuts *> SynthGUI::mod::InOuts

In and outs.

Todo:
unused?

Definition at line 69 of file MainDialog.h.

7.28.3.2 ModVector SynthGUI::mod::mv

If the module is part of a voice (for polyphonic synths) One mod can point to many instances (but of course only one of these: the so called firstVoice is drawn). This is the list of modules.

Definition at line 54 of file MainDialog.h.

Referenced by addModule(), displayDialog(), SynthGUI::mainDialog::drawMod(), SynthGUI::EnvDialog::EnvDialog(), SynthGUI::FilterDialog::FilterDialog(), mod(), and SynthGUI::OscDialog::OscDialog().

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

- MainDialog.h
- MainDialog.cpp

7.29 SynthGUI::modInOuts Class Reference

Class for graphical representing in and outputs.

`#include <MainDialog.h>`

Public Methods

- **modInOuts** ()
- ~**modInOuts** ()

Public Attributes

- bool **Input**
- std::string **label**
- Module * **connectTo**

7.29.1 Detailed Description

Class for graphical representing in and outputs.

Todo:
obsolete?

Definition at line 32 of file MainDialog.h.

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

- MainDialog.h

7.30 SynthGUI::modList Class Reference

Containter for the 'mods' displayed on the main UI window.

`#include <MainDialog.h>`

Public Methods

- void **addMod** (**mod** *m)
  Adds mod to vector.
- void **addModule** (Module *m)
- **modList** ()
  Constructor.
- ~**modList** ()
  Destructor.
- void **print** ()
  Debugging information.
- void **setChosen** (int i)
  Set currently selected 'mod'.
- int **getMod** (Module *m)
- int **getFirstModule** (int ID)
  return index of a module in the first voice with a given index.

Public Attributes

- int **chosen**
  Number of currently selected mod (if any).
- std::vector< **mod** * > **mods**
  The actual mod vector.

7.30.1 Detailed Description

Containter for the 'mods' displayed on the main UI window.

Definition at line 89 of file MainDialog.h.

7.30.2 Member Function Documentation

7.30.2.1 void SynthGUI::modList::addModule (Module * *m*) [inline]

Adds a module to vector. As of now, more instances (polyphony) of a voice must be added manually!

Todo:
fix this.

Definition at line 107 of file MainDialog.h.

References addMod(), mods, SynthGUI::mod::x, SynthGUI::mod::x2, SynthGUI::mod::y, and SynthGUI::mod::y2.

Referenced by SynthGUI::mainDialog::mainDialog().

7.30.2.2 int SynthGUI::modList::getMod (Module * *m*) [inline]

returns index on 'mods' if the module is on list. returns -1 if module is not on list.

Definition at line 156 of file MainDialog.h.

References mods.

Referenced by SynthGUI::mainDialog::drawMod().

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

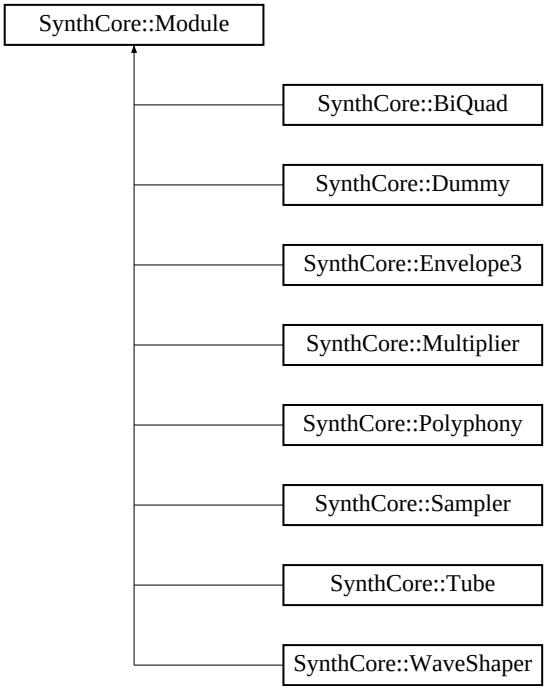
- MainDialog.h

7.31 SynthCore::Module Class Reference

Base class for all modules.

`#include <Modules.h>`

Inheritance diagram for SynthCore::Module::



Public Methods

- **Module** ()

  Constructor.

- virtual void **update** ()=0
- virtual int **getNoInputs** ()
- virtual **Input** ∗ **getInput** (int no)

  Returns pointer to an input.

- virtual void **addInput** (**Input** ∗I)
- virtual int **getNoOutputs** ()

  Returns number of outputs.

- virtual **Output** ∗ **getOutput** (int no)

  Returns pointer to an output.

- virtual void **addOutput** (**Output** ∗I)
- virtual std::string **getName** ()

  The name is just a virtual placeholder.

- virtual void **newMidiEvent** ()
- virtual std::string **getXML** (int indent=0)
- virtual void **loadXML** (XMLNode *n, bool firstPass)
- std::string **saveID** (int indent=0)
- virtual Module * **sharedClone** ()
- void **loadParameter** (string tagName, float par, XMLNode *n)

Static Public Methods

- int **getUniqueID** ()

  Call this to get an unique ID for a module.

- void **resetIDCount** (int i)

  Resets ID Count.

- Module * **newInstance** ()

  Use this as a 'factory' constructor.

- std::string **saveParameter** (std::string tag, float f, int indent=0)
- std::string **saveParameter** (std::string tag, std::string s, int indent=0)

Public Attributes

- **SynthVoice** * **hostVoice**
- int **ModID**
- bool **firstVoice**

Static Public Attributes

- int **IDCount** = 1

  The counter for the unique ID.

Friends

- std::ostream & **operator**<< (std::ostream &ostr, Module &myMod)

  Modules can be streamed to an ostream.

- std::ostream & **operator**<< (std::ostream &ostr, Module *myMod)

  Modules pointers can also be streamed to an ostream.

7.31.1 Detailed Description

Base class for all modules.

Todo:
consider the RTTI reflection.

Definition at line 39 of file Modules.h.

7.31.2 Member Function Documentation

7.31.2.1 virtual void SynthCore::Module::addInput (Input * *I*) [inline, virtual]

Adds the input to the list of inputs (myInputs) This is called from the **Input** (p. 61) constructor - so it shouldnt be called manually.

Definition at line 68 of file Modules.h.

Referenced by SynthCore::Input::Input().

7.31.2.2 virtual void SynthCore::Module::addOutput (Output * *I*) [inline, virtual]

Adds the output to the list of outputs (myOutputs) This is called from the **Output** (p. 85) constructor - so it shouldnt be called manually.

Definition at line 78 of file Modules.h.

Referenced by SynthCore::Output::Output().

7.31.2.3 virtual int SynthCore::Module::getNoInputs () [inline, virtual]

A bit of 'reflection' must be implemented This returns the number of inputs. Used with **getInput**() (p. 75)

Definition at line 61 of file Modules.h.

7.31.2.4 string SynthCore::Module::getXML (int *indent* = 0) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented in **SynthCore::Envelope3** (p. 49), **SynthCore::BiQuad** (p. 28), **SynthCore::Polyphony** (p. 89), **SynthCore::Sampler** (p. 96), **SynthCore::Tube** (p. 113), and **SynthCore::WaveShaper** (p. 115).

Definition at line 27 of file Modules.cpp.

References getName(), Utils::newline(), and Utils::space().

Referenced by SynthCore::operator<<().

7.31.2.5 void SynthCore::Module::loadXML (XMLNode * *n*, bool *firstPass*) [virtual]

Loads parameter setting from an XML file. Returns true if succesfull.

Reimplemented in **SynthCore::Envelope3** (p. 48), **SynthCore::BiQuad** (p. 27), **SynthCore::Polyphony** (p. 88), **SynthCore::Sampler** (p. 94), **SynthCore::Tube** (p. 112), and **SynthCore::WaveShaper** (p. 114).

Definition at line 132 of file Modules.cpp.

Referenced by SynthCore::Synth::loadXML().

7.31.2.6 virtual void SynthCore::Module::newMidiEvent () [inline, virtual]

This function is called if the module is part of a synthVoice, and a midi event occours (As of now, the detected events are: key up/down) Overwrite this if the module needs to react on midi events (IE oscillators).

Reimplemented in **SynthCore::Sampler** (p. 94).

Definition at line 91 of file Modules.h.

7.31.2.7 virtual Module* SynthCore::Module::sharedClone () [inline, virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented in **SynthCore::Envelope3** (p. 49), **SynthCore::BiQuad** (p. 29), **SynthCore::Sampler** (p. 96), **SynthCore::Tube** (p. 113), and **SynthCore::WaveShaper** (p. 115).

Definition at line 132 of file Modules.h.

7.31.2.8 virtual void SynthCore::Module::update () [pure virtual]

update() (p. 78) is where the main sound processing in a module takes place. Called every sample-step. It is purely virtual

Implemented in **SynthCore::Envelope3** (p. 48), **SynthCore::BiQuad** (p. 27), **SynthCore::Multiplier** (p. 82), **SynthCore::Dummy** (p. 41), **SynthCore::Polyphony** (p. 88), **SynthCore::Sampler** (p. 94), **SynthCore::Tube** (p. 112), and **SynthCore::WaveShaper** (p. 114).

Referenced by SynthCore::Input::updateRead().

7.31.3 Member Data Documentation

7.31.3.1 bool SynthCore::Module::firstVoice

If an module is cloned (for use in a polyphonic context) the flag indicate if this is the original module. This is important if modules shares ressources: IE: you have 32 voices with a sampler module in each. These will share the waveform data. when cleaning up, only one of these modules is allowed to delete waveform.

Definition at line 111 of file Modules.h.

Referenced by SynthCore::Sampler::sharedClone(), SynthCore::BiQuad::sharedClone(), and SynthCore::Envelope3::sharedClone().

7.31.3.2 SynthVoice* SynthCore::Module::hostVoice

Used if the **Module** (p. 75) is part of a HostVoice. When a polyphonic instrument is played, certain modules (IE the oscillators and envelopes) must be duplicated. These units, which are to be duplicated are called voices.

Todo:
Change HostSynth to HostVoice

Definition at line 100 of file Modules.h.

Referenced by SynthCore::BiQuad::BiQuad(), SynthCore::Envelope3::Envelope3(), Module(), SynthCore::Multiplier::Multiplier(), SynthCore::Sampler::newMidiEvent(), SynthCore::Polyphony::Polyphony(), SynthCore::WaveShaper::sharedClone(), SynthCore::Tube::sharedClone(), SynthCore::Sampler::sharedClone(), SynthCore::BiQuad::sharedClone(), SynthCore::Envelope3::sharedClone(), SynthCore::Tube::Tube(), SynthCore::BiQuad::update(), SynthCore::Envelope3::update(), and SynthCore::WaveShaper::WaveShaper().

7.31.3.3 int SynthCore::Module::ModID

A HostVoice consists of modules with different ModID's But all HostVoices should share the same set of ModID's Thereby establishing an identity between modules in different Voices.

Definition at line 105 of file Modules.h.

Referenced by Module(), SynthCore::Sampler::sharedClone(), SynthCore::BiQuad::sharedClone(), and SynthCore::Envelope3::sharedClone().

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

- Modules.h
- Modules.cpp

7.32 SynthCore::moduleRegistry Class Reference

Register all modules in this class In order to load **Synth** (p. 101) presets from an XML file and in order to use the GUI the system needs to 'know' about the modules. It needs a pointer to a constructor so that it can create objects from string representations at run-time. Therefore it is neccesary to register using the 'add' method in the registry.

`#include <Modules.h>`

Public Types

- typedef **Module** *(* **newPtr**)(**SynthVoice** *S)

  newPtr is a pointer to function taking a **SynthVoice** (p. 105)* *as argument, and returning a* **Module** (p. 75)*.

Public Methods

- void **add** (string s, **newPtr** np)
- **Module** * **getNew** (string s, **SynthVoice** *S)

  Lookup the string and return new object.

Static Public Methods

- moduleRegistry * **getInstance** ()

7.32.1 Detailed Description

Register all modules in this class In order to load **Synth** (p. 101) presets from an XML file and in order to use the GUI the system needs to 'know' about the modules. It needs a pointer to a constructor so that it can create objects from string representations at run-time. Therefore it is neccesary to register using the 'add' method in the registry.

There can only be one instance of this class (the constructor is private) - it is an example of a 'singleton' design pattern.

Definition at line 168 of file Modules.h.

7.32.2 Member Function Documentation

7.32.2.1 void SynthCore::moduleRegistry::add (string *s*, newPtr *np*) `[inline]`

Add module to registry. Every module MUST register here in order to be able to be serialized (saved) and in order for the GUI to function properly. The newPtr is a pointer to a static 'constructor' taking a **SynthVoice** (p. 105)* as argument and returning a new **Module** (p. 75) *.

Definition at line 179 of file Modules.h.

References newPtr.

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

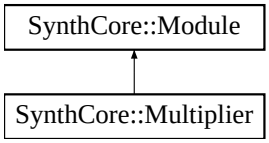
- Modules.h
- Modules.cpp

7.33 SynthCore::Multiplier Class Reference

Multiplier (p. 82): IE ring-modulator, VCA or amplitude modulator.

`#include <Modules.h>`

Inheritance diagram for SynthCore::Multiplier::



Public Methods

- virtual std::string **getName** ()
  The name is just a virtual placeholder.
- **Multiplier** (**SynthVoice** *S1)
  Constructor.
- ~**Multiplier** ()
  Destructor;.
- void **update** ()
  Actual Code;.

Static Public Methods

- **Module** * **newInstance** (**SynthVoice** *S1)
  Static constructor (for 'factory' use).

Public Attributes

- **Output** * **Output1**
  Mono **Module** (p. 75).
- **Input** * **Input1**
- **Input** * **Input2**

7.33.1 Detailed Description

Multiplier (p. 82): IE ring-modulator, VCA or amplitude modulator.

Multiplier (p. 82). That is, a VCA (voltage controlled amplifier) This can be used for ring modulation. No oversampling yet What is the difference between ring modulation and amplitude modulation?

Todo:

Oversampling? Should it be possible to take absolute value of one (or both) of the signals?

Definition at line 220 of file Modules.h.

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

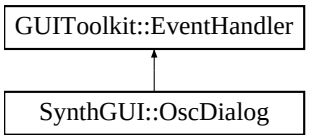
- Modules.h
- Modules.cpp

7.34 SynthGUI::OscDialog Class Reference

Dialog window for manipulating Oscillator/Sampler setting.

`#include <OscDialog.h>`

Inheritance diagram for SynthGUI::OscDialog::



Public Methods

- **OscDialog** (**mod** *mmod)
  Takes pointer to mod.
- **~OscDialog** ()
  Destructor.
- void **MessageLoop** ()
  Enter infinite loop.
- void **HandleEvents** (event ev)
  Private message handler.

7.34.1 Detailed Description

Dialog window for manipulating Oscillator/Sampler setting.

Definition at line 26 of file OscDialog.h.

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

- OscDialog.h
- OscDialog.cpp

7.35 SynthCore::Output Class Reference

Output (p. 85) is a generic connector endpoint.

`#include <Output.h>`

Public Methods

- void **ConnectTo** (**Input** *I)
- **Output** (**Module** *host)

  Constructor takes pointer to host module.

- ~**Output** ()

  Destructor.

- float **read** ()

  This will return the buffer value.

- void **loadXML** (string tagName, XMLNode *n)

  Load values from XMLNode.

- std::string **getXML** (std::string tag, int indent=0)

  Save state to XML string.

- void **write** (float val)

  Write shold be called from Host Modules Update()-function.

Public Attributes

- float **value**

  This buffers the output value.

- **Module** * **HostModule**

  We live in this host.

7.35.1 Detailed Description

Output (p. 85) is a generic connector endpoint.

Output (p. 85) is a generic connector endpoint Connects to an input. Every module contains a number of inputs/outputs

Definition at line 33 of file Output.h.

7.35.2 Member Function Documentation

7.35.2.1 void SynthCore::Output::ConnectTo (Input * *I*)

This is a bit obsolete, Since outputs can be connected to severel inputs. Dont use it.

Todo:
Remove this!

Definition at line 69 of file Input.cpp.

References SynthCore::Input::ConnectFrom(), and value.

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

- Output.h
- Input.cpp

7.36 Utils::parseError Class Reference

Exception thrown by XMLDoc.parser.

`#include <Utils.h>`

Public Methods

- **parseError** (string message)
  Constructor with Error message.
- void **print** ()
  Report error.

7.36.1 Detailed Description

Exception thrown by XMLDoc.parser.

Definition at line 70 of file Utils.h.

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

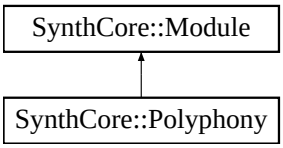
- Utils.h

7.37 SynthCore::Polyphony Class Reference

Combines the polyphonic parts into one signal.

`#include <polyphony.h>`

Inheritance diagram for SynthCore::Polyphony::



Public Methods

- virtual std::string **getName** ()

  The name is just a virtual placeholder.

- **Polyphony** ()

  Constructor.

- **~Polyphony** ()

  Destructor;.

- void **update** ()

  Actual Code.

- std::string **getXML** (int indent)
- void **addVoice** (**SynthVoice** *voice)

  Call this to register a voice.

- void **setVoices** (int number)
- int **getVoiceCount** ()

  Return the maximum number of simultaneous voices.

- void **clearInputs** ()

  Clears the inputs vector.

- void **loadXML** (XMLNode *n, bool firstPass)

  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

Static Public Methods

- **Module** * **newInstance** (**SynthVoice** *S)

  Static constructor (for 'factory' use).

Public Attributes

- **Output * Output1**

  Mono output.

7.37.1 Detailed Description

Combines the polyphonic parts into one signal.

A polyphonic synthesizer needs to add a number a voices together. This is done by this module.

Definition at line 26 of file polyphony.h.

7.37.2 Member Function Documentation

7.37.2.1 std::string SynthCore::Polyphony::getXML (int *indent*) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 75 of file polyphony.cpp.

References SynthCore::Output::getXML(), Utils::intToString(), Utils::newline(), Output1, and Utils::space().

7.37.2.2 void SynthCore::Polyphony::setVoices (int *number*)

Set the number of polyphonic voices.

Todo:
not finished yet

Definition at line 52 of file polyphony.cpp.

Referenced by addVoice().

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

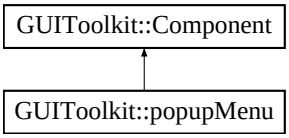
- polyphony.h
- polyphony.cpp

7.38 GUIToolkit::popupMenu Class Reference

A context menu (usually invoked on right mouse button click).

`#include <menu.h>`

Inheritance diagram for GUIToolkit::popupMenu::



Public Methods

- **popupMenu** (HWND hw)
- ~**popupMenu** ()

  Destructor.

- void **addToHWND** (HWND hwnd)

  If HWND is not known at constructor time, use this later.

- void **add** (**menuItem** *item)

  Adds item to menu.

- void **showMenu** ()

  Show menu. Ensure HWND is set.

7.38.1 Detailed Description

A context menu (usually invoked on right mouse button click).

Definition at line 36 of file menu.h.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 GUIToolkit::popupMenu::popupMenu (HWND *hw*)

Constructor. Must be called with handle to the window we'll display it on.

Todo:
should be called with something else than a HWND

Definition at line 13 of file menu.cpp.

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

- menu.h
- menu.cpp

7.39 SynthCore::Sample Class Reference

Sample (p. 91) Container.

`#include <sndutils.h>`

Public Methods

- **Sample** ()
  Trivial Constructor.
- **Sample** (int sampleLength)
  Allocate space for 'sampleLength' samples. Theres no wrap.
- **Sample** (int sampleLength, int wrapAround)
  Allocate space for 'sampleLength' samples. Theres 'wrapAround' samples duplicated at the end.
- ~**Sample** ()
  Destructor.
- float **getMax** ()
  Get maximum sample value.
- float **getMin** ()
  Get minimum sample value.
- void **makeWrap** ()
  Copy the first 'wrapAround' samples to end of sample.
- void **wavenormalize** ()
  Normalize sample to values between (-1,1).
- void **normalize** ()
  Normalize sample to values between (0,1).

Public Attributes

- int **length**
  Sample (p. 91) *Length.*
- int **wrap**
  The 'wrap' first samples are duplicated at the end. Useful for interpolation.
- float **samplingFreq**
  Sampling Frequency.
- float **samplingTune**
  Frequency of the note sampled.

- float * **samples**

  Actual 32 bit float data.

7.39.1 Detailed Description

Sample (p. 91) Container.

Only 32-bit float samples are supported.

Definition at line 152 of file sndutils.h.

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

- sndutils.h

7.40 SynthCore::SampleMap Class Reference

SampleMap (p. 93) maps midikeys (0-127) to corresponding samples.

`#include <sampler.h>`

Public Methods

- **SampleMap** ()

  Constructor: initializes all map entries to 0.

- ~**SampleMap** ()

  The destructor will delete all used samples in the samplemap.

- void **clean** ()

  Clears samplemap and delete samples.

- float * **getSamples** (int Midikey)

  Returns sample pointer.

Public Attributes

- **Sample** * **map** [128]

7.40.1 Detailed Description

SampleMap (p. 93) maps midikeys (0-127) to corresponding samples.

Definition at line 24 of file sampler.h.

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

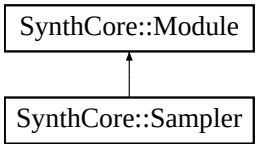
- sampler.h

7.41 SynthCore::Sampler Class Reference

The main oscillator class.

`#include <sampler.h>`

Inheritance diagram for SynthCore::Sampler::



Public Types

- enum { **sine**, **triangle**, **sawtooth**, **square**, **wave1**, **wave2**, **wave3**, **wave4** }

  type of factory Waves

- enum { **none**, **linear**, **Lagrange**, **cubicHermite**, **spline5Point**, **sinc**, **sincWindow**, **pureSine** }

  Not all of these are implemented: only 'none', linear, cubicHermite, spline5point,pureSine so far.

Public Methods

- void **loadXML** (XMLNode *n, bool firstPass)

  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

- std::string **getXML** (int indent)
- void **newMidiEvent** ()

  Tell sampler to change note and sampleMap entry.

- virtual std::string **getName** ()

  Module (p. 75) *name.*

- **Sampler** (**SynthVoice** *S1)
- ~**Sampler** ()
- virtual **Module** * **sharedClone** ()
- void **update** ()

  Actual Code;.

- void **setMidiNote** (int note)

  The sampler must know not only frequency, but also the midi-note in order to look the right sample in the samplemap.

- void **setTriangle** (int length)

  Initialize sampler to band-limited Triangle waveform.

- void **setSquare** (int length)

  Initialize sampler to band-limited Square waveform.

- void **setSawtooth** (int length)

  Initialize sampler to band-limited SawTooth waveform.

- void **setSine** (int length)

  Initialize sampler to Sinus waveform.

- void **setWave1** (int length)
- void **setWave2** (int length)
- void **setWave3** (int length)
- void **setWave4** (int length)
- void **resetPhase** ()

  Reset the phase of the oscillator.

Static Public Methods

- **Module** ∗ **newInstance** (**SynthVoice** ∗S1)

  Static constructor (for 'factory' use).

Public Attributes

- enum SynthCore::Sampler:: { ... } **waveformType**

  type of factory Waves

- int **factoryWaveform**
- int **interpolationType**

  Decides how to interpolate between samples. See the enum for more info.

- **Output** ∗ **Output1**

7.41.1 Detailed Description

The main oscillator class.

'sampler' is the main oscillator. It is meant to be a versatile oscillator, built around interpolated sample playback. Different interpolation techniques will be implemented: none, linear, Lagrange, Hermitean, Sinc. For now, only Hermitean interpolation is implemented (but it is better than linear and Langrange and much faster than Sinc!) Aliasing is avoided by having a band-limited sample map (for EACH midi key a 1024 sample waveform is calculated using inverse FFT. Notice that some of the low-keys (around f3 or so) won't be able to have a full harmonic spectrum, due to 1024 sample limitation.)

Frequency Modulation and Pulse Width Modulation will be implemented, but not in a alias-free way (perhaps an oversampling solution will be employed at some point)

Some assembler optimization is done (I found the routines on music-dsp - seems I ought to find a reference).

Definition at line 76 of file sampler.h.

7.41.2 Member Function Documentation

7.41.2.1 std::string SynthCore::Sampler::getXML (int *indent*) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 65 of file sampler.cpp.

References factoryWaveform, SynthCore::Output::getXML(), interpolationType, Utils::intToString(), Utils::newline(), and Utils::space().

7.41.2.2 Module * SynthCore::Sampler::sharedClone () [virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented from **SynthCore::Module** (p. 78).

Definition at line 23 of file sampler.cpp.

References SynthCore::Module::firstVoice, freq, SynthCore::Module::hostVoice, interpolationType, SynthCore::Module::ModID, and mySampleMap.

7.41.3 Member Data Documentation

7.41.3.1 int SynthCore::Sampler::factoryWaveform

The current type of waveform (if factory waveform is choosen). (for the time being no user waveforms are implemented).

Definition at line 85 of file sampler.h.

Referenced by getXML(), loadXML(), setSawtooth(), setSine(), setSquare(), and setTriangle().

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

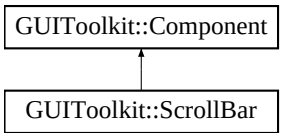
- sampler.h
- sampler.cpp

7.42 GUIToolkit::ScrollBar Class Reference

A standard scroll bar.

`#include <ListBox.h>`

Inheritance diagram for GUIToolkit::ScrollBar::



Public Methods

- **ScrollBar** (float min, float max, int steps)
- ~**ScrollBar** ()

  Destructor.

- void **PutInWinForm** (**WinForm** *WF, int x0, int y0, int sizex, int sizey)

  Place on **WinForm** (p. 117).

- void **HandleEvents** (**event** e)
- HWND **getHWND** ()

  Return windows handle.

- void **show** ()

  Shows **ScrollBar** (p. 97).

- void **hide** ()

  Hides **ScrollBar** (p. 97).

- void **set** (float p)

  Sets thumb position to 'p' in the interval (min;max).

- void **setInt** (int p)

  Sets thumb position to integer 'p' in the interval (0; steps).

- float **get** ()

  Returns current position.

7.42.1 Detailed Description

A standard scroll bar.

Definition at line 85 of file ListBox.h.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 GUIToolkit::ScrollBar::ScrollBar (float *min*, float *max*, int *steps*)

Constructor. Specify min and max value (as returned from the **get**() (p. 97) method) and the number of possible states, 'steps'.

Definition at line 13 of file Listbox.cpp.

7.42.3 Member Function Documentation

7.42.3.1 void GUIToolkit::ScrollBar::HandleEvents (event *e*) [virtual]

Calls user event handler. Notice: in contrast to a standard windows control, this one automatically takes care of updating the thumb position whenever someone clicks on it.

Reimplemented from **GUIToolkit::Component** (p. 35).

Definition at line 62 of file Listbox.cpp.

References GUIToolkit::event::EventSource, get(), GUIToolkit::EventHandler::HandleEvents(), GUIToolkit::event::msg, set(), setInt(), and GUIToolkit::event::wParam.

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

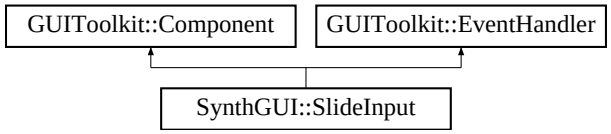
- ListBox.h
- Listbox.cpp

7.43 SynthGUI::SlideInput Class Reference

A combined slider and text box input component.

`#include <AInput.h>`

Inheritance diagram for SynthGUI::SlideInput::



Public Methods

- **SlideInput** (std::string s, float min, float max, int steps, bool integral)
- **~SlideInput** ()

  Destructor.

- void **PutInWinForm** (WinForm *WF, int x0, int x1, int y0, int y1)

  Places component on WinForm.

- void **HandleEvents** (event ev)

  Set user event handler.

- float **getValue** ()

  Returns value.

7.43.1 Detailed Description

A combined slider and text box input component.

Definition at line 88 of file AInput.h.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 SynthGUI::SlideInput::SlideInput (std::string *s*, float *min*, float *max*, int *steps*, bool *integral*)

's' is the caption of the module. The allowed interval is [min;max] quantized in 'steps' steps. 'integral' can be set to true, if only integral values are allowed.

Definition at line 133 of file AInput.cpp.

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

- AInput.h
- AInput.cpp

7.44 softSynth Class Reference

Softsynth is derived from AudioEffectX and implements the basic VST interface.

`#include <softsynth.h>`

Public Methods

- **softSynth** (audioMasterCallback audioMaster)
- ~**softSynth** ()
- virtual void **process** (float **inputs, float **outputs, long sampleframes)
- virtual void **processReplacing** (float **inputs, float **outputs, long sampleframes)
- virtual long **processEvents** (VstEvents *events)
- virtual void **getProgramName** (char *name)
- virtual void **setSampleRate** (float sampleRate)
- virtual void **setBlockSize** (long blockSize)
- virtual void **resume** ()
- virtual bool **getOutputProperties** (long index, VstPinProperties *properties)
- virtual bool **getEffectName** (char *name)
- virtual bool **getVendorString** (char *text)
- virtual bool **getProductString** (char *text)
- virtual long **getVendorVersion** ()
- virtual long **canDo** (char *text)

Public Attributes

- Synth * **mySynth**

7.44.1 Detailed Description

Softsynth is derived from AudioEffectX and implements the basic VST interface.

Definition at line 28 of file softsynth.h.

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

- softsynth.h

7.45 SynthCore::Synth Class Reference

The main container for the various modules.

`#include <Synth.h>`

Public Methods

- **Synth** ()
- ~**Synth** ()
- void **update** ()

  Calls Update on Modules in UpdateMap.

- void **StateChange** ()
- void **add** (**Module** *m)
- void **noteOff** (int note)

  Pass MIDI signals to this.

- void **noteOn** (int note)
- void **allNotesOff** ()

  Reset MIDI notes.

- std::string **getXML** (int indent=0)

  Return a serialized XML representation of this object, including all sub-components.

- void **loadXML** (std::string fileName)

  Load a serialized XML.

- void **XMLRegister** (string Attr, **Output** *out)

  Used while parsing XML files. (semi-private).

- **Output** * **XMLGetOutput** (string Attr)

Public Attributes

- **Output** * **Output1**

  Left **Output** (p. 85) *From* **Synth** (p. 101).

- **Output** * **Output2**

  Main **Output** (p. 85) *From* **Synth** (p. 101).

- bool **active**
- **SynthVoice** * **voices** [4]
- **ModuleMap UpdateMap**

  Order of automatic updated modules.

- **ModuleMap totalMap**

  Map of all components (also including manually updated ones).

7.45.1 Detailed Description

The main container for the various modules.

All active modules resides in here. There are methods for building updatemaps, which controls the order in which the sample calculation takes place

Todo:
needs some cleaning. Midi section needs improvement: remember velocity, pedal on/off, and pitch bend.

Definition at line 109 of file Synth.h.

7.45.2 Member Function Documentation

7.45.2.1 void SynthCore::Synth::add (Module * *m*)

Call this to register module on synth. Registered on totalMap Must be done if UI is to display the module.

Definition at line 231 of file Synth.cpp.

References totalMap.

Referenced by SynthCore::SynthVoice::add().

7.45.2.2 void SynthCore::Synth::noteOn (int *note*)

Pass MIDI signals to this.

Todo:
velocity

Definition at line 350 of file Synth.cpp.

References SynthCore::FreqTable::freqTab, StateChange(), and voices.

7.45.2.3 void SynthCore::Synth::StateChange ()

Must be called after a StateChange, IE when some module routes are not active anymore (a voice dies out) (After NoteOn & Last decay of note especially)

Definition at line 330 of file Synth.cpp.

References SynthCore::Output::HostModule, Output1, and UpdateMap.

Referenced by noteOn().

7.45.2.4 Output* SynthCore::Synth::XMLGetOutput (string *Attr*) [inline]

Search for an **Output** (p. 85) on XMLOutput map. Returns 0 if the search fails.

Definition at line 174 of file Synth.h.

7.45.3 Member Data Documentation

7.45.3.1 bool SynthCore::Synth::active

'active' functions as a mutex: When multi-threaded applications (IE an GUI and the VST interface) accesses the synth class It is sometimes neccesary to turn the VST interface access off. This is done by setting active to false.

Definition at line 123 of file Synth.h.

7.45.3.2 SynthVoice* SynthCore::Synth::voices[4]

The various synth voices (IE the polyphonic components)

Todo:
Change to vector!

Definition at line 153 of file Synth.h.

Referenced by allNotesOff(), noteOff(), and noteOn().

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

- Synth.h
- Synth.cpp

7.46 SynthCore::SynthEngine Class Reference

`#include <SynthEngine.h>`

Public Methods

- **SynthEngine** ()
- ~**SynthEngine** ()

Static Public Methods

- long **GlobalStep** ()
- void **setSynth** (**Synth** *S)
- void **active** (bool b)
- bool **isActive** (bool b)

Public Attributes

- float **samplerate**

Static Public Attributes

- long **Globalstep** = 0
- **Synth** * **mySynth** = 0

7.46.1 Detailed Description

Static class for setting some 'global' parameters: IE samplerate and so on.

Todo:
not really implemented yet.

Definition at line 18 of file SynthEngine.h.

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

- SynthEngine.h
- SynthEngine.cpp

7.47 SynthCore::SynthVoice Class Reference

`#include <SynthVoice.h>`

Public Methods

- **SynthVoice** (**Synth** *S, bool defaultSetup=true)

  Constructor.

- ~**SynthVoice** ()

  Destructor;.

- void **setMidiNote** (int n)

  Sets currently played midi note number,.

- int **getMidiNote** ()

  returns currently played midi note number

- void **setMidiFreq** (float f)
- float **getMidiFreq** ()
- void **add** (**Module** *m)
- void **SynthVoice::newMidiEvent** ()

  Ask **SynthVoice** (p. 105) *to informs its children.*

- SynthVoice * **SynthVoice::sharedClone** ()

Public Attributes

- **Synth** * **HostSynth**
- int **playingNote**
- bool **noteOn**
- bool **aLive**
- **Output** * **SynthOutput**
- **Sampler** * **myVFO**
- **Envelope3** * **myEnvelope**
- **BiQuad** * **myRez**
- **Envelope3** * **myEnv2**

7.47.1 Detailed Description

This class is a container for the various modules in a voice unit.

Todo:
needs serious rewriting: must have dynamical routing!

Definition at line 42 of file SynthVoice.h.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 SynthCore::SynthVoice::~SynthVoice ()

Destructor;.

The synth (the host) takes care of cleaning up modules so dont delete them in destructor

Definition at line 128 of file SynthVoice.cpp.

7.47.3 Member Function Documentation

7.47.3.1 void SynthCore::SynthVoice::add (Module * *m*)

Call this to register module on synth. Must be done if the **SynthVoice** (p. 105) are to pass on MidiEvents.

Definition at line 51 of file SynthVoice.cpp.

References SynthCore::Synth::add().

Referenced by SynthVoice().

7.47.3.2 float SynthCore::SynthVoice::getMidiFreq () `[inline]`

Sets currently played frequency (this is not always obvious from the midi note because of the pitch bender)

Definition at line 79 of file SynthVoice.h.

Referenced by SynthCore::Sampler::newMidiEvent(), and SynthCore::BiQuad::update().

7.47.3.3 void SynthCore::SynthVoice::setMidiFreq (float *f*) `[inline]`

Sets currently played frequency (this is not always obvious from the midi note because of the pitch bender)

Definition at line 75 of file SynthVoice.h.

7.47.3.4 SynthVoice* SynthCore::SynthVoice::SynthVoice::sharedClone ()

Make a shared Clone of this synthVoice. A shared clone 'shares' ressources with the other synthVoices.

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

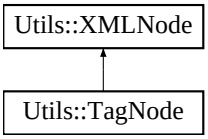
- SynthVoice.h
- SynthVoice.cpp

7.48 Utils::TagNode Class Reference

Used to represent a tag node (I.E.: <html>) .

`#include <Utils.h>`

Inheritance diagram for Utils::TagNode::



Public Methods

- **TagNode** ()
  Constructor.
- **TagNode** (std::string s)
  Init node with the tagName s.
- **~TagNode** ()
  Destructor.
- string **printString** ()
  Debug info : name , pointer, path.
- std::string **getXML** (int indent)
  Used when constructing a XML text representation.
- string **pathName** ()
  Used in **path***()* (p. 121) *to print path.*
- string **getAttribute** (string Attr)
  Returns attribute.

Public Attributes

- std::string **tagName**
  If the node is a <tag> this is the name;.

7.48.1 Detailed Description

Used to represent a tag node (I.E.: <html>) .

Definition at line 145 of file Utils.h.

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

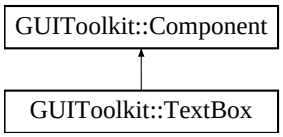
- Utils.h
- Utils.cpp

7.49 GUIToolkit::TextBox Class Reference

Single- og multi-line text box.

`#include <TextBox.h>`

Inheritance diagram for GUIToolkit::TextBox::



Public Types

- enum **TextBoxType** { **singleLine**, **multiLine** }
  Possible textbox types: singleLine or multiLine.

Public Methods

- **TextBox** ()
  Constructor. Defaults to singleLine.
- **TextBox** (**TextBoxType** myType)
  Constructor. Set type to singleLine or multiLine.
- ∼**TextBox** ()
  Destructor.
- void **PutInWinForm** (**WinForm** ∗WF, int x0, int x1, int y0, int y1)
  Place on winform.
- void **setText** (std::string s)
  Set text.
- void **setText** (float f)
  Set text to a float.
- void **addText** (std::string s)
  Adds text.
- void **clearText** (std::string s)
  Clears text.
- float **getFloatVal** ()
  Returns the floating point value of the text.
- void **HandleEvents** (**event** e)

Set user event handler.

- void **setFloatLimit** (float from, float to)

  Set limits on the acceptable float values. (Used when input is validated).

- bool **validateFloat** ()

  Validates whether the text is a floating pointer number in the interval set by setFloatLimit.

- std::string * **getText** ()

  Returns text.

- void **show** ()

  Shows text box.

- void **hide** ()

  Hides text box.

7.49.1 Detailed Description

Single- og multi-line text box.

Definition at line 22 of file TextBox.h.

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

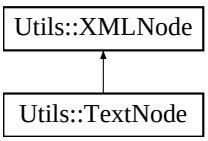
- TextBox.h
- TextBox.cpp

7.50 Utils::TextNode Class Reference

Used to represent text nodes.

`#include <Utils.h>`

Inheritance diagram for Utils::TextNode::



Public Methods

- **TextNode** ()
  Constructor.
- **TextNode** (std::string s)
  Init node with text = s.
- **~TextNode** ()
  Destructor.
- string **printString** ()
  Debug info : name , pointer, path.
- std::string **getXML** (int indent)
  Used when constructing a XML text representation.
- string **pathName** ()
  Used in **path***()* (p. 121) *to print path.*

Public Attributes

- std::string **text**
  The actual text.

7.50.1 Detailed Description

Used to represent text nodes.

Definition at line 171 of file Utils.h.

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

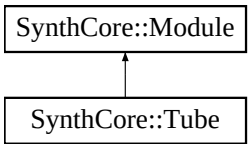
- Utils.h
- Utils.cpp

7.51 SynthCore::Tube Class Reference

Tube (p. 112) is a waveguide based experimental delay/reverb kind of effect.

`#include <tube.h>`

Inheritance diagram for SynthCore::Tube::



Public Methods

- virtual std::string **getName** ()

  We have a name?

- virtual **Module** ∗ **sharedClone** ()
- **Tube** (**SynthVoice** ∗S1)

  Constructor.

- std::string **getXML** (int indent)
- ∼**Tube** ()

  Destructor;.

- void **update** ()

  Actual Code resides here.

- void **loadXML** (XMLNode ∗n, bool firstPass)

  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

Static Public Methods

- **Module** ∗ **newInstance** (**SynthVoice** ∗S1)

  Static constructor (for 'factory' use).

Public Attributes

- float **feedback**

  Should be less than one, in order to avoid unstable feedback.

- **Output** ∗ **Output1**

  Stereo output: left channel.

- **Output** ∗ **Output2**

  Stereo output: left channel.

- **Input * Input1**

  Mono input.

7.51.1 Detailed Description

Tube (p. 112) is a waveguide based experimental delay/reverb kind of effect.

The tube is a waveguide: two delay lines where the signal flows in opposite direction in each one. The delay lines are connected at the ends throught a filter (low pass second order). The signal fades away exponentially if the feedback coefficient is less than one. (And guess what larger values leads to). Notice: this module takes a mono input, but delivers a stereo output.

Definition at line 29 of file tube.h.

7.51.2 Member Function Documentation

7.51.2.1 std::string SynthCore::Tube::getXML (int *indent*) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 35 of file tube.cpp.

References feedback, SynthCore::Input::getXML(), SynthCore::Output::getXML(), Input1, Utils::intToString(), Utils::newline(), Output1, Output2, and Utils::space().

7.51.2.2 Module * SynthCore::Tube::sharedClone () [virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented from **SynthCore::Module** (p. 78).

Definition at line 12 of file tube.cpp.

References SynthCore::Module::hostVoice, and Tube().

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

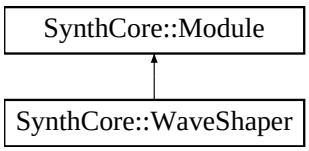
- tube.h
- tube.cpp

7.52 SynthCore::WaveShaper Class Reference

Aplies a non-linear transformation to input signal.

`#include <waveshaper.h>`

Inheritance diagram for SynthCore::WaveShaper::



Public Methods

- virtual std::string **getName** ()
  The name is just a virtual placeholder.
- virtual **Module** ∗ **sharedClone** ()
- **WaveShaper** (**SynthVoice** ∗S1)
  Constructor.
- ∼**WaveShaper** ()
  Destructor;.
- void **update** ()
  Actual Code;.
- void **setAmount** (float amount)
  Adjust a parameter of the waveshaping (if there is one available).
- std::string **getXML** (int indent)
- void **loadXML** (XMLNode ∗n, bool firstPass)
  Reads parameters from XML document (use the parser in the **Utils** (p. 19) *namespace).*

Static Public Methods

- **Module** ∗ **newInstance** (**SynthVoice** ∗S1)
  Static constructor (for 'factory' use).

Public Attributes

- **Output** ∗ **Output1**
  Input (p. 61) *signal.*
- **Input** ∗ **Input1**
  Distorted output.

7.52.1 Detailed Description

Aplies a non-linear transformation to input signal.

Waveshaping is nothing more than applying a function to the sample. IE: output = F(input). Usually the function will be non-linear (otherwise we are just amplifying) resulting in a distortion of the signal.

Typically use is distortion, simple compression but also generation of higher harmonics is possible. Since higher can be introduced during non-linear operations, aliasing is almost sure to happen.

Todo:
implement oversampling in order to cope with oversampling.

Definition at line 34 of file waveshaper.h.

7.52.2 Member Function Documentation

7.52.2.1 std::string SynthCore::WaveShaper::getXML (int *indent*) [virtual]

Return XML representation of parameters. Used for serializing modules.

Reimplemented from **SynthCore::Module** (p. 77).

Definition at line 49 of file waveshaper.cpp.

References SynthCore::Input::getXML(), SynthCore::Output::getXML(), Input1, Utils::intToString(), Utils::newline(), Output1, and Utils::space().

7.52.2.2 Module * SynthCore::WaveShaper::sharedClone () [virtual]

Creates a copy of the **Module** (p. 75). If any ressources can be shared across a voice the method should utilize it.

Reimplemented from **SynthCore::Module** (p. 78).

Definition at line 29 of file waveshaper.cpp.

References SynthCore::Module::hostVoice, and WaveShaper().

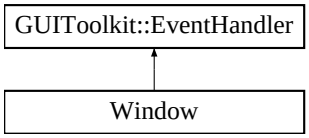
The documentation for this class was generated from the following files:

- waveshaper.h
- waveshaper.cpp

7.53 Window Class Reference

Test class for **GUIToolkit** (p. 11).

Inheritance diagram for Window::



Public Methods

- **Window** ()
- ~**Window** ()
- void **MessageLoop** ()
- void **HandleEvents** (event ev)

7.53.1 Detailed Description

Test class for **GUIToolkit** (p. 11).

Definition at line 57 of file GUIToolkit/main.cpp.

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

- GUIToolkit/main.cpp

7.54 GUIToolkit::WinForm Class Reference

The main windows class. A containter for all Components.

`#include <WinForm.h>`

Public Types

- enum **winTypes** { **standard** }
  For later user.

Public Methods

- **WinForm** (int xsize, int ysize, **winTypes** myTypes, std::string title="No title")
  Constructor. Set size and type.
- ~**WinForm** ()
  Destructor.
- void **show** ()
  Shows window.

Public Attributes

- HWND **hwnd**
  Windows handle.

7.54.1 Detailed Description

The main windows class. A containter for all Components.

Definition at line 34 of file WinForm.h.

7.54.2 Member Data Documentation

7.54.2.1 HWND GUIToolkit::WinForm::hwnd

Windows handle.

Todo:
privatize?

Definition at line 50 of file WinForm.h.

Referenced by GUIToolkit::TextBox::getText(), GUIToolkit::TextBox::PutInWinForm(), GUIToolkit::ComboBox::PutInWinForm(), GUIToolkit::ScrollBar::PutInWinForm(), GUIToolkit::Label::PutInWinForm(), GUIToolkit::divider::PutInWinForm(), GUIToolkit::MCanvas::PutInWinForm(), GUIToolkit::Button::PutInWinForm(), GUIToolkit::TextBox::setText(), show(), and WinForm().

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

- WinForm.h
- WinForm.cpp

7.55 Utils::XMLDoc Class Reference

Class for representing a XML document.

`#include <Utils.h>`

Public Methods

- **XMLDoc** ()
  Constructor.
- ∼**XMLDoc** ()
  Destructor.
- void **parseFile** (std::string filename)
  Parses a XML (specified by 'filename') into a XMLDOC DOM structure.
- vector< string > **tagPartition** (string s)
  split string into substrings

Public Attributes

- **XMLNode** ∗ **rootNode**
  Toplevel node in document.

7.55.1 Detailed Description

Class for representing a XML document.

Typical Use:

```
using namespace Utils;

// Create empty XMLDoc.
XMLDoc myDoc;

try {

       // Read file from hard disk.
       myDoc.parseFile("text.xml");

       // Create a search class.
       XMLSearch noteFinder(myDoc);

       // Match all notes starting with the following path:
       // As it can be seen, a wildcard at the end is legal. This is the only place you can use for now!
       noteFinder.find("/xml/note*");

       // Iterate through them:
       for (int i = 0; i<noteFinder.query.size(); i++)
       {
              cout << noteFinder.query[i]->printString() << endl;;
       }
```

```
 } catch (parseError pE) {

       // Report errors.
       pE.print();
}
```

Definition at line 252 of file Utils.h.

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

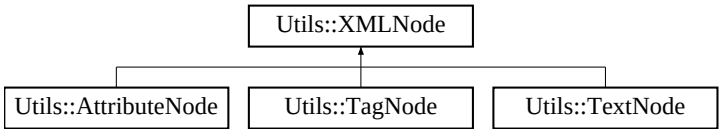
- Utils.h
- Utils.cpp

7.56 Utils::XMLNode Class Reference

The base class for all XMLNodes.

`#include <Utils.h>`

Inheritance diagram for Utils::XMLNode::



Public Methods

- bool **hasChildren** ()
  True if any children.
- bool **hasRightSibling** ()
  True if a Sibling to the right in the tree exists.
- bool **hasParent** ()
  True if a parentnode exists (only the top XML node is false!).
- void **addChild** (XMLNode *ch)
  Adds a child to node. It will be added to the right of other children.
- virtual string **printString** ()
  Debug info : name , pointer, path.
- virtual void **print** ()
  Used for print debug tree.
- **XMLNode** ()
  Constructor.
- **~XMLNode** ()
  Destructor.
- virtual std::string **getXML** (int indent)
  Used when constructing a XML text representation.
- virtual string **pathName** ()
  Used in **path***()* (p. 121) *to print path.*
- virtual string **path** ()
  the path to this element (I.E.: "/html/body/p/" matches text in toplevel paragraphs in a html doc.)
- string **getTagText** (string tag)
- **TagNode** * **getTag** (string tag)

Public Attributes

- XMLNode * **firstChild**

  Pointer to child.

- XMLNode * **rightSibling**

  Pointer to first sibling to the right (in a graphical tree view).

- XMLNode * **parent**

  Pointer to parent.

7.56.1 Detailed Description

The base class for all XMLNodes.

Definition at line 84 of file Utils.h.

7.56.2 Member Function Documentation

7.56.2.1 TagNode * Utils::XMLNode::getTag (string *tag*)

Returns the first child tagNode which name matches 'tag'. Does not recurse. Returns 0 if theres no such tag.

Definition at line 189 of file Utils.cpp.

References firstChild, hasChildren(), and rightSibling.

7.56.2.2 string Utils::XMLNode::getTagText (string *tag*)

Returns the text of the first node encountered with the tag 'tag' Returns "" if this cannot be matched. ! Notice that the tag must NOT nest other tags.

Definition at line 163 of file Utils.cpp.

References firstChild, hasChildren(), and rightSibling.

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

- Utils.h
- Utils.cpp

7.57 Utils::XMLSearch Class Reference

`#include <Utils.h>`

Public Methods

- **XMLSearch** (**XMLDoc** *d)
  Constructor.
- **XMLSearch** ()
  Destructor.
- void **find** (string MPath, **XMLNode** *startNode=0)
- void **clearQuery** ()
  Old queries will generate a bit of wasted space. Call this to clean up.

Public Attributes

- vector< **XMLNode** * > **answer**
  Used for buffering query.

7.57.1 Detailed Description

Implements search facilities. See find for more info

Definition at line 279 of file Utils.h.

7.57.2 Member Function Documentation

7.57.2.1 void Utils::XMLSearch::find (string *MPath*, XMLNode * *startNode* = 0)

Searches through nodes for a path (I.E.: "/html/body/p"). As of now:

- Only tag nodes are searched!
- You con optionally place a wildcard (*) at the end of the string.

Result is stored in the answer vector.

Definition at line 756 of file Utils.cpp.

References answer, and Utils::XMLDoc::rootNode.

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

- Utils.h
- Utils.cpp

Chapter 8

Syntopia Page Documentation

8.1 Todo List

Member SynthCore::BiQuad::setDry(float s) consider the soundness of this.

Member SynthCore::BiQuad::HACK clean this

Member GUIToolkit::Component::draw() what uses this?

Class GUIToolkit::DBCanvas Obsolete?

Class SynthCore::EnvData The loop point is unused for now.

Member SynthCore::EnvData::getValue(long &time, bool loop) This also depends on an interpolation method. Should be fixed.

Member SynthGUI::EnvDialog::~EnvDialog() create this

Class SynthCore::Envelope3 check the two modes.

Class GUIToolkit::EventHandler should it be purely virtual?

Member SynthCore::Fourier::transform(float data[, unsigned long nn, int isign)] clean up wrappers.

Member SynthCore::FreqTable::getMaxPartialsFromFreq(float freq) Only correct at 48KHz!

Member GUIToolkit::GUIControl::GUIControl(HINSTANCE my_hInstance, int my_nCmdShow) add apropriate default constructor.

Member GUIToolkit::GUIControl::my_itos(int i) Obsolete.

Member GUIToolkit::GUIControl::RegisterRaw(Component *m) is this the way?

Member SynthGUI::mainDialog::drawMod(int x, int y, int x2, int y2, std::string s, mod *md, bool draw privatize?

Member SynthGUI::mainDialog::mouseOverMod(int x, int y) privatize?

Member GUIToolkit::menuItem::ID privatize

Member GUIToolkit::menuItem::Label privatize

Member SynthGUI::mod::InOuts unused?

Class SynthGUI::modInOuts obsolete?

Member SynthGUI::modList::addModule(Module *m) fix this.

Class SynthCore::Module consider the RTTI reflection.

Member SynthCore::Module::newMidiEvent() Make a better reflection implementation (RTTI like)

Member SynthCore::Module::hostVoice Change HostSynth to HostVoice

Class SynthCore::Multiplier Oversampling? Should it be possible to take absolute value of one (or both) of the signals?

Member SynthCore::Output::ConnectTo(Input *I) Remove this!

Member SynthCore::Polyphony::setVoices(int number) not finished yet

Member GUIToolkit::popupMenu::popupMenu(HWND hw) should be called with something else than a HWND

Class SynthCore::Synth needs some cleaning. Midi section needs improvement: remember velocity, pedal on/off, and pitch bend.

Member SynthCore::Synth::noteOn(int note) velocity

Member SynthCore::Synth::voices[4] Change to vector!

Class SynthCore::SynthEngine not really implemented yet.

Class SynthCore::SynthVoice needs serious rewriting: must have dynamical routing!

Class SynthCore::WaveShaper implement oversampling in order to cope with oversampling.

Member GUIToolkit::WinForm::hwnd privatize?

Member SynthCore::ModuleMap Why do I have to define these two times: Why can't the compiler use the definitions in "Synth.h"? I dont get it.

Index