The Open Master Hearing Aid (openMHA)

4.5.0

Plugin Developers' Manual

The Open Master Hearing Aid (openMHA) – Plugin Developers' Manual

HörTech gGmbH
Marie-Curie-Str. 2
D–26129 Oldenburg

Contents

1 Overview

The HörTech Open Master Hearing Aid (openMHA), is a development and evaluation software platform that is able to execute hearing aid signal processing in real-time on standard computing hardware with a low delay between sound input and output.

1.1 Structure

The openMHA can be split into four major components :

- **The openMHA command line application (MHA)** (p. 33)
- **Signal processing plugins** (p. 6)
- Audio input-output (IO) plugins (see **io_file_t** (p. 100), **MHAIOJack** (p. 71), **io_parser_t** (p. 102), **io_tcp_parser_t** (p. 105))
- **The openMHA toolbox library** (p. 34)

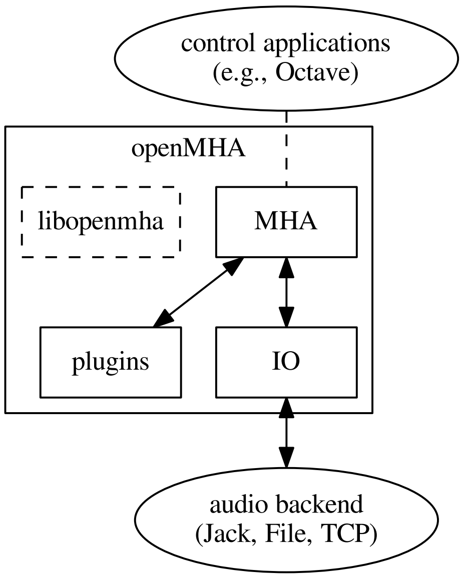


Figure 1 openMHA structure

The openMHA command line application (MHA) (p. 33) acts as a plugin host. It can load signal processing plugins as well as audio input-output (IO) plugins. Additionally, it provides the command line configuration interface and a TCP/IP based configuration interface. Several IO plugins exist: For real-time signal processing, commonly the openMHA **MHAIOJack** (p. 71) plugin (see plugins' manual) is used, which provides an interface to the Jack Audio Connection Kit (JACK). Other IO plugins provide audio file access or TCP/IP-based processing.

openMHA plugins (p. 6) provide the audio signal processing capabilities and audio signal handling. Typically, one openMHA plugin implements one specific algorithm. The complete virtual hearing aid signal processing can be achieved by a combination of several openMHA plugins.

1.2 Platform Services and Conventions

The openMHA platform offers some services and conventions to algorithms implemented in plugins, that make it especially well suited to develop hearing aid algorithms, while still supporting general-purpose signal processing.

1.2.1 Audio Signal Domains

As in most other plugin hosts, the audio signal in the openMHA is processed in audio chunks. However, plugins are not restricted to propagate audio signal as blocks of audio samples in the time domain another option is to propagate the audio signal in the short time Fourier transform (STFT) domain, i.e. as spectra of blocks of audio signal, so that not every plugin has to perform its own STFT analysis and synthesis. Since STFT analysis and re-synthesis of acceptable audio quality always introduces an algorithmic delay, sharing STFT data is a necessity for a hearing aid signal processing platform, because the overall delay of the complete processing has to be as short as possible.

Similar to some other platforms, the openMHA allows also arbitrary data to be exchanged between plugins through a mechanism called **algorithm communication variables** (p. 27) or short "AC vars". This mechanism is commonly used to share data such as filter coefficients or filter states.

1.2.2 Real-Time Safe Complex Configuration Changes

Hearing aid algorithms in the openMHA can export configuration settings that may be changed by the user at run time.

To ensure real-time safe signal processing, the audio processing will normally be done in a signal processing thread with real-time priority, while user interaction with configuration parameters would be performed in a configuration thread with normal priority, so that the audio processing does not get interrupted by configuration tasks. Two types of problems may occur when the user is changing parameters in such a setup:

- The change of a simple parameter exposed to the user may cause an involved recalculation of internal runtime parameters that the algorithm actually uses in processing. The duration required to perform this recalculation may be a significant portion of (or take even longer than) the time available to process one block of audio signal. In hearing aid usage, it is not acceptable to halt audio processing for the duration that the recalculation may require.
- If the user needs to change multiple parameters to reach a desired configuration state of an algorithm from the original configuration state, then it may not be acceptable that processing is performed while some of the parameters have already been changed while others still retain their original values. It is also not acceptable to interrupt signal processing until all pending configuration changes have been performed.

The openMHA provides a mechanism in its toolbox library to enable real-time safe configuration changes in openMHA plugins:

Basically, existing runtime configurations are used in the processing thread until the work of creating an updated runtime configuration has been completed in the configuration thread.

In hearing aids, it is more acceptable to continue to use an outdated configuration for a few more milliseconds than blocking all processing.

The openMHA toolbox library provides an easy-to-use mechanism to integrate real-time safe runtime configuration updates into every plugin.

1.2.3 Plugins can Themselves Host Other Plugins

An openMHA plugin can itself act as a plugin host. This allows to combine analysis and re-synthesis methods in a single plugin. We call plugins that can themselves load other plugins “bridge plugins" in the openMHA.

When such a bridge plugin is then called by the openMHA to process one block of signal, it will first perform its analysis, then invoke (as a function call) the signal processing in the loaded plugin to process the block of signal in the analysis domain, wait to receive a processed block of signal in the analysis domain back from the loaded plugin when the signal processing function call to that plugin returns, then perform the re-synthesis transform, and finally return the block of processed signal in the original domain back to the caller of the bridge plugin.

1.2.4 Central Calibration

The purpose of hearing aid signal processing is to enhance the sound for hearing impaired listeners. Hearing impairment generally means that people suffering from it have increased hearing thresholds, i.e. soft sounds that are audible for normal hearing listeners may be imperceptible for hearing impaired listeners. To provide accurate signal enhancement for hearing impaired people, hearing aid signal processing algorithms have to be able to determine the absolute physical sound pressure level corresponding to a digital signal given to any openM↩ HA plugin for processing. Inside the openMHA, we achieve this with the following convention: The single-precision floating point time-domain sound signal samples, that are processed inside the openMHA plugins in blocks of short durations, have the physical pressure unit Pascal ($1\mathrm{Pa} = 1\mathrm{N}/\mathrm{m}^2$). With this convention in place, all plugins can determine the absolute physical sound pressure level from the sound samples that they process. A derived convention is employed in the spectral domain for STFT signals. Due to the dependency of the calibration on the hardware used, it is the responsibility of the user of the openMHA to perform calibration measurements and adapt the openMHA settings to make sure that this calibration convention is met. We provide the plugin `transducers` which can be configured to perform the necessary signal adjustments.

2 Module Documentation

2.1 Concept of Variables and Data Exchange in the openMHA

Accessibility of configuration variables and data exchange between plugins (processing blocks) are an important issue in the openMHA.

In general, variable types in the openMHA are distinguished by their different access methods. The variable types in the openMHA are:

- **Configuration variables** : Read and write accesses are possible through the openM↩ HA configuration language interface. Configuration variables are implemented as C++ classes with a public data member of the underlying C type. Configuration variables can be read and modified from “outside" using the configuration language. The plugin which provides the configuration variable can use the exposed data member directly. All accesses through the openMHA configuration language are checked for data type, valid range, and access restrictions.
- **Monitor variables** : Read access is possible through the openMHA configuration language. Write access is only possible from the C++ code. Internally, monitor variables have a similar C++ class interface as configuration variables.
- **AC variables** (**algorithm communication variables** (p. 27)): Any C or C++ data structure can be shared within an openMHA chain. Access management and name space is realised in openMHA chain plugin ('mhachain'). AC variables are not available to the openMHA configuration language interface, although a read-only converter plugin `acmon` is available.
- **Runtime configuration** : Algorithms usually derive more parameters (runtime configuration) from the openMHA configuration language variables. When a configuration variable changes through configuration language write access, then the runtime configuration has to be recomputed. Plugin developers are encouraged to encapsulate the runtime configuration in a C++ class, which recomputes the runtime configuration from configuration variables in the constructor. The openMHA supports lock-free and thread-safe replacement of the runtime configuration instance (see **example5.cpp** (p. 20)) and references therein).

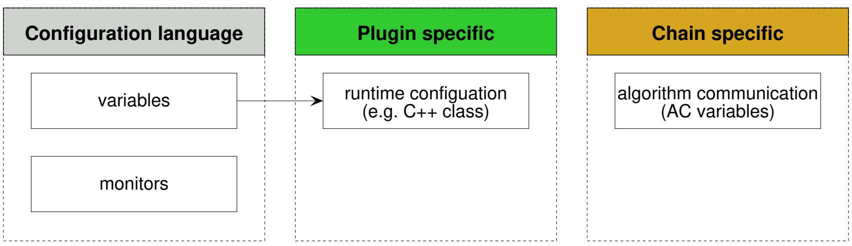


Figure 2 Variable types in the openMHA

The C++ data types are shown in the figure below. These variables can be accessed via the openMHA host application using the openMHA configuration language. For more details see 'Application engineers' manual'.

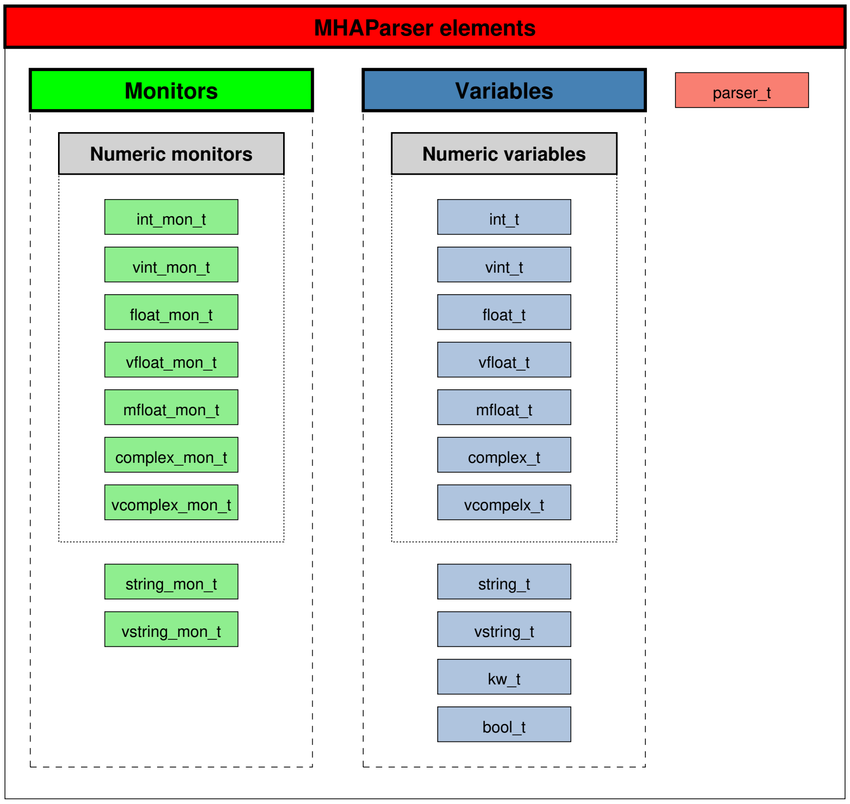


Figure 3 MHAParser elements

2.2 The openMHA Plugins (programming interface)

An openMHA plugin is the signal processing unit, usually an algorithm.

Classes

- class **MHAPlugin::plugin_t< runtime_cfg_t >**
  The template class for C++ openMHA plugins.

Macros

- #define **MHAPLUGIN_CALLBACKS_PREFIX**(prefix, classname, indom, outdom)
  C++ wrapper macro for the plugin interface.
- #define **MHAPLUGIN_DOCUMENTATION_PREFIX**(prefix, cat, doc)
  Wrapper macro for the plugin documentation interface.
- #define **MHAPLUGIN_CALLBACKS**(plugname, classname, indom, outdom) **MHAPLUGIN_CALLBACKS_PREFIX**(MHA_STATIC_ ## plugname ## _,classname,indom,outdom)
  C++ wrapper macro for the plugin interface.
- #define **MHAPLUGIN_DOCUMENTATION**(plugname, cat, doc) **MHAPLUGIN_DOCUMENTATION_PREFIX**(MHA_STATIC_ ## plugname ## _,cat,doc)
  Wrapper macro for the plugin documentation interface.

2.2.1 Detailed Description

openMHA plugins can be combined into processing chains. One of the configured chains can be selected for output which allows direct comparison of single algorithms or complex signal processing configurations. Algorithms within one chain can communicate with each other by sharing some of their variables, see section **Communication between algorithms** (p. 27).

The openMHA plugins can use the openMHA configuration language for their configuration. If they do so, the configuration can be changed through the framework even at run time. A description of this language can be found in section **The openMHA configuration language** (p. 33). If the algorithms should make use of the openMHA configuration language, they need to be written in C++ rather than pure C.

In the openMHA package a set of example plugins is included. These examples are the base of a step by step tutorial on how to write an openMHA plugin. See section **Writing openMHA Plugins. A step-by-step tutorial** (p. 10) for detailes.

openMHA plugins communicate with the openMHA using a simple ANSI-C interface. This way it is easy to mix plugins compiled with different C++ compilers. For convenience, we provide C++ classes which can be connected to the C++ interface. We strongly recommend the usage of these C++ wrappers. They include out-of-the box support exporting variables to the configuration interface and for thread safe configuration update.

The openMHA C++ plugin interface consists of a few number of method prototypes:

The output domain (spectrum or waveform) of an openMHA plugin will typically be the same as the input domain:

- **mha_wave_t** (p. 154) $*$ process(**mha_wave_t** (p. 154) $*$): pure waveform processing
- **mha_spec_t** (p. 141) $*$ process(**mha_spec_t** (p. 141) $*$): pure spectral processing

But it is also possible to implement domain transformations (from the time domain into spectrum or vice versa). The corresponding method signatures are:

- **mha_spec_t** (p. 141) $*$ process(**mha_wave_t** (p. 154) $*$): Domain transformation from waveform to spectrum
- **mha_wave_t** (p. 154) $*$ process(**mha_spec_t** (p. 141) $*$): Domain transformation from spectrum to waveform

For preparation and release of a plugin, the methods

- void prepare(**mhaconfig_t** (p. 155) &) and
- void release(void)

have to be implemented. The openMHA will call the `process()` method only ater the prepare method has returned and before `release()` is invoked. It is guarantteed by the open↩ MHA framework that signal processing is performed only between calls of `prepare()` and `release()`. Each call of `prepare()` is followed by a call of `release()` (after some optional signal processing).

For configuration purposes, the plugin class has to export a method called `parse()` which implements the openMHA configuration language. We strongly recommend that you do not implement this method yourself, but by inheriting from the class **MHAParser::parser_t** (p. 220) from the openMHA toolbox, directly or indirectly (inheriting from a class that itself inherits from **MHAParser::parser_t** (p. 220)).

2.2.2 Connecting the C++ class with the C Interface

A C++ class which provides the appropriate methods can be used as an openMHA Plugin by connecting it to the C interface using the **MHAPLUGIN_CALLBACKS** (p. 9) macro.

The openMHA Toolbox library provides a base class **MHAPlugin::plugin_t** (p. 238)$<$T$>$ (a template class) which can be used as the base class for a plugin class. This base class implements some necessary features for openMHA plugin developers like integration into the openMHA configuration language environment (it inherits from **MHAParser::parser_t** (p. 220)) and thread-safe runtime configuration update.

2.2.3 Error reporting

When your plugin detects a situation that it cannot handle, like input signal of the wrong signal domain at preparation time, unsupported number of input channels at preparation time, unsupported combinations of values in the plugin's variables during configuration, it should throw a C++ exception. The exception should be of type MHAError. Exceptions of this type are caught by the **MHAPLUGIN_CALLBACKS** (p. 9) macro for further error Reporting.

Throwing exceptions in response to unsupported configuration changes does not stop the signal processing. The openMHA configuration language parser will restore the previous value of that variable and report an error to the configurator, while the signal processing continues. Throwing exceptions from the signal processing thread will terminate the signal processing. Therefore, you should generally avoid throwing exceptions from the process method. Only do this if you detected a defect in your plugin, and then you should include enough information in the error message to be able to fix the defect.

2.2.4 Contents of the openMHA Plugin programming interface

2.2.5 Macro Definition Documentation

2.2.5.1 #define MHAPLUGIN_CALLBACKS_PREFIX(*prefix*, *classname*, *indom*, *outdom*)

Parameters

classname	The name of the plugin class
indom	Input domain (`wave` or `spec`)
outdom	Output domain (`wave` or `spec`)

This macro defines all required openMHA Plugin interface functions and passes calls of these functions to the corresponding member functions of the class `'classname'`. The parameters `'indom'` and `'outdom'` specify the input and output domain of the processing method. The `MHAInit()` and `MHADestroy()` functions will create or destroy an instance of the class. The approriate member functions have to be defined in the class. It is suggested to make usage of the **MHAPlugin::plugin_t** (p. 238) template class. Exceptions of type **MHA_Error** (p. 132) are caught and transformed into apropriate error codes with their corresponding error messages.

2.2.5.2 #define MHAPLUGIN_DOCUMENTATION_PREFIX(*prefix*, *cat*, *doc*)

Parameters

cat	Space separated list of categories to which belong the plugin (as const char*)
doc	Documentation of the plugin (as const char*)

This macro defines the openMHA Plugin interface function for the documentation. The categories can be any space seperated list of category names. An empty string will categorize the plugin in the category 'other'.

The documentation should contain a description of the plugin including a description of the underlying models, and a paragraph containing hints for usage. The text should be LaTeX compatible (e.g., avoid or quote underscores in the text part); equations should be formatted as LaTeX.

2.2.5.3 #define MHAPLUGIN_CALLBACKS(*plugname,* *classname,* *indom,* *outdom*) **MHAPLUGIN_CALLBACKS_PREFIX**(MHA_STATIC_ ## plugname ## _,classname,indom,outdom)

Parameters

plugname	The file name of the plugin without the .so or .dll extension
classname	The name of the plugin class
indom	Input domain (`wave` or `spec`)
outdom	Output domain (`wave` or `spec`)

This macro defines all required openMHA Plugin interface functions and passes calls of these functions to the corresponding member functions of the class ‘classname'. The parameters ‘indom' and ‘outdom' specify the input and output domain of the processing method. The MHA↩ Init() and MHADestroy() functions will create or destroy an instance of the class. The approriate member functions have to be defined in the class. It is suggested to make usage of the **MH↩ APlugin::plugin_t** (p. 238) template class. Exceptions of type **MHA_Error** (p. 132) are caught and transformed into apropriate error codes with their corresponding error messages.

2.2.5.4 #define MHAPLUGIN_DOCUMENTATION(*plugname,* *cat,* *doc*) MHA↩ PLUGIN_DOCUMENTATION_PREFIX(MHA_STATIC_ ## plugname ## _,cat,doc)

Parameters

plugin	The file name of the plugin without the .so or .dll extension
cat	Space separated list of categories to which belong the plugin (as const char∗)
doc	Documentation of the plugin (as const char∗)

This macro defines the openMHA Plugin interface function for the documentation. The categories can be any space seperated list of category names. An empty string will categorize the plugin in the category 'other'.

The documentation should contain a description of the plugin including a description of the underlying models, and a paragraph containing hints for usage. The text should be LaTeX compatible (e.g., avoid or quote underscores in the text part); equations should be formatted as LaTeX.

2.3 Writing openMHA Plugins. A step-by-step tutorial

A step-by-step tutorial on writing openMHA plugins.

openMHA contains a small number of example plugins as C++ source code. They are meant to help developers in understanding the concepts of openMHA plugin programming starting from the simplest example and increasing in complexity. This tutorial explains the basic parts of the example files.

2.3.1 example1.cpp

The example plugin file `example1.cpp` demonstrates the easiest way to implement an openMHA Plugin. It attenuates the sound signal in the first channel by multiplying the sound samples with a factor. The plugin class **MHAPlugin::plugin_t** (p. 238) exports several methods, but only two of them need a non-empty implementation: `prepare()` method is a pure virtual function and `process()` is called when signal processing starts.

```
#include "mha_plugin.hh"

class example1_t : public MHAPlugin::plugin_t<int> {
public:
  example1_t(algo_comm_t & ac,
             const std::string & chain_name,
             const std::string & algo_name)
      : MHAPlugin::plugin_t<int>("",ac)
  {/* Do nothing in constructor */}

  void release(void)
  {/* Do nothing in release */}
```

Every plugin implementation should include the '**mha_plugin.hh** (p. 295)' header file. C++ helper classes for plugin development are declared in this header file, and most header files needed for plugin development are included by **mha_plugin.hh** (p. 295).

The class plugin1_t inherits from the class **MHAPlugin::plugin_t** (p. 238), which then inherits from **MHAParser::parser_t** (p. 220) – the configuration language interface in the method "parse". Our plugin class therefore exports the working "parse" method inherited from **MHAParser::parser_t** (p. 220), and the plugin is visible in the openMHA configuration tree.

The constructor has to accept 3 parameters of correct types. In this simple example, we do not make use of them.

The `release()` method is used to free resources after signal processing. In this simple example, we do not allocate resources, so there is no need to free them.

2.3.1.1 The prepare method

```
  void prepare(mhaconfig_t & signal_info)
  {
    if (signal_info.domain != MHA_WAVEFORM)
      throw MHA_Error(__FILE__, __LINE__,
                      "This plugin can only process waveform signals.");
    if (signal_info.channels < 1)
      throw MHA_Error(__FILE__,__LINE__,
                      "This plugin requires at least one input channel.");
  }
```

Parameters

signal_info	Contains information about the input signal's parameters, see **mhaconfig_t** (p. 155).

The `prepare()` method of the plugin is called before the signal processing starts, when the input signal parameters like domain, number of channels, frames per block, and sampling rate are known. The `prepare()` method can check these values and raise an exception if the plugin cannot cope with them, as is done here. The plugin can also change these values if the signal processing performed in the plugin results in an output signal with different parameters. This plugin does not change the signal's parameters, therefore they are not modified here.

2.3.1.2 The signal processing method

```
  mha_wave_t * process(mha_wave_t * signal)
  {
    unsigned int channel = 0; // channels and frames counting starts with 0
    float factor = 0.1f;
    unsigned int frame;

    // Scale channel number "channel" by "factor":
    for(frame = 0; frame < signal->num_frames; frame++) {
      // Waveform channels are stored interleaved.
      signal->buf[signal->num_channels * frame + channel] *= factor;
    }
    // Algorithms may process data in-place and return the input signal
    // structure as their output signal:
    return signal;
  }
};
```

Parameters

signal	Pointer to the input signal structure **mha_wave_t** (p. 154).

Returns

Pointer to the output signal structure. The input signal structure may be reused if the signal has the same domain and dimensions.

The plugin works with time domain input signal (indicated by the data type **mha_wave_t** (p. 154) of the process method's parameter). It scales the first channel by a factor of 0.1. The output signal reuses the structure that previously contained the input signal (in-place processing).

2.3.1.3 Connecting the C++ class with the C plugin interface

Plugins have to export C functions as their interface (to avoid C++ name-mangling issues and other incompatibilities when mixing plugins compiled with different C++ compilers).

```
MHAPLUGIN_CALLBACKS(example1,example1_t,wave,wave)
```

This macro takes care of accessing the C++ class from the C functions required as the plugin's interface. It implements the C funtions and calls the corresponding C++ instance methods. Plugin classes should be derived from the template class **MHAPlugin::plugin_t** (p. 238) to be compatible with the C interface wrapper.

This macro also catches C++ exceptions of type **MHA_Error** (p. 132), when raised in the methods of the plugin class, and reports the error using an error flag as the return value of the underlying C function. It is therefore important to note that only C++ exceptions of type **MHA_Error** (p. 132) may be raised by your plugin. If your code uses different Exception classes, you will have to catch them yourself before control leaves your plugin class, and maybe report the error by throwing an instance of **MHA_Error** (p. 132). This is important, because: (1) C++ exceptions cannot cross the plugin interface, which is in C, and (2) there is no error handling code for your exception classes in the openMHA framework anyways.

2.3.2 example2.cpp

This is another simple example of openMHA plugin written in C++. This plugin also scales one channel of the input signal, working in the time domain. The scale factor and which channel to scale (index number) are made accessible to the configuration language.

The algorithm is again implemented as a C++ class.

```
class example2_t : public MHAPlugin::plugin_t<int> {
    MHAParser::int_t scale_ch;
    MHAParser::float_t factor;
public:
    example2_t(algo_comm_t & ac,
               const std::string & chain_name,
               const std::string & algo_name);

    void prepare(mhaconfig_t & signal_info);

    void release(void);

    mha_wave_t * process(mha_wave_t * signal);
};
```

Parameters

scale_ch	– the channel number to be scaled
factor	– the scale factor of the scaling.

This class again inherits from the template class **MHAPlugin::plugin_t** (p. 238) for intergration with the openMHA configuration language. The two data members serve as externally visible configuration variables. All methods of this class have a non-empty implementation.

2.3.2.1 Constructor

```
example2_t::example2_t(algo_comm_t & ac,
                       const std::string & chain_name,
```

```
                    const std::string & algo_name)
    : MHAPlugin::plugin_t<int>("This plugin multiplies the sound signal"
                               " in one audio channel by a factor",ac),
      scale_ch("Index of audio channel to scale. Indices start from 0.",
               "0",
               "[0,["),
      factor("The scaling factor that is applied to the selected channel.",
             "0.1",
             "[0,[")
{
    insert_item("channel", &scale_ch);
    insert_item("factor", &factor);
}
```

The constructor invokes the superclass constructor with a string parameter. This string parameter serves as the help text that describes the functionality of the plugin. The constructor registers configuration variables with the openMHA configuration tree and sets their default values and permitted ranges. The minimum permitted value for both variables is zero, and there is no maximum limit (apart from the limitations of the underlying C data type). The configuration variables have to be registered with the parser node instance using the **MHAParser::parser_t::insert_item** (p. 222) method.

2.3.2.2 The prepare method

```
void example2_t::prepare(mhaconfig_t & signal_info)
{
    if (signal_info.domain != MHA_WAVEFORM)
        throw MHA_Error(__FILE__, __LINE__,
                        "This plugin can only process waveform signals.");
    // The user may have configured scale_ch before prepare is called.
    // Check that the configured channel is present in the input signal.
    if (signal_info.channels <= unsigned(scale_ch.data))
        throw MHA_Error(__FILE__,__LINE__,
                        "This plugin requires at least %d input channels.",
                        scale_ch.data + 1);
    // Adjust the range of the channel configuration variable so that it
    // cannot be set to an out-of-range value during processing.
    using MHAParser::StrCnv::val2str;
    scale_ch.set_range("[0," + val2str(int(signal_info.channels)) + "[");
}
```

Parameters

signal_info	– contains information about the input signal's parameters, see **mhaconfig_t** (p. 155).

The user may have changed the configuration variables before preparing the openMHA plugin. A consequence of this is that it is not sufficient any more to check if the input signal has at least 1 audio channel.

Instead, this prepare method checks that the input signal has enough channels so that the current value of `scale_ch.data` is a valid channel index, i.e. $0 \leq$ `scale_ch.data` $<$ `signal_info.channels`. The prepare method does not have to check that $0 \leq$ `scale_ch.data`, since this is guaranteed by the valid range setting of the configuration variable.

The prepare method then modifies the valid range of the `scale_ch` variable, it modifies the upper bound so that the user cannot set the variable to a channel index higher than the available channels. Setting the range is done using a string parameter. The prepare method contatenates a string of the form "[0,n[". n is the number of channels in the input signal, and is used here as an exclusive upper boundary. To convert the number of channels into a string, a helper function for string conversion from the openMHA Toolbox is used. This function is overloaded and works for several data types.

It is safe to assume that the value of configuration variables does not change while the prepare method executes, since openMHA preparation is triggered from a configuration language command, and the openMHA configuration language parser is busy and cannot accept other commands until all openMHA plugins are prepared (or one of them stops the process by raising an exception). As we will see later in this tutorial, the same assumption cannot be made for the process method.

2.3.2.3 The release method

```
void example2_t::release(void)
{
    scale_ch.set_range("[0,[");
}
```

The release method should undo the state changes that were performed by the prepare method. In this example, the prepare method has reduced the valid range of the `scale_ch`, so that only valid channels could be selected during signal processing.

The release method reverts this change by setting the valid range back to its original value, "[0,[".

2.3.2.4 The signal processing method

```
mha_wave_t * example2_t::process(mha_wave_t * signal)
{
    unsigned int frame;
    for(frame = 0; frame < signal->num_frames; frame++)
        value(signal,frame,scale_ch.data) *= factor.data;
    return signal;
}
```

The processing function uses the current values of the configuration variables to scale every frame in the selected audio channel.

Note that the value of each configuration variable can change while the processing method executes, since the process method usually executes in a different thread than the configuration interface.

For this simple plugin, this is not a problem, but for more advanced plugins, it has to be taken into consideration. The next section takes a closer look at the problem.

Consistency

Assume that one thread reads the value stored in a variable while another thread writes a new value to that variable concurrently. In this case, you may have a consistency problem. You would perhaps expect that the value retrieved from the variable either (a) the old value, or (b) the new value, but not (c) something else. Yet generally case (c) is a possibility.

Fortunately, for some data types on PC systems, case (c) cannot happen. These are 32bit wide data types with a 4-byte alignment. Therefore, the values in **MHAParser::int_t** (p. 211) and **MHAParser::float_t** (p. 208) are always consistent, but this is not the case for vectors, strings, or complex values. With these, you can get a mixture of the bit patterns of old and new values, or you can even cause a memory access violation in case a vector or string grows and has to be reallocated to a different memory address.

There is also a consistency problem if you take the combination of two "safe" datatypes. The openMHA provides a mechanism that can cope with these types of problems. This thread-safe runtime configuration update mechanism is introduced in example 5.

2.3.3 example3.cpp

This example introduces the openMHA Event mechanism. Plugins that provide configuration variable can receive a callback from the parser base class when a configuration variable is accessed through the configuration language interface.

The third example performes the same processing as before, but now only even channel indices are permitted when selecting the audio channel to scale. This restriction cannot be ensured by setting the range of the channel index configuration variable. Instead, the event mechanism of openMHA configuration variables is used. Configuration variables emit 4 different events, and your plugin can connect callback methods that are called when the events are triggered. These events are:

writeaccess

- triggered on write access to a configuration variable.

valuechanged

- triggered when write access to a configuration variable actually changes the value of this variable.

readaccess

- triggered after the value of the configuration variable has been read.

prereadaccess

- triggered before the value of a configuration variable is read, i.e. the value of the requested variable can be changed by the callback to implement computation on demand.

All of these callbacks are executed in the configuration thread. Therefore, the callback implementation does not have to be realtime-safe. No other updates of configuration language variables through the configuration language can happen in parallel, but your processing method can execute in parallel and may change values.

2.3.3.1 Data member declarations

```
class example3_t : public MHAPlugin::plugin_t<int> {
    MHAParser::int_t scale_ch;
    MHAParser::float_t factor;
    MHAParser::int_mon_t prepared;

    MHAEvents::patchbay_t<example3_t> patchbay;
```

This plugin exposes another configuration variable, "`prepared`", that keeps track of the prepared state of the plugin. This is a read-only (monitor) integer variable, i.e. its value can only be changed by your plugin's C++ code. When using the configuration language interface, the value of this variable can only be read, but not changed.

The patchbay member is an instance of a connector class that connects event sources with callbacks.

2.3.3.2 Method declarations

```
    /* Callbacks triggered by Events */
    void on_scale_ch_writeaccess();
    void on_scale_ch_valuechanged();
    void on_scale_ch_readaccess();
    void on_prereadaccess();
public:
    example3_t(algo_comm_t & ac,
               const std::string & chain_name,
               const std::string & algo_name);

    void prepare(mhaconfig_t & signal_info);

    void release(void);

    mha_wave_t * process(mha_wave_t * signal);
};
```

This plugin exposes 4 callback methods that are triggered by events. Multiple events (from the same or different configuration variables) can be connected to the same callback method, if desired.

This example plugin uses the `valuechanged` event to check that the `scale_ch` configuration variable is only set to valid values.

The other callbacks only cause log messages to stdout, but the comments in the logging callbacks give a hint when listening on the events would be useful.

2.3.3.3 Example 3 constructor

```
example3_t::example3_t(algo_comm_t & ac,
                       const std::string & chain_name,
                       const std::string & algo_name)
    : MHAPlugin::plugin_t<int>("This plugin multiplies the sound signal"
                               " in one audio channel by a factor",ac),
      scale_ch("Index of audio channel to scale. Indices start from 0."
               " Only channels with even indices may be scaled.",
               "0",
               "[0,["),
      factor("The scaling factor that is applied to the selected channel.",
             "0.1",
             "[0,["),
      prepared("State of this plugin: 0 = unprepared, 1 = prepared")
{
    insert_item("channel", &scale_ch);
    insert_item("factor", &factor);
    prepared.data = 0;
    insert_item("prepared", &prepared);

    patchbay.connect(&scale_ch.writeaccess, this,
                     &example3_t::on_scale_ch_writeaccess);
    patchbay.connect(&scale_ch.valuechanged, this,
                     &example3_t::on_scale_ch_valuechanged);
    patchbay.connect(&scale_ch.readaccess, this,
                     &example3_t::on_scale_ch_readaccess);
    patchbay.connect(&scale_ch.prereadaccess, this,
                     &example3_t::on_prereadaccess);
    patchbay.connect(&factor.prereadaccess, this,
                     &example3_t::on_prereadaccess);
    patchbay.connect(&prepared.prereadaccess, this,
                     &example3_t::on_prereadaccess);
}
```

The constructor of monitor variables does not take a parameter for setting the initial value. The single parameter here is the help text describing the contents of the read-only variable. If the initial value should differ from 0, then the `.data` member of the configuration variable has to be set to the initial value in the plugin constructor's body explicitly, as is done here for demonstration although the initial value of this monitor variable is 0.

Events and callback methods are then connected using the patchbay member variable.

2.3.3.4 The prepare method

```
void example3_t::prepare(mhaconfig_t & signal_info)
{
    if (signal_info.domain != MHA_WAVEFORM)
        throw MHA_Error(__FILE__, __LINE__,
                        "This plugin can only process waveform signals.");
    // The user may have configured scale_ch before prepare is called.
    // Check that the configured channel is present in the input signal.
    if (signal_info.channels <= unsigned(scale_ch.data))
        throw MHA_Error(__FILE__,__LINE__,
                        "This plugin requires at least %d input channels.",
                        scale_ch.data + 1);
    // bookkeeping
    prepared.data = 1;
}
```

The prepare method checks wether the current setting of the scale_ch variable is possible with the input signal dimension. It does not adjust the range of the variable, since the range alone is not sufficient to ensure all future settings are also valid: The scale channel index has to be even.

2.3.3.5 The release method

```
void example3_t::release(void)
{
  prepared.data = 0;
}
```

The release method is needed for tracking the prepared state only in this example.

2.3.3.6 The signal processing method

```
mha_wave_t * example3_t::process(mha_wave_t * signal)
{
    unsigned int frame;
    for(frame = 0; frame < signal->num_frames; frame++)
        value(signal,frame,scale_ch.data) *= factor.data;
    return signal;
}
```

The signal processing member function is the same as in example 2.

2.3.3.7 The callback methods

```
void example3_t::on_scale_ch_writeaccess()
{
    printf("Write access: Attempt to set scale_ch=%d.\n", scale_ch.data);
    // Can be used to track any writeaccess to the configuration, even
    // if it does not change the value.  E.g. setting the name of the
    // sound file in a string configuration variable can cause a sound
    // file player plugin to start playing the sound file from the
    // beginning.
}
void example3_t::on_scale_ch_valuechanged()
{
    if (scale_ch.data & 1)
        throw MHA_Error(__FILE__,__LINE__,
                        "Attempt to set scale_ch to non-even value %d",
                        scale_ch.data);
    // Can be used to recompute a runtime configuration only if some
    // configuration variable actually changed.
}
void example3_t::on_scale_ch_readaccess()
{
    printf("scale_ch has been read.\n");
    // A configuration variable used as an accumulator can be reset
    // after it has been read.
}
void example3_t::on_prereadaccess()
{
    printf("A configuration language variable is about to be read.\n");
    // Can be used to compute the value on demand.
}

MHAPLUGIN_CALLBACKS(example3,example3_t,wave,wave)
```

When the `writeaccess` or `valuechanged` callbacks throw an MHAError exception, then the change made to the value of the configuration variable is reverted.

If multiple event sources are connected to a single callback method, then it is not possible to determine which event has caused the callback to execute. Often, this information is not crucial, i.e. when the answer to a change of any variable in a set of variables is the same, e.g. the recomputation of a new runtime configuration that takes all variables of this set as input.

2.3.4 example4.cpp

This plugin is the same as example 3 except that it works on the spectral domain (STFT).

2.3.4.1 The Prepare method

```
void example4_t::prepare(mhaconfig_t & signal_info)
{
    if (signal_info.domain != MHA_SPECTRUM)
        throw MHA_Error(__FILE__, __LINE__,
                        "This plugin can only process spectrum signals.");
    // The user may have configured scale_ch before prepare is called.
    // Check that the configured channel is present in the input signal.
    if (signal_info.channels <= unsigned(scale_ch.data))
        throw MHA_Error(__FILE__,__LINE__,
                        "This plugin requires at least %d input channels.",
                        scale_ch.data + 1);
    // bookkeeping
    prepared.data = 1;
}
```

The prepare method now checks that the signal domain is MHA_SPECTRUM.

2.3.4.2 The signal processing method

```
mha_spec_t * example4_t::process(mha_spec_t * signal)
{
    unsigned int bin;
    // spectral signal is stored non-interleaved.
    mha_complex_t * channeldata =
        signal->buf + signal->num_frames * scale_ch.data;
    for(bin = 0; bin < signal->num_frames; bin++)
        channeldata[bin] *= factor.data;
    return signal;
}
```

The signal processing member function works on the spectral signal instead of the wave signal as before.

The **mha_spec_t** (p. 141) instance stores the complex (**mha_complex_t** (p. 123)) spectral signal for positive frequences only (since the waveform signal is always real). The num_frames member of **mha_spec_t** (p. 141) actually denotes the number of STFT bins.

Please note that different from **mha_wave_t** (p. 154), a multichannel signal in **mha_spec_t** (p. 141) is stored non-interleaved in the signal buffer.

Some arithmetic operations are defined on struct **mha_complex_t** (p. 123) to facilitate efficient complex computations. The *= operator used here (defined for real and for complex arguments) is one of them.

2.3.4.3 Connecting the C++ class with the C plugin interface

```
MHAPLUGIN_CALLBACKS(example4,example4_t,spec,spec)
```

When connecting a class that performs spectral processing with the C interface, use `spec` instead of `wave` as the domain indicator.

2.3.5 example5.cpp

Many algorithms use complex operations to transform the user space variables into run time configurations. If this takes a noticeable time (e.g. more than 100-500 μ sec), the update of the runtime configuration can not take place in the real time processing thread. Furthermore, the parallel access to complex structures may cause unpredictable results if variables are read while only parts of them are written to memory (cf. section **Consistency** (p. 15)). To handle these situations, a special C++ template class **MHAPlugin::plugin_t** (p. 238) was designed. This class helps keeping all access to the configuration language variables in the **configuration** thread rather than in the **processing** thread.

The runtime configuration class `example5_t` is the parameter of the template class **MHAPlugin::plugin_t** (p. 238). Its constructor converts the user variables into a runtime configuration. Because the constructor executes in the configuration thread, there is no harm if the constructor takes a long time. All other member functions and data members of the runtime configurations are accessed only from the signal processing thread (real-time thread).

```
class example5_t {
public:
    example5_t(unsigned int,unsigned int,mha_real_t);
    mha_spec_t* process(mha_spec_t*);
private:
    unsigned int channel;
    mha_real_t scale;
};
```

The plugin interface class inherits from the plugin template class **MHAPlugin::plugin_t** (p. 238), parameterised by the runtime configuration. Configuration changes (write access to the variables) will emit a write access event of the changed variables. These events can be connected to member functions of the interface class by the help of a **MHAEvents::patchbay_t** (p. 156) instance.

```
class plugin_interface_t : public MHAPlugin::plugin_t<example5_t> {
public:
    plugin_interface_t(const algo_comm_t&,const std::string&,const std::string&);
    mha_spec_t* process(mha_spec_t*);
    void prepare(mhaconfig_t&);
private:
    void update_cfg();
    /* integer variable of MHA-parser: */
    MHAParser::int_t scale_ch;
    /* float variable of MHA-parser: */
    MHAParser::float_t factor;
    /* patch bay for connecting configuration parser
       events with local member functions: */
    MHAEvents::patchbay_t<plugin_interface_t> patchbay;
};
```

The constructor of the runtime configuration analyses and validates the user variables. If the configuration is invalid, an exception of type **MHA_Error** (p. 132) is thrown. This will cause the openMHA configuration language command which caused the change to fail: The modified configuration language variable is then reset to its original value, and the error message will contain the message string of the **MHA_Error** (p. 132) exception.

```
example5_t::example5_t(unsigned int ichannel,
                       unsigned int numchannels,
                       mha_real_t iscale)
    : channel(ichannel),scale(iscale)
{
    if( channel >= numchannels )
        throw MHA_Error(__FILE__,__LINE__,
                        "Invalid channel number %d (only %d channels configured).",
                        channel,numchannels);
}
```

In this example, the run time configuration class `example5_t` has a signal processing member function. In this function, the selected channel is scaled by the given scaling factor.

```
mha_spec_t* example5_t::process(mha_spec_t* spec)
{
    /* Scale channel number "scale_ch" by "factor": */
    for(unsigned int fr = 0; fr < spec->num_frames; fr++){
        spec->buf[fr + channel * spec->num_frames].re *= scale;
        spec->buf[fr + channel * spec->num_frames].im *= scale;
    }
    return spec;
}
```

The constructor of the example plugin class is similar to the previous examples. A callback triggered on write access to the variables is registered using the **MHAEvents::patchbay_t** (p. 156) instance.

```
plugin_interface_t::plugin_interface_t(
    const algo_comm_t& iac,
    const std::string&,const std::string&)
    : MHAPlugin::plugin_t<example5_t>("example plugin configuration structure",iac),
      /* initialzing variable 'scale_ch' with MHAParser::int_t(char* name, .... ) */
      scale_ch("channel number to be scaled","0","[0,["),
      /* initialzing variable 'factor' with MHAParser::float_t(char* name, .... ) */
      factor("scale factor","1.0","[0,2]")
{
    /* Register variables to the configuration parser: */
    insert_item("channel",&scale_ch);
    insert_item("factor",&factor);
    /*
     * On write access to the parser variables a notify callback of
     * this class will be called. That funtion will update the runtime
     * configuration.
     */
    patchbay.connect(&scale_ch.writeaccess,this,&plugin_interface_t::update_cfg);
    patchbay.connect(&factor.writeaccess,this,&plugin_interface_t::update_cfg);
}
```

The processing function can gather the latest valid runtime configuration by a call of `poll_config`. On success, the class member `cfg` points to this configuration. On error, if there is no usable runtime configuration instance, an exception is thrown. In this example, the prepare method ensures that there is a valid runtime configuration, so that in this example, no error can be raised at this point. The prepare method is always executed before the process method is called. The runtime configuration class in this example provides a signal processing method. The process method of the plugin interface calls the process method of this instance to perform the actual signal processing.

```
mha_spec_t* plugin_interface_t::process(mha_spec_t* spec)
{
    poll_config();
    return cfg->process(spec);
}
```

The prepare method ensures that a valid runtime configuration exists by creating a new runtime configuration from the current configuration language variables. If the configuraion is invalid, then an exception of type **MHA_Error** (p. 132) is raised and the preparation of the openMHA fails with an error message.

```
void plugin_interface_t::prepare(mhaconfig_t& tfcfg)
{
    if( tfcfg.domain != MHA_SPECTRUM )
        throw MHA_Error(__FILE__,__LINE__,
                        "Example5: Only spectral processing is supported.");
    /* remember the transform configuration (i.e. channel numbers): */
    tftype = tfcfg;
    /* make sure that a valid runtime configuration exists: */
    update_cfg();
}
```

The update_cfg member function is called when the value of a configuration language variable changes, or from the prepare method. It allocates a new runtime configuration and registers it for later access from the real time processing thread. The function **push_config** (p. 238) stores the configuration in a FiFo queue of runtime configurations. Once they are inserted in the FiFo, the **MHAPlugin::plugin_t** (p. 238) template is responsible for deleting runtime configuration instances stored in the FiFo. You don't need to keep track of the created instances, and you must not delete them yourself.

```
void plugin_interface_t::update_cfg()
{
    if( tftype.channels )
        push_config(new example5_t(scale_ch.data,tftype.channels,factor.data));
}
```

In the end of the example code file, the macro **MHAPLUGIN_CALLBACKS** (p. 9) defines all ANSI-C interface functions and passes them to the corresponding C++ class member functions (partly defined by the **MHAPlugin::plugin_t** (p. 238) template class). All exceptions of type **MHA_Error** (p. 132) are caught and transformed into an appropriate error code and error message.

```
MHAPLUGIN_CALLBACKS(example5,plugin_interface_t,spec,spec)
```

2.3.6 example6.cpp

This last example is the same as the previous one, but it additionally creates an 'Algorithm Communication Variable' (AC variable). It calculates the RMS level of a given channel and stores it into this variable. The variable can be accessed by any other algorithm in the same chain. To store the data onto disk, the 'acsave' plugin can be used. 'acmon' is a plugin which converts AC variables into parsable monitor variables.

In the constructor of the plugin class the variable `rmsdb` is registered under the name `example6_rmslev` as a one-dimensional AC variable of type float. For registration of other types, read access and other detailed informations please see **Communication between algorithms** (p. 27).

```
example6_t::example6_t(const algo_comm_t& iac,
                       const std::string&,const std::string&)
    : MHAPlugin::plugin_t<cfg_t>("example plugin configuration structure",iac),
      /* initialzing variable 'channel_no' with MHAParser::int_t(char* name, .... ) */
      channel_no("channel in which the RMS level is measured","0","[0,[")
{
    /* Register variables to the configuration parser: */
    insert_item("channel",&channel_no);
    /*
     * On write access to the parser variables a notify callback of
     * this class will be called. That funtion will update the runtime
     * configuration.
     */
    patchbay.connect(&channel_no.writeaccess,this,&example6_t::update_cfg);
    /*
     * Propagate the level variable to all algorithms in the
     * processing chain. If multiple instances of this algorithm are
     * required, than it is necessary to use different names for this
     * variable (i.e. prefixing the name with the algorithm name
     * passed to MHAInit).
     */
    ac.insert_var_float( ac.handle, "example6_rmslev", &rmsdb );
}
```

2.3.7 Debugging openMHA plugins

Suppose you would want to step through the code of your openMHA plugin with a debugger. This example details how to use the linux gdb debugger to inspect the `example6_t::prepare()` and `example6_t::process()` routines of **example6.cpp** (p. 23) example 6.

First, make sure that your plugin is compiled with the compiler option to include debugging symbols: Apply the -ggdb switch to all gcc, g++ invocations.

Once the plugin is compiled, with debugging symbols, create a test configuration. For example 6, assuming there is an audio file named input.wav in your working directory, you could create a configuration file named 'debugexample6.cfg', with the following content:

```
# debugexample6.cfg
fragsize = 64
srate = 44100
nchannels_in = 2
iolib = MHAIOFile

io.in = input.wav
io.out = output.wav
mhalib = example6
mha.channel = 1
cmd=start
```

Assuming all your binaries and shared-object libraries are in your 'bin' directory (see README.md), you could start gdb using

```
$ export MHA_LIBRARY_PATH=$PWD/bin
$ gdb $MHA_LIBRARY_PATH/mha
```

Set breakpoints in prepare and process methods, and start execution. Note that specifying the breakpoint by symbol (`example6_t::prepare`) does not yet work, as the symbol lives in the openMHA plugin that has not yet been loaded. Specifying by line number works, however. Specifying the breakpoint by symbol also works once the plugin is loaded (i.e. when the debugger stops in the first break point). You can set the breakpoints like this (example shown here is run in gdb version 7.11.1):

```
(gdb) run ?read:debugexample6.cfg
Starting program: {openMHA_directory}/bin/mha ?read:debugexample6.cfg
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The Open Master Hearing Aid (openMHA) server
Copyright (c) 2005-2017 HoerTech gGmbH, D-26129 Oldenburg, Germany

This program comes with ABSOLUTELY NO WARRANTY; for details see file COPYING.
This is free software, and you are welcome to redistribute it
under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE, Version 3;
for details see file COPYING.


Breakpoint 1, example6_t::prepare (this=0x6478b0, tfcfg=...)
    at example6.cpp:192
192         if( tfcfg.domain != MHA_WAVEFORM )
(gdb) b example6.cpp:162
Breakpoint 2 at 0x7ffff589744a: file example6.cpp, line 162.
(gdb) c
Continuing.
```

Where '{openMHA_directory}' is the directory where openMHA is located (which should also be your working directory in this case). Next stop is the `process()` method. You can now examine and change the variables, step through the program as needed (using, for example 'n' to step in the next line):

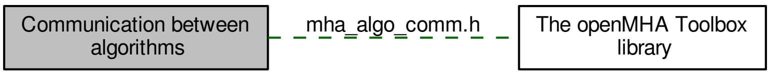
```
Breakpoint 2, example6_t::process (this=0x7ffff6a06c0d, wave=0x10a8b550)
    at example6.cpp:162
162     {
(gdb) n
163         poll_config();
(gdb)
```

2.4 The MHA Framework interface

2.5 Communication between algorithms

Algorithms within one chain can share variables for communication with other algorithms.

Collaboration diagram for Communication between algorithms:



Files

- file **mha_algo_comm.h**

  Header file for Algorithm Communication.

Namespaces

- **MHA_AC**

  Functions and classes for Algorithm Communication (AC) support.

Classes

- class **MHA_AC::spectrum_t**

  *Insert a **MHASignal::spectrum_t** (p. 261) class into the AC space.*
- class **MHA_AC::waveform_t**

  *Insert a **MHASignal::waveform_t** (p. 268) class into the AC space.*
- class **MHA_AC::int_t**

  Insert a integer variable into the AC space.
- class **MHA_AC::float_t**

  Insert a float point variable into the AC space.
- class **MHA_AC::double_t**

  Insert a double precision floating point variable into the AC space.
- class **MHA_AC::ac2matrix_t**

  Copy AC variable to a matrix.
- class **MHA_AC::acspace2matrix_t**

  Copy all or a subset of all numeric AC variables into an array of matrixes.
- struct **algo_comm_t**

  A reference handle for algorithm communication variables.
- struct **comm_var_t**

  Algorithm communication variable structure.

Functions

- **mha_spec_t MHA_AC::get_var_spectrum** (**algo_comm_t** ac, const std::string &name)
  Convert an AC variable into a spectrum.
- **mha_wave_t MHA_AC::get_var_waveform** (**algo_comm_t** ac, const std::string &name)
  Convert an AC variable into a waveform.
- int **MHA_AC::get_var_int** (**algo_comm_t** ac, const std::string &name)
  Return value of an integer scalar AC variable.
- float **MHA_AC::get_var_float** (**algo_comm_t** ac, const std::string &name)
  Return value of an floating point scalar AC variable.
- std::vector< float > **MHA_AC::get_var_vfloat** (**algo_comm_t** ac, const std::string &name)
  Return value of an floating point vector AC variable as standard vector of floats.

2.5.1 Detailed Description

This mechanism allows interaction between algorithms (i.e. separation of noise estimation and noise reduction algorithms, combination of dynamic compression and noise estimation). Through a set of simple C functions, algorithms can propagate variables of any type, even C++ classes, to other algorithms.

An algorithm communication handle (**algo_comm_t** (p. 88)) is passed at initialisation time to the constructor of each plugin class **constructor** (p. 238). This handle contains a reference handle, **algo_comm_t::handle** (p. 88), and a number of function pointers, **algo_comm_t::insert_var** (p. 88) etc.. An algorithm communication variable is an object of type **comm_var_t** (p. 95).

For AC variables of numeric types, openMHA Plugins for conversion into parsable monitor variables, acmon, and storage into Matlab or text files, acsave, are available.

2.5.2 Function Documentation

2.5.2.1 mha_spec_t MHA_AC::get_var_spectrum (algo_comm_t *ac,* const std::string & *name*)

This function reads an AC variable and tries to convert it into a valid spectrum. The Spectrum variable is granted to be valid only for one call of the processing function.

Parameters

ac	AC handle
name	Name of the variable

Returns

Spectrum structure

2.5.2.2 mha_wave_t MHA_AC::get_var_waveform (algo_comm_t *ac,* const std::string & *name*)

This function reads an AC variable and tries to convert it into a valid waveform. The waveform variable is granted to be valid only for one call of the processing function.

Parameters

ac	AC handle
name	Name of the variable

Returns

waveform structure

2.5.2.3 int MHA_AC::get_var_int (algo_comm_t *ac,* const std::string & *name*)

Parameters

ac	AC handle
name	Name of the variable

Returns

Variable value

2.5.2.4 float MHA_AC::get_var_float (algo_comm_t *ac,* const std::string & *name*)

Parameters

ac	AC handle
name	Name of the variable

Returns

Variable value

2.5.2.5 std::vector< float > MHA_AC::get_var_vfloat (algo_comm_t *ac,* const std::string & *name*)

Parameters

ac	AC handle
name	Name of the variable

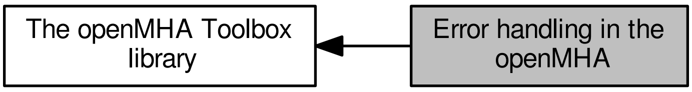
Returns

Variable value

2.6 Error handling in the openMHA

Errors are reported to the user via the **MHA_Error** (p. 132) exception.

Collaboration diagram for Error handling in the openMHA:



Classes

- class **MHA_Error**

    Error reporting exception class.

Macros

- #define **MHA_ErrorMsg**(x) **MHA_Error**(__FILE__,__LINE__,"%s",x)

    Throw an openMHA error with a text message.
- #define **MHA_assert**(x) if(!(x)) throw **MHA_Error**(__FILE__,__LINE__,"\"%s\" is false.",#x)

    Assertion macro, which throws an ***MHA_Error*** *(p. 132).*
- #define **MHA_assert_equal**(a, b) if(a != b) throw **MHA_Error**(__FILE__,__LINE__↩ ,"\"%s == %s\" is false (%s = %g, %s = %g).",#a,#b,#a,(double)(a),#b,(double)(b))

    Equality assertion macro, which throws an ***MHA_Error*** *(p. 132) with the values.*

Functions

- void **mha_debug** (const char ∗fmt,...)

    Print an info message (stderr on Linux, OutputDebugString in Windows).

2.6.1 Detailed Description

2.6.2 Macro Definition Documentation

2.6.2.1 #define MHA_ErrorMsg(x) MHA_Error(__FILE__,__LINE__,"%s",x)

Parameters

x	Text message.

2.6.2.2 #define MHA_assert(*x*) if(!(x)) throw MHA_Error(__FILE__,__LINE__,"\"%s\" is false.",#x)

Parameters

x	Boolean expression which should be true.

2.6.2.3 #define MHA_assert_equal(*a, b*) if(a != b) throw MHA_Error(__FILE__,__LINE__,"\"%s == %s\" is false (%s = %g, %s = %g).",#a,#b,#a,(double)(a),#b,(double)(b))

Parameters

a	Numeric expression which can be converted to double (for printing).
b	Numeric expression which should be equal to a

2.7 The openMHA configuration language

openMHA Plugins that should use the openMHA configuration language for their configuration have to be implemented in C++ and need to include **mha_parser.hh** (p. 292).

All required classes and functions for parser access are declared in the namespace **MHAParser** (p. 76). The plugin class should be derived from the class **MHAParser::parser_t** (p. 220) (or **MHAPlugin::plugin_t** (p. 238)), which symbolises a sub-parser node in the openMHA script hierarchy. Variables of many types can be registered to the sub-parser node by calling the member function **insert_item** (p. 222).

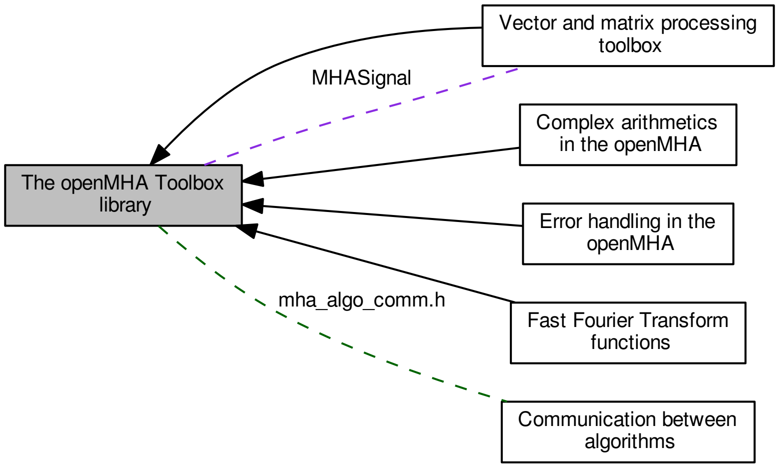
The openMHA Plugin template class **MHAPlugin::plugin_t** (p. 238) together with the Plugin macro **MHAPLUGIN_CALLBACKS** (p. 9) provide the callback mappings and correct inheritance. If your plugin is based on that template class, you simply have to use the insert_item command to give access to your variables, everything else is managed internally.

A complete list of all openMHA script items is given in the description of the **MHAParser** (p. 76) namespace.

2.8 The openMHA Toolbox library

The openMHA toolbox is a static C++ library which makes it more comfortable to develop openMHA plugins.

Collaboration diagram for The openMHA Toolbox library:



Modules

- **Error handling in the openMHA**

  Errors are reported to the user via the ***MHA_Error*** *(p. 132) exception.*
- **Vector and matrix processing toolbox**

  The vector and matrix processing toolbox consists of a number of classes defined in the namespace ***MHASignal*** *(p. 81), and many functions and operators for use with the structures* ***mha_wave_t*** *(p. 154) and* ***mha_spec_t*** *(p. 141).*
- **Complex arithmetics in the openMHA**
- **Fast Fourier Transform functions**

Files

- file **mha_algo_comm.h**

  Header file for Algorithm Communication.
- file **mha_filter.hh**

  Header file for IIR filter classes.
- file **mha_signal.hh**

  Header file for audio signal handling and processing classes.
- file **mha_tablelookup.hh**

  Header file for table lookup classes.

Namespaces

- **MHAOvlFilter**

    Namespace for overlapping FFT based filter bank classes and functions.
- **MHAFilter**

    Namespace for IIR and FIR filter classes.
- **MHAParser**

    Name space for the openMHA-Parser configuration language.
- **MHASignal**

    Namespace for audio signal handling and processing classes.
- **MHATableLookup**

    Namespace for table lookup classes.

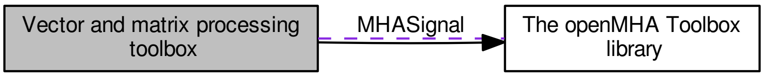
2.8.1 Detailed Description

It contains the openMHA script language classes.

2.9 Vector and matrix processing toolbox

The vector and matrix processing toolbox consists of a number of classes defined in the namespace **MHASignal** (p. 81), and many functions and operators for use with the structures **mha_wave_t** (p. 154) and **mha_spec_t** (p. 141).

Collaboration diagram for Vector and matrix processing toolbox:



Namespaces

- **MHASignal**

  Namespace for audio signal handling and processing classes.
- **MHAWindow**

  Collection of Window types.

Classes

- struct **mha_wave_t**

  Waveform signal structure.
- struct **mha_spec_t**

  Spectrum signal structure.
- struct **mha_audio_descriptor_t**

  *Description of an audio fragment (planned as a replacement of **mhaconfig_t** (p. 155)).*
- struct **mha_audio_t**

  *An audio fragment in the openMHA (planned as a replacement of **mha_wave_t** (p. 154) and **mha_spec_t** (p. 141)).*
- class **MHASignal::spectrum_t**

  *a signal processing class for spectral data (based on **mha_spec_t** (p. 141))*
- class **MHASignal::waveform_t**

  *signal processing class for waveform data (based on **mha_wave_t** (p. 154))*
- class **MHASignal::doublebuffer_t**

  Double-buffering class.
- class **MHASignal::hilbert_t**

  Hilbert transformation of a waveform segment.
- class **MHASignal::minphase_t**

  Minimal phase function.
- class **MHASignal::uint_vector_t**

Vector of unsigned values, used for size and index description of n-dimensional matrixes.

- class **MHASignal::matrix_t**

  n-dimensional matrix with real or complex floating point values.
- class **MHAParser::window_t**

  MHA configuration interface for a window function generator.
- class **MHASignal::delay_wave_t**

  Delayline containing wave fragments.
- class **MHASignal::async_rmslevel_t**

  Class for asynchronous level metering.

Typedefs

- typedef float **mha_real_t**

  openMHA type for real numbers

Functions

- **mha_wave_t range** (**mha_wave_t** s, unsigned int k0, unsigned int len)

  Return a time interval from a waveform chunk.
- **mha_spec_t channels** (**mha_spec_t** s, unsigned int ch_start, unsigned int nch)

  Return a channel interval from a spectrum.
- void **MHASignal::for_each** (**mha_wave_t** ∗s, **mha_real_t**(∗fun)(**mha_real_t**))

  Apply a function to each element of a ***mha_wave_t*** *(*p. *154**).*
- **mha_real_t MHASignal::lin2db** (**mha_real_t** x)

  Conversion from linear scale to dB (no SPL reference)
- **mha_real_t MHASignal::db2lin** (**mha_real_t** x)

  Conversion from dB scale to linear (no SPL reference)
- **mha_real_t MHASignal::pa2dbspl** (**mha_real_t** x)

  Conversion from linear Pascal scale to dB SPL.
- **mha_real_t MHASignal::pa22dbspl** (**mha_real_t** x, **mha_real_t** eps=1e-20f)

  Conversion from squared Pascal scale to dB SPL.
- **mha_real_t MHASignal::dbspl2pa** (**mha_real_t** x)

  Conversion from dB SPL to linear Pascal scale.
- **mha_real_t MHASignal::smp2sec** (**mha_real_t** n, **mha_real_t** srate)

  conversion from samples to seconds
- **mha_real_t MHASignal::sec2smp** (**mha_real_t** sec, **mha_real_t** srate)

  conversion from seconds to samples
- **mha_real_t MHASignal::bin2freq** (**mha_real_t** bin, unsigned fftlen, **mha_real_t** srate)

  conversion from fft bin index to frequency
- **mha_real_t MHASignal::freq2bin** (**mha_real_t** freq, unsigned fftlen, **mha_real_t** srate)

  conversion from frequency to fft bin index
- **mha_real_t MHASignal::smp2rad** (**mha_real_t** samples, unsigned bin, unsigned fftlen)

  conversion from delay in samples to phase shift

- **mha_real_t MHASignal::rad2smp** (**mha_real_t** phase_shift, unsigned bin, unsigned fftlen)

  conversion from phase shift to delay in samples
- template<class elem_type >
  std::vector< elem_type > **MHASignal::dupvec** (std::vector< elem_type > vec, unsigned n)

  Duplicate last vector element to match desired size.
- template<class elem_type >
  std::vector< elem_type > **MHASignal::dupvec_chk** (std::vector< elem_type > vec, unsigned n)

  Duplicate last vector element to match desired size, check for dimension.
- bool **equal_dim** (const **mha_wave_t** &a, const **mha_wave_t** &b)

  Test for equal dimension of waveform structures.
- bool **equal_dim** (const **mha_wave_t** &a, const **mhaconfig_t** &b)

  Test for match of waveform dimension with mhaconfig structure.
- bool **equal_dim** (const **mha_spec_t** &a, const **mha_spec_t** &b)

  Test for equal dimension of spectrum structures.
- bool **equal_dim** (const **mha_spec_t** &a, const **mhaconfig_t** &b)

  Test for match of spectrum dimension with mhaconfig structure.
- bool **equal_dim** (const **mha_wave_t** &a, const **mha_spec_t** &b)

  Test for equal dimension of waveform/spectrum structures.
- bool **equal_dim** (const **mha_spec_t** &a, const **mha_wave_t** &b)

  Test for equal dimension of waveform/spectrum structures.
- void **integrate** (**mha_wave_t** &s)

  Numeric integration of a signal vector (real values)
- void **integrate** (**mha_spec_t** &s)

  Numeric integration of a signal vector (complex values)
- unsigned int **size** (const **mha_wave_t** &s)

  Return size of a waveform structure.
- unsigned int **size** (const **mha_spec_t** &s)

  Return size of a spectrum structure.
- unsigned int **size** (const **mha_wave_t** *s)

  Return size of a waveform structure.
- unsigned int **size** (const **mha_spec_t** *s)

  Return size of a spectrum structure.
- void **clear** (**mha_wave_t** &s)

  Set all values of waveform to zero.
- void **clear** (**mha_wave_t** *s)

  Set all values of waveform to zero.
- void **clear** (**mha_spec_t** &s)

  Set all values of spectrum to zero.
- void **clear** (**mha_spec_t** *s)

  Set all values of spectrum to zero.
- void **assign** (**mha_wave_t** self, **mha_real_t** val)

  Set all values of waveform 'self' to 'val'.

- void **assign** (**mha_wave_t** self, const **mha_wave_t** &val)

  Set all values of waveform 'self' to 'val'.
- void **assign** (**mha_spec_t** self, const **mha_spec_t** &val)

  Set all values of spectrum 'self' to 'val'.
- void **timeshift** (**mha_wave_t** &self, int shift)

  Time shift of waveform chunk.
- **mha_real_t** & **value** (**mha_wave_t** ∗s, unsigned int fr, unsigned int ch)

  Access an element of a waveform structure.
- const **mha_real_t** & **value** (const **mha_wave_t** ∗s, unsigned int fr, unsigned int ch)

  Constant access to an element of a waveform structure.
- **mha_complex_t** & **value** (**mha_spec_t** ∗s, unsigned int fr, unsigned int ch)

  Access to an element of a spectrum.
- const **mha_complex_t** & **value** (const **mha_spec_t** ∗s, unsigned int fr, unsigned int ch)

  Constant access to an element of a spectrum.
- **mha_real_t** & **value** (**mha_wave_t** &s, unsigned int fr, unsigned int ch)

  Access to an element of a waveform structure.
- const **mha_real_t** & **value** (const **mha_wave_t** &s, unsigned int fr, unsigned int ch)

  Constant access to an element of a waveform structure.
- **mha_complex_t** & **value** (**mha_spec_t** &s, unsigned int fr, unsigned int ch)

  Access to an element of a spectrum.
- const **mha_complex_t** & **value** (const **mha_spec_t** &s, unsigned int fr, unsigned int ch)

  Constant access to an element of a spectrum.
- std::vector< float > **std_vector_float** (const **mha_wave_t** &)

  *Converts a **mha_wave_t** (p. 154) structure into a std::vector<float> (interleaved order).*
- std::vector< std::vector< float > > **std_vector_vector_float** (const **mha_wave_t** &)

  *Converts a **mha_wave_t** (p. 154) structure into a std::vector< std::vector<float> > (outer vector represents channels).*
- std::vector< std::vector< **mha_complex_t** > > **std_vector_vector_complex** (const **mha_spec_t** &)

  *Converts a **mha_spec_t** (p. 141) structure into a std::vector< std::vector<mha_complex_t> > (outer vector represents channels).*
- **mha_wave_t** & **operator+=** (**mha_wave_t** &, const **mha_real_t** &)

  Addition operator.
- **mha_wave_t** & **operator+=** (**mha_wave_t** &, const **mha_wave_t** &)

  Addition operator.
- **mha_wave_t** & **operator-=** (**mha_wave_t** &, const **mha_wave_t** &)

  Subtraction operator.
- **mha_spec_t** & **operator-=** (**mha_spec_t** &, const **mha_spec_t** &)

  Subtraction operator.
- **mha_wave_t** & **operator∗=** (**mha_wave_t** &, const **mha_real_t** &)

  Element-wise multiplication operator.
- **mha_wave_t** & **operator∗=** (**mha_wave_t** &, const **mha_wave_t** &)

  Element-wise multiplication operator.
- **mha_spec_t** & **operator∗=** (**mha_spec_t** &, const **mha_real_t** &)

  Element-wise multiplication operator.

- **mha_spec_t** & **operator*=** (**mha_spec_t** &, const **mha_wave_t** &)
  Element-wise multiplication operator.
- **mha_spec_t** & **operator*=** (**mha_spec_t** &, const **mha_spec_t** &)
  Element-wise multiplication operator.
- **mha_spec_t** & **operator/=** (**mha_spec_t** &, const **mha_spec_t** &)
  Element-wise division operator.
- **mha_wave_t** & **operator/=** (**mha_wave_t** &, const **mha_wave_t** &)
  Element-wise division operator.
- **mha_spec_t** & **operator+=** (**mha_spec_t** &, const **mha_spec_t** &)
  Addition operator.
- **mha_spec_t** & **operator+=** (**mha_spec_t** &, const **mha_real_t** &)
  Addition operator.
- **mha_wave_t** & **operator^=** (**mha_wave_t** &self, const **mha_real_t** &arg)
  Exponent operator.
- void **MHASignal::copy_channel** (**mha_spec_t** &self, const **mha_spec_t** &src, unsigned sch, unsigned dch)
  Copy one channel of a source signal.
- void **MHASignal::copy_channel** (**mha_wave_t** &self, const **mha_wave_t** &src, unsigned src_channel, unsigned dest_channel)
  Copy one channel of a source signal.
- **mha_real_t MHASignal::rmslevel** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen)
  Return RMS level of a spectrum channel.
- **mha_real_t MHASignal::colored_intensity** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen, **mha_real_t** sqfreq_response[ ])
  Colored spectrum intensity.
- **mha_real_t MHASignal::maxabs** (const **mha_spec_t** &s, unsigned int channel)
  Find maximal absolute value.
- **mha_real_t MHASignal::rmslevel** (const **mha_wave_t** &s, unsigned int channel)
  Return RMS level of a waveform channel.
- **mha_real_t MHASignal::maxabs** (const **mha_wave_t** &s, unsigned int channel)
  Find maximal absolute value.
- **mha_real_t MHASignal::maxabs** (const **mha_wave_t** &s)
  Find maximal absolute value.
- **mha_real_t MHASignal::max** (const **mha_wave_t** &s)
  Find maximal value.
- **mha_real_t MHASignal::min** (const **mha_wave_t** &s)
  Find minimal value.
- **mha_real_t MHASignal::sumsqr_channel** (const **mha_wave_t** &s, unsigned int channel)
  Calculate sum of squared values in one channel.
- **mha_real_t MHASignal::sumsqr_frame** (const **mha_wave_t** &s, unsigned int frame)
  Calculate sum over all channels of squared values.
- void **conjugate** (**mha_spec_t** &self)
  *Replace (!) the value of this **mha_spec_t** (p. 141) with its conjugate.*

2.9.1 Detailed Description

2.9.2 Typedef Documentation

2.9.2.1 typedef float mha_real_t

This type is expected to be allways the C-type 'float' (IEEE 754 single).

2.9.3 Function Documentation

2.9.3.1 mha_wave_t range (mha_wave_t *s*, unsigned int *k0*, unsigned int *len*)

A waveform chunk containing a time intervall of a larger waveform chunk is returned. The number of channels remains constant. The data of the output waveform structure points to the data of the input structure, i.e., write access to the output waveform chunk modifies the corresponding entries in the input chunk.

Parameters

s	Waveform structure
k0	Index of first value in output
len	Number of frames in output

Returns

Waveform structure representing the sub-interval.

2.9.3.2 mha_spec_t channels (mha_spec_t *s*, unsigned int *ch_start*, unsigned int *nch*)

Parameters

s	Input spectrum
ch_start	Index of first channel in output
nch	Number of channels in output

Returns

Spectrum structure representing the sub-interval.

2.9.3.3 void MHASignal::for_each (mha_wave_t ∗ *s*, mha_real_t(∗)(mha_real_t) *fun*) [inline]

Parameters

s	Pointer to a **mha_wave_t** (p. 154) structure
fun	Function to be applied (one argument)

2.9.3.4 mha_real_t MHASignal::lin2db (mha_real_t *x*) [inline]

Parameters

x	Linear input.

2.9.3.5 mha_real_t MHASignal::db2lin (mha_real_t *x*) [inline]

Parameters

x	dB input.

2.9.3.6 mha_real_t MHASignal::pa2dbspl (mha_real_t *x*) [inline]

Parameters

x	Linear input.

2.9.3.7 mha_real_t MHASignal::pa22dbspl (mha_real_t *x*, mha_real_t *eps* = 1e-20f) [inline]

Parameters

x	squared pascal input
eps	minimum squared-pascal value

2.9.3.8 mha_real_t MHASignal::dbspl2pa (mha_real_t *x*) [inline]

Parameters

x	Linear input.

2.9.3.9 mha_real_t MHASignal::smp2sec (mha_real_t *n*, mha_real_t *srate*) [inline]

Parameters

n	number of samples
srate	sampling rate / Hz

2.9.3.10 mha_real_t MHASignal::sec2smp (mha_real_t *sec*, mha_real_t *srate*) [inline]

Parameters

sec	time in seconds
srate	sampling rate / Hz

Returns

number of samples, generally has non-zero fractional part

2.9.3.11 mha_real_t MHASignal::bin2freq (mha_real_t *bin*, unsigned *fftlen*, mha_real_t *srate*) [inline]

Parameters

bin	index of fft bin, index 0 has dc
fftlen	FFT length
srate	sampling frequency / Hz

Returns

frequency of fft bin / Hz

2.9.3.12 mha_real_t MHASignal::freq2bin (mha_real_t *freq*, unsigned *fftlen*, mha_real_t *srate*) [inline]

Parameters

freq	frequency / Hz
fftlen	FFT length
srate	sampling frequency / Hz

Returns

0-based index of fft bin, generally has non-zero fractional part

2.9.3.13 mha_real_t MHASignal::smp2rad (mha_real_t *samples*, unsigned *bin*, unsigned *fftlen*) [inline]

Compute phase shift that needs to be applied to fft spectrum to achieve the desired delay.

Parameters

samples	delay in samples. Positive delay: shift current signal to future.
bin	index of fft bin, index 0 has dc (index 0 and nyqvist bin cannot be delayed)
fftlen	FFT length

Returns

The phase shift in radiant that needs to be applied to fft bin to achieve the desired delay. A positive delay requires a negative phase shift. If required phase shift is >pi or <-pi, then the desired delay cannot be applied in the fft domain with given parameters. Required phase shifts close to pi should not be used. If bin is 0 or nyqvist, returns 0 phase shift.

2.9.3.14 mha_real_t MHASignal::rad2smp (mha_real_t *phase_shift*, unsigned *bin*, unsigned *fftlen*) [inline]

Compute delay in samples that is achieved by a phase shift.

Parameters

phase_shift	phase shift in radiant
bin	index of fft bin, index 0 has dc (index 0 and nyqvist bin cannot be delayed)
fftlen	FFT length

Returns

The delay in samples achieved by applying the phase shift. A negative phase shift causes a positive delay: shifts current signal to future.

2.9.3.15 template<class elem_type > std::vector<elem_type> MHASignal::dupvec (std::vector< elem_type > *vec*, unsigned *n*)

Parameters

vec	Input vector.
n	Target number of elements.

Return values

Resized	vector.

2.9.3.16 template<class elem_type > std::vector<elem_type> MHASignal::dupvec_chk (std::vector< elem_type > *vec*, unsigned *n*)

The input dimension can be either 1 or the target length.

Parameters

vec	Input vector.
n	Target number of elements.

Return values

Resized	vector.

2.9.3.17 bool equal_dim (const **mha_wave_t** & *a*, const **mha_spec_t** & *b*) [inline]

Warning

Waveform structures **mha_wave_t** (p. 154) use interleaved data order, while spectrum structures **mha_spec_t** (p. 141) use non-interleaved.

2.9.3.18 bool equal_dim (const **mha_spec_t** & *a*, const **mha_wave_t** & *b*) [inline]

Warning

Waveform structures **mha_wave_t** (p. 154) use interleaved data order, while spectrum structures **mha_spec_t** (p. 141) use non-interleaved.

2.9.3.19 void integrate (**mha_wave_t** & *s*)

Parameters

s	Input signal vector

2.9.3.20 void integrate (**mha_spec_t** & *s*)

Parameters

s	Input signal vector

2.9.3.21 void assign (**mha_wave_t** *self*, **mha_real_t** *val*) [inline]

Parameters

self	Waveform to be modified.
val	Value to be assigned to all entries of waveform.

2.9.3.22 void assign (**mha_wave_t** *self*, const **mha_wave_t** & *val*)

Parameters

self	Waveform to be modified.
val	Source waveform structure.

2.9.3.23 void assign (**mha_spec_t** *self,* const **mha_spec_t** & *val*)

Parameters

self	Spectrum to be modified.
val	Source spectrum.

2.9.3.24 void timeshift (**mha_wave_t** & *self,* int *shift*)

Shifted areas are filled with zeros.

Parameters

self	Waveform chunk to be shifted
shift	Shift amount, positive values shift to later times

2.9.3.25 **mha_real_t**& value (**mha_wave_t** ∗ *s,* unsigned int *fr,* unsigned int *ch*) [inline]

Parameters

s	Waveform structure
fr	Frame number
ch	Channel number

Returns

Reference to element

2.9.3.26 const **mha_real_t**& value (const **mha_wave_t** ∗ *s,* unsigned int *fr,* unsigned int *ch*) [inline]

Parameters

s	Waveform structure
fr	Frame number
ch	Channel number

Returns

Reference to element

2.9.3.27 **mha_complex_t**& value (**mha_spec_t** ∗ *s,* unsigned int *fr,* unsigned int *ch*) [inline]

Parameters

s	Spectrum structure
fr	Bin number
ch	Channel number

Returns

Reference to element

2.9.3.28 const mha_complex_t& value (const mha_spec_t * s, unsigned int fr, unsigned int ch) [inline]

Parameters

s	Spectrum structure
fr	Bin number
ch	Channel number

Returns

Reference to element

2.9.3.29 mha_real_t& value (mha_wave_t & s, unsigned int fr, unsigned int ch) [inline]

Parameters

s	Waveform structure
fr	Frame number
ch	Channel number

Returns

Reference to element

2.9.3.30 const mha_real_t& value (const mha_wave_t & s, unsigned int fr, unsigned int ch) [inline]

Parameters

s	Waveform structure
fr	Frame number
ch	Channel number

Returns

Reference to element

2.9.3.31 mha_complex_t& value (mha_spec_t & *s*, unsigned int *fr*, unsigned int *ch*) [inline]

Parameters

s	Spectrum structure
fr	Bin number
ch	Channel number

Returns

Reference to element

2.9.3.32 const mha_complex_t& value (const mha_spec_t & *s*, unsigned int *fr*, unsigned int *ch*) [inline]

Parameters

s	Spectrum structure
fr	Bin number
ch	Channel number

Returns

Reference to element

2.9.3.33 std::vector<float> std_vector_float (const mha_wave_t &)

Warning

This function is not real-time safe. Do not use in signal processing thread.

2.9.3.34 std::vector<std::vector<float> > std_vector_vector_float (const mha_wave_t &)

Warning

This function is not real-time safe. Do not use in signal processing thread.

2.9.3.35 std::vector<std::vector<mha_complex_t> > std_vector_vector_complex (const mha_spec_t &)

Warning

This function is not real-time safe. Do not use in signal processing thread.

2.9.3.36 mha_wave_t& operator^= (mha_wave_t & *self,* const mha_real_t & *arg*)

Warning

This overwrites the xor operator!

2.9.3.37 void MHASignal::copy_channel (mha_spec_t & *self,* const mha_spec_t & *src,* unsigned *sch,* unsigned *dch*)

Parameters

self	Destination.
src	Source
sch	Source channel number
dch	Destination channel number

2.9.3.38 void MHASignal::copy_channel (mha_wave_t & *self,* const mha_wave_t & *src,* unsigned *src_channel,* unsigned *dest_channel*)

Parameters

self	Destination.
src	Source
src_channel	Source channel number
dest_channel	Destination channel number

2.9.3.39 mha_real_t MHASignal::rmslevel (const mha_spec_t & *s,* unsigned int *channel,* unsigned int *fftlen*)

Parameters

s	Input spectrum
channel	Channel number to be tested
fftlen	FFT length (to correctly count the level of the Nyquist bin)

Returns

RMS level in Pa

2.9.3.40 mha_real_t MHASignal::colored_intensity (const mha_spec_t & *s,* unsigned int *channel,* unsigned int *fftlen,* mha_real_t *sqfreq_response[ ]*)

computes the squared sum of the spectrum after filtering with the frequency response

Parameters

s	Input spectrum
channel	Channel number to be tested
fftlen	FFT length (to correctly count the level of the Nyquist bin)
sqfreq_response	A squared weighting factor for every fft bin.

Returns

sum of squares. Root of this is the colored level in Pa

2.9.3.41 mha_real_t MHASignal::maxabs (const mha_spec_t & *s*, unsigned int *channel*)

Parameters

s	Input signal
channel	Channel to be tested

Returns

maximum absolute value

2.9.3.42 mha_real_t MHASignal::rmslevel (const mha_wave_t & *s*, unsigned int *channel*)

Parameters

s	Input waveform signal
channel	Channel number to be tested

Returns

RMS level in Pa

2.9.3.43 mha_real_t MHASignal::maxabs (const mha_wave_t & *s*, unsigned int *channel*)

Parameters

s	Input signal
channel	Channel to be tested

Returns

maximum absolute value

2.9.3.44 mha_real_t MHASignal::maxabs (const mha_wave_t & *s*)

Parameters

s	Input signal

Returns

maximum absolute value

2.9.3.45 mha_real_t MHASignal::max (const mha_wave_t & *s*)

Parameters

s	Input signal

Returns

maximum absolute value

2.9.3.46 mha_real_t MHASignal::min (const mha_wave_t & *s*)

Parameters

s	Input signal

Returns

maximum absolute value

2.9.3.47 mha_real_t MHASignal::sumsqr_channel (const mha_wave_t & *s*, unsigned int *channel*)

Parameters

s	Input signal
channel	Channel

Returns

$\sum x^2$

2.9.3.48 mha_real_t MHASignal::sumsqr_frame (const mha_wave_t & *s*, unsigned int *frame*)

Parameters

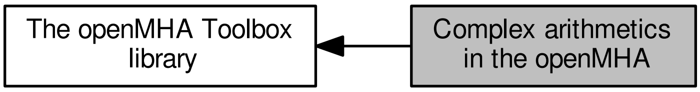
s	Input signal
frame	Frame number

Returns

$\sum x^2$

2.10 Complex arithmetics in the openMHA

Collaboration diagram for Complex arithmetics in the openMHA:



Classes

- struct **mha_complex_t**

  Type for complex floating point values.

Functions

- **mha_complex_t** & **set** (**mha_complex_t** &self, **mha_real_t** real, **mha_real_t** imag=0)

  Assign real and imaginary parts to a ***mha_complex_t*** *(p. 123) variable.*
- **mha_complex_t** **mha_complex** (**mha_real_t** real, **mha_real_t** imag=0)

  Create a new ***mha_complex_t*** *(p. 123) with specified real and imaginary parts.*
- **mha_complex_t** & **set** (**mha_complex_t** &self, const std::complex< **mha_real_t** > &**stdcomplex**)

  Assign a ***mha_complex_t*** *(p. 123) variable from a std::complex.*
- std::complex< **mha_real_t** > **stdcomplex** (const **mha_complex_t** &self)

  Create a std::complex from ***mha_complex_t*** *(p. 123).*
- **mha_complex_t** & **expi** (**mha_complex_t** &self, **mha_real_t** **angle**)

  replaces the value of the given ***mha_complex_t*** *(p. 123) with exp(i∗b).*
- double **angle** (const **mha_complex_t** &self)

  Computes the angle of a complex number in the complex plane.
- **mha_complex_t** & **operator+=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Addition of two complex numbers, overwriting the first.
- **mha_complex_t** **operator+** (const **mha_complex_t** &self, const **mha_complex_t** &other)

  Addition of two complex numbers, result is a temporary object.
- **mha_complex_t** & **operator+=** (**mha_complex_t** &self, **mha_real_t** other_real)

  Addition of a complex and a real number, overwriting the complex.
- **mha_complex_t** **operator+** (const **mha_complex_t** &self, **mha_real_t** other_real)

  Addition of a complex and a real number, result is a temporary object.
- **mha_complex_t** & **operator-=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Subtraction of two complex numbers, overwriting the first.

- **mha_complex_t operator-** (const **mha_complex_t** &self, const **mha_complex_↩ t** &other)

    Subtraction of two complex numbers, result is a temporary object.
- **mha_complex_t** & **operator-=** (**mha_complex_t** &self, **mha_real_t** other_real)

    Subtraction of a complex and a real number, overwriting the complex.
- **mha_complex_t operator-** (const **mha_complex_t** &self, **mha_real_t** other_real)

    Subtraction of a complex and a real number, result is a temporary object.
- **mha_complex_t** & **operator*=** (**mha_complex_t** &self, const **mha_complex_t** &other)

    Multiplication of two complex numbers, overwriting the first.
- **mha_complex_t operator*** (const **mha_complex_t** &self, const **mha_complex_↩ t** &other)

    Multiplication of two complex numbers, result is a temporary object.
- **mha_complex_t** & **operator*=** (**mha_complex_t** &self, **mha_real_t** other_real)

    Multiplication of a complex and a real number, overwriting the complex.
- **mha_complex_t** & **expi** (**mha_complex_t** &self, **mha_real_t angle**, **mha_real_t** factor)

    *replaces (!) the value of the given **mha_complex_t** (p. 123) with a * exp(i*b)*
- **mha_complex_t operator*** (const **mha_complex_t** &self, **mha_real_t** other_real)

    Multiplication of a complex and a real number, result is a temporary object.
- **mha_real_t abs2** (const **mha_complex_t** &self)

    Compute the square of the absolute value of a complex value.
- **mha_real_t abs** (const **mha_complex_t** &self)

    Compute the absolute value of a complex value.
- **mha_complex_t** & **operator/=** (**mha_complex_t** &self, **mha_real_t** other_real)

    Division of a complex and a real number, overwriting the complex.
- **mha_complex_t operator/** (const **mha_complex_t** &self, **mha_real_t** other_real)

    Division of a complex and a real number, result is a temporary object.
- **mha_complex_t** & **safe_div** (**mha_complex_t** &self, const **mha_complex_t** &other, **mha_real_t** eps, **mha_real_t** eps2)

    Safe division of two complex numbers, overwriting the first.
- **mha_complex_t** & **operator/=** (**mha_complex_t** &self, const **mha_complex_t** &other)

    Division of two complex numbers, overwriting the first.
- **mha_complex_t operator/** (const **mha_complex_t** &self, const **mha_complex_↩ t** &other)

    Division of two complex numbers, result is a temporary object.
- **mha_complex_t operator-** (const **mha_complex_t** &self)

    Unary minus on a complex results in a negative temporary object.
- bool **operator==** (const **mha_complex_t** &x, const **mha_complex_t** &y)

    Compare two complex numbers for equality.
- bool **operator!=** (const **mha_complex_t** &x, const **mha_complex_t** &y)

    Compare two complex numbers for inequality.
- void **conjugate** (**mha_complex_t** &self)

    *Replace (!) the value of this **mha_complex_t** (p. 123) with its conjugate.*
- **mha_complex_t _conjugate** (const **mha_complex_t** &self)

    Compute the cojugate of this complex value.
- void **reciprocal** (**mha_complex_t** &self)

Replace the value of this complex with its reciprocal.

- **mha_complex_t _reciprocal** (const **mha_complex_t** &self)

  compute the reciprocal of this complex value.
- void **normalize** (**mha_complex_t** &self)

  Divide a complex by its absolute value, thereby normalizing it (projecting onto the unit circle).
- void **normalize** (**mha_complex_t** &self, **mha_real_t** margin)

  Divide a complex by its absolute value, thereby normalizing it (projecting onto the unit circle), with a safety margin.
- bool **almost** (const **mha_complex_t** &self, const **mha_complex_t** &other, **mha_real_t** times_epsilon=1e2)

  Compare two complex numbers for equality except for a small relative error.
- bool **operator**< (const **mha_complex_t** &x, const **mha_complex_t** &y)

  Compares the absolute values of two complex numbers.

2.10.1 Detailed Description

2.10.2 Function Documentation

2.10.2.1 mha_complex_t& set (mha_complex_t & *self*, mha_real_t *real*, mha_real_t *imag* = 0) [inline]

Parameters

self	The **mha_complex_t** (p. 123) variable whose value is about to change.
real	The new real part.
imag	The new imaginary part.

Returns

A reference to the changed variable.

2.10.2.2 mha_complex_t mha_complex (mha_real_t *real*, mha_real_t *imag* = 0) [inline]

Parameters

real	The real part.
imag	The imaginary part.

Returns

The new value.

2.10.2.3 mha_complex_t& set (mha_complex_t & *self*, const std::complex< mha_real_t > & *stdcomplex*) [inline]

Parameters

self	The **mha_complex_t** (p. 123) variable whose value is about to change.
stdcomplex	The new complex value.

Returns

A reference to the changed variable.

2.10.2.4 mha_complex_t& expi (mha_complex_t & *self*, mha_real_t *angle*) [inline]

Parameters

self	The **mha_complex_t** (p. 123) variable whose value is about to change.
angle	The angle in the complex plane [rad].

Returns

A reference to the changed variable.

2.10.2.5 double angle (const mha_complex_t & *self*) [inline]

Parameters

self	The complex number whose angle is needed.

Returns

The angle of a complex number in the complex plane.

2.10.2.6 mha_complex_t& expi (mha_complex_t & *self*, mha_real_t *angle*, mha_real_t *factor*) [inline]

Parameters

self	The **mha_complex_t** (p. 123) variable whose value is about to change.
angle	The imaginary exponent.
factor	The absolute value of the result.

Returns

A reference to the changed variable.

2.10.2.7 mha_real_t abs2 (const mha_complex_t & *self*) [inline]

Returns

The square of the absolute value of self.

2.10.2.8 mha_real_t abs (const mha_complex_t & *self*) [inline]

Returns

The absolute value of self.

2.10.2.9 mha_complex_t& safe_div (mha_complex_t & *self,* const mha_complex_t & *other,* mha_real_t *eps,* mha_real_t *eps2*) [inline]

If abs(divisor) < eps, then divisor is replaced by eps. eps2 = eps∗eps.

2.10.2.10 mha_complex_t _conjugate (const mha_complex_t & *self*) [inline]

Returns

A temporary object holding the conjugate value.

2.10.2.11 mha_complex_t _reciprocal (const mha_complex_t & *self*) [inline]

Returns

A temporary object holding the reciprocal value.

2.10.2.12 bool almost (const mha_complex_t & *self,* const mha_complex_t & *other,* mha_real_t *times_epsilon* = `1e2`) [inline]

Parameters

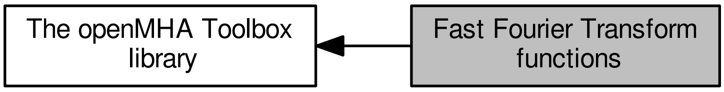
self	The first complex number.
other	The second complex number.
times_epsilon	Permitted relative error is this number multiplied with the machine accuracy for this Floating point format (std::numeric_limits<mha_real_t>::epsilon)

Returns

true if the relative difference is below times_epsilon ∗ std::numeric_limits<mha_real_t>↩
::epsilon

2.11 Fast Fourier Transform functions

Collaboration diagram for Fast Fourier Transform functions:



Typedefs

- typedef void ∗ **mha_fft_t**
  Handle for an FFT object.

Functions

- **mha_fft_t mha_fft_new** (unsigned int n)
  Create a new FFT handle.
- void **mha_fft_free** (**mha_fft_t** h)
  Destroy an FFT handle.
- void **mha_fft_wave2spec** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_t** ∗out)
  Tranform waveform segment into spectrum.
- void **mha_fft_wave2spec** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_t** ∗out, bool swaps)
  Tranform waveform segment into spectrum.
- void **mha_fft_spec2wave** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_t** ∗out)
  Tranform spectrum into waveform segment.
- void **mha_fft_spec2wave** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_t** ∗out, unsigned int offset)
  Tranform spectrum into waveform segment.
- void **mha_fft_forward** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)
  Complex to complex FFT (forward).
- void **mha_fft_backward** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)
  Complex to complex FFT (backward).
- void **mha_fft_forward_scale** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)
  Complex to complex FFT (forward).
- void **mha_fft_backward_scale** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)
  Complex to complex FFT (backward).
- void **mha_fft_wave2spec_scale** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_t** ∗out)
  Tranform waveform segment into spectrum.
- void **mha_fft_spec2wave_scale** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_t** ∗out)
  Tranform spectrum into waveform segment.

2.11.1 Detailed Description

2.11.2 Typedef Documentation

2.11.2.1 typedef void* mha_fft_t

This FFT object is used by the functions mha_fft_wave2spec and mha_fft_spec2wave. The F↩
FT back-end is the FFTW library. The back-end is completely hidden, including external header files or linking external libraries is not required.

2.11.3 Function Documentation

2.11.3.1 mha_fft_t mha_fft_new (unsigned int *n*)

Parameters

n	FFT length.

Create a new FFT handle.

Parameters

n	FFT length

Return values

FFT	object

2.11.3.2 void mha_fft_free (mha_fft_t *h*)

Parameters

h	Handle to be destroyed.

Destroy an FFT handle.

Parameters

h	FFT object to be removed

2.11.3.3 void mha_fft_wave2spec (mha_fft_t *h*, const mha_wave_t * *in*, mha_spec_t * *out*)

Parameters

h	FFT handle.
in	Input waveform segment.
out	Output spectrum.

Tranform waveform segment into spectrum.

Parameters

h	FFT object handle
in	pointer to input waveform signal
out	pointer to output spectrum signal (has to be allocated)

2.11.3.4 void mha_fft_wave2spec (mha_fft_t *h,* const mha_wave_t ∗ *in,* mha_spec_t ∗ *out,* bool *swaps*)

Like normal wave2spec, but swaps wave buffer halves before transforming if the swaps parameter is true.

Warning: These openMHA FFTs adopt a nonstandard scaling scheme in which the forward transform scales by 1/N and the backward does not scale. We would recommend using the '_scale' methods instead.

Parameters

h	FFT handle.
in	Input waveform segment.
out	Output spectrum.
swaps	Function swaps the first and second half of the waveform buffer before the FFT transform when this parameter is set to true.

2.11.3.5 void mha_fft_spec2wave (mha_fft_t *h,* const mha_spec_t ∗ *in,* mha_wave_t ∗ *out*)

Warning: These openMHA FFTs adopt a nonstandard scaling scheme in which the forward transform scales by 1/N and the backward does not scale. We would recommend using the '_scale' methods instead.

Parameters

h	FFT handle.
in	Input spectrum.
out	Output waveform segment.

Tranform spectrum into waveform segment.

Parameters

h	FFT object handle
in	pointer to input spectrum
out	pointer to output waveform signal (has to be allocated)

2.11.3.6 void mha_fft_spec2wave (mha_fft_t *h*, const mha_spec_t ∗ *in*, mha_wave_t ∗ *out*, unsigned int *offset*)

out may have fewer number of frames than needed for a complete iFFT. Only as many frames are written into out as fit, starting with offset offset of the complete iFFT.

Warning: These openMHA FFTs adopt a nonstandard scaling scheme in which the forward transform scales by 1/N and the backward does not scale. We would recommend using the '_scale' methods instead.

Parameters

h	FFT handle.
in	Input spectrum.
out	Output waveform segment.
offset	Offset into iFFT wave buffer

Tranform spectrum into waveform segment.

Only part of the iFFT is tranferred into the out buffer.

Out may have fewer number of freames than needed for a complete iFFT. Only as many frames are written into out as fit, starting with offset offset of the complete iFFT.

Parameters

h	FFT object handle
in	pointer to input spectrum
out	pointer to output waveform signal (has to be allocated)
offset	Offset into complete iFFT buffer.

2.11.3.7 void mha_fft_forward (mha_fft_t *h*, mha_spec_t ∗ *sIn*, mha_spec_t ∗ *sOut*)

sIn and sOut need to have nfft bins (please note that **mha_spec_t** (p. 141) typically has nfft/2+1 bins for half-complex representation).

Warning: These openMHA FFTs adopt a nonstandard scaling scheme in which the forward transform scales by 1/N and the backward does not scale. We would recommend using the '_scale' methods instead.

Parameters

h	FFT handle.
sIn	Input spectrum.
sOut	Output spectrum.

2.11.3.8 void mha_fft_backward (**mha_fft_t** *h,* **mha_spec_t** ∗ *sIn,* **mha_spec_t** ∗ *sOut*)

sIn and sOut need to have nfft bins (please note that **mha_spec_t** (p. 141) typically has nfft/2+1 bins for half-complex representation).

Warning: These openMHA FFTs adopt a nonstandard scaling scheme in which the forward transform scales by 1/N and the backward does not scale. We would recommend using the '_scale' methods instead.

Parameters

h	FFT handle.
sIn	Input spectrum.
sOut	Output spectrum.

2.11.3.9 void mha_fft_forward_scale (**mha_fft_t** *h,* **mha_spec_t** ∗ *sIn,* **mha_spec_t** ∗ *sOut*)

sIn and sOut need to have nfft bins (please note that **mha_spec_t** (p. 141) typically has nfft/2+1 bins for half-complex representation).

The _scale methods use standard DFT scaling: There is no scaling in the forward transformation, and 1/N scaling for the backward.

Parameters

h	FFT handle.
sIn	Input spectrum.
sOut	Output spectrum.

2.11.3.10 void mha_fft_backward_scale (**mha_fft_t** *h,* **mha_spec_t** ∗ *sIn,* **mha_spec_t** ∗ *sOut*)

sIn and sOut need to have nfft bins (please note that **mha_spec_t** (p. 141) typically has nfft/2+1 bins for half-complex representation).

The _scale methods use standard DFT scaling: There is no scaling in the forward transformation, and 1/N scaling for the backward.

Parameters

h	FFT handle.
sIn	Input spectrum.
sOut	Output spectrum.

2.11.3.11 void mha_fft_wave2spec_scale (**mha_fft_t** *h*, const **mha_wave_t** * *in*, **mha_spec_t** * *out*)

The _scale methods use standard DFT scaling: There is no scaling in the forward transformation, and 1/N scaling for the backward.

Parameters

h	FFT handle.
in	Input waveform segment.
out	Output spectrum.

2.11.3.12 void mha_fft_spec2wave_scale (**mha_fft_t** *h*, const **mha_spec_t** * *in*, **mha_wave_t** * *out*)

The _scale methods use standard DFT scaling: There is no scaling in the forward transformation, and 1/N scaling for the backward.

Parameters

h	FFT handle.
in	Input spectrum.
out	Output waveform segment.

3 Namespace Documentation

3.1 AuditoryProfile Namespace Reference

Namespace for classes and functions around the auditory profile (e.g., audiogram handling)

Classes

- class **fmap_t**

  A class to store frequency dependent data (e.g., HTL and UCL).
- class **parser_t**

  Class to make the auditory profile accessible through the parser interface.
- class **profile_t**

  The Auditory Profile class.

3.1.1 Detailed Description

The auditory profile as defined by HearCom or BMBF Modellbasierte Hoergeraete is stored in the class **AuditoryProfile::profile_t** (p. 94). Until a complete definition is available, only the currently needed elements are implemented.

3.2 DynComp Namespace Reference

dynamic compression related classes and functions

Classes

- class **dc_afterburn_rt_t**

  Real-time class for after burn effect.
- class **dc_afterburn_t**

  Afterburn class, to be defined as a member of compressors.
- class **dc_afterburn_vars_t**

  *Variables for **dc_afterburn_t*** *(p. 97)* *class.*
- class **gaintable_t**

  Gain table class.

Functions

- **mha_real_t interp1** (const std::vector< **mha_real_t** > &vX, const std::vector< **mha_real_t** > &vY, **mha_real_t** X)

    One-dimensional linear interpolation.
- **mha_real_t interp2** (const std::vector< **mha_real_t** > &vX, const std::vector< **mha_real_t** > &vY, const std::vector< std::vector< **mha_real_t** > > &mZ, **mha_real_t** X, **mha_real_t** Y)

    Linear interpolation in a two-dimensional field.

3.2.1 Function Documentation

3.2.1.1 mha_real_t DynComp::interp1 (const std::vector< mha_real_t > & *vX*, const std::vector< mha_real_t > & *vY*, mha_real_t *X*)

Parameters

vX	Vector with input samples.
vY	Vector with values at input samples.
X	Input value to be interpolated.

Return values

Interpolated	value Y(X) at position X.

3.2.1.2 mha_real_t DynComp::interp2 (const std::vector< mha_real_t > & *vX*, const std::vector< mha_real_t > & *vY*, const std::vector< std::vector< mha_real_t > > & *mZ*, mha_real_t *X*, mha_real_t *Y*)

Parameters

vX	Vector with input samples, first dimension.
vY	Vector with input samples, second dimension.
mZ	Field with values at input samples.
X	First dimension of input value to be interpolated.
Y	Second dimension of input value to be interpolated.

Return values

Interpolated	value Z(X,Y) at position X,Y.

3.3 MHA_AC Namespace Reference

Functions and classes for Algorithm Communication (AC) support.

Classes

- class **ac2matrix_t**

  Copy AC variable to a matrix.
- class **acspace2matrix_t**

  Copy all or a subset of all numeric AC variables into an array of matrixes.
- class **double_t**

  Insert a double precision floating point variable into the AC space.
- class **float_t**

  Insert a float point variable into the AC space.
- class **int_t**

  Insert a integer variable into the AC space.
- class **spectrum_t**

  *Insert a **MHASignal::spectrum_t** (p. 261) class into the AC space.*
- class **waveform_t**

  *Insert a **MHASignal::waveform_t** (p. 268) class into the AC space.*

Functions

- **mha_spec_t get_var_spectrum** (**algo_comm_t** ac, const std::string &name)

  Convert an AC variable into a spectrum.
- **mha_wave_t get_var_waveform** (**algo_comm_t** ac, const std::string &name)

  Convert an AC variable into a waveform.
- int **get_var_int** (**algo_comm_t** ac, const std::string &name)

  Return value of an integer scalar AC variable.
- float **get_var_float** (**algo_comm_t** ac, const std::string &name)

  Return value of an floating point scalar AC variable.
- std::vector< float > **get_var_vfloat** (**algo_comm_t** ac, const std::string &name)

  Return value of an floating point vector AC variable as standard vector of floats.

3.3.1 Detailed Description

3.4 MHA_TCP Namespace Reference

A Namespace for TCP helper classes.

Classes

- class **Async_Notify**

  Portable Multiplexable cross-thread notification.
- class **Client**

  A portable class for a tcp client connections.
- class **Connection**

  ***Connection** (p. 144) handles Communication between client and server, is used on both sides.*
- class **Event_Watcher**

  OS-independent event watcher, uses select on Unix and WaitForMultipleObjects on Windows.
- class **Sockread_Event**

  Watch socket for incoming data.
- class **Thread**

  A very simple class for portable threads.
- class **Timeout_Watcher**

  OS-independent event watcher with internal fixed-end-time timeout.
- class **Wakeup_Event**

  A base class for asynchronous wakeup events.

Functions

- std::string **STRERROR** (int err)

  Portable conversion from error number to error string.
- std::string **HSTRERROR** (int err)

  Portable conversion from hostname error number to error string.
- int **N_ERRNO** ()

  Portable access to last network error number.
- int **H_ERRNO** ()

  Portable access to last hostname error number.
- int **G_ERRNO** ()

  Portable access to last non-network error number.
- double **dtime** ()

  Time access function for system's high resolution time, retrieve current time as double.
- double **dtime** (const struct timeval &tv)

  Time access function for unix' high resolution time, converts struct timeval to double.
- struct timeval **stime** (double d)

  Time access function for unix' high resolution time, converts time from double to struct timeval.

3.5 MHAEvents Namespace Reference

Collection of event handling classes.

Classes

- class **emitter_t**
  Class for emitting openMHA events.
- class **patchbay_t**
  Patchbay which connects any event emitter with any member function of the parameter class.

3.6 MHAFilter Namespace Reference

Namespace for IIR and FIR filter classes.

Classes

- class **adapt_filter_t**
  Adaptive filter.
- class **blockprocessing_polyphase_resampling_t**
  A class that does polyphase resampling and takes into account block processing.
- class **complex_bandpass_t**
  Complex bandpass filter.
- class **diff_t**
  Differentiator class (non-normalized)
- class **fftfilter_t**
  FFT based FIR filter implementation.
- class **fftfilterbank_t**
  FFT based FIR filterbank implementation.
- class **filter_t**
  Generic IIR filter class.
- class **gamma_flt_t**
  Class for gammatone filter.
- class **iir_filter_t**
  IIR filter class wrapper for integration into parser structure.
- class **iir_ord1_real_t**
  First order recursive filter.
- class **o1_ar_filter_t**
  First order attack-release lowpass filter.
- class **o1flt_lowpass_t**
  First order low pass filter.
- class **o1flt_maxtrack_t**
  First order maximum tracker.
- class **o1flt_mintrack_t**
  First order minimum tracker.
- class **partitioned_convolution_t**
  A filter class for partitioned convolution.

- class **polyphase_resampling_t**

  A class that does polyphase resampling.
- class **resampling_filter_t**

  Hann shaped low pass filter for resampling.
- class **smoothspec_t**

  Smooth spectral gains, create a windowed impulse response.
- struct **transfer_function_t**

  a structure containing a source channel number, a target channel number, and an impulse response.
- struct **transfer_matrix_t**

  A sparse matrix of transfer function partitionss.

Functions

- void **o1_lp_coeffs** (const **mha_real_t** tau, const **mha_real_t** fs, **mha_real_t** &c1, **mha↩ _real_t** &c2)

  Set first order filter coefficients from time constant and sampling rate.
- void **butter_stop_ord1** (double ∗A, double ∗B, double f1, double f2, double fs)

  Setup a first order butterworth band stop filter.
- **MHASignal::waveform_t** ∗ **spec2fir** (const **mha_spec_t** ∗spec, const unsigned int fftlen, const **MHAWindow::base_t** &window, const bool minphase)

  Create a windowed impulse response/FIR filter coefficients from a spectrum.
- unsigned **gcd** (unsigned a, unsigned b)

  greatest common divisor
- double **sinc** (double x)

  sin(x)/x function, coping with x=0.
- std::pair< unsigned, unsigned > **resampling_factors** (float source_sampling_rate, float target_sampling_rate, float factor=1.0f)

  Computes rational resampling factor from two sampling rates.

3.6.1 Function Documentation

3.6.1.1 void MHAFilter::o1_lp_coeffs (const mha_real_t *tau,* const mha_real_t *fs,* mha_real_t & *c1,* mha_real_t & *c2*)

Parameters

tau	Time constant
fs	Sampling rate

Return values

c1	Recursive filter coefficient
c2	Non-recursive filter coefficient

3.6.1.2 void MHAFilter::butter_stop_ord1 (double * *A,* double * *B,* double *f1,* double *f2,* double *fs*)

This function calculates the filter coefficients of a first order butterworth band stop filter.

Return values

A	recursive filter coefficients
B	non recursive filter coefficients

Parameters

f1	lower frequency
f2	upper frequency
fs	sample frequency

3.6.1.3 MHASignal::waveform_t * MHAFilter::spec2fir (const mha_spec_t * *spec,* const unsigned int *fftlen,* const MHAWindow::base_t & *window,* const bool *minphase*)

Parameters

spec	Input spectrum
fftlen	FFT length of spectrum
window	Window shape (with length, e.g. initialized with MHAWindow::hanning(54)).
minphase	Flag, true if original phase should be discarded and replaced by a minimal phase function.

3.6.1.4 double MHAFilter::sinc (double *x*)

This is the historical sinc function, not the normalized sinc function.

3.6.1.5 std::pair< unsigned, unsigned > MHAFilter::resampling_factors (float *source_sampling_rate,* float *target_sampling_rate,* float *factor* = `1.0f`)

The function will fail if either sampling_rate * factor is not an integer

Parameters

source_sampling_rate	The original sampling rate
target_sampling_rate	The desired sampling rate
factor	A helper factor to use for non-integer sampling rates

Returns

a pair that contains first the upsampling factor and second the downsampling factor required for the specified resampling.

Exceptions

MHA_Error *(*p. *132)*	if no rational resampling factor can be found.

3.7 MHAIOJack Namespace Reference

JACK IO.

Classes

- class **io_jack_t**

  Main class for JACK IO.

3.8 MHAJack Namespace Reference

Classes and functions for openMHA and JACK interaction.

Classes

- class **client_avg_t**

  Generic JACK client for averaging a system response across time.
- class **client_noncont_t**

  Generic client for synchronous playback and recording of waveform fragments.
- class **client_t**

  Generic asynchronous JACK client.
- class **port_t**

  Class for one channel/port.

Functions

- void **io** (**mha_wave_t** ∗s_out, **mha_wave_t** ∗s_in, const std::string &name, const std::vector< std::string > &p_out, const std::vector< std::string > &p_in, float ∗srate=NULL, unsigned int ∗fragsize=NULL, bool use_jack_transport=false)

    Functional form of generic client for synchronous playback and recording of waveform fragments.
- std::vector< unsigned int > **get_port_capture_latency** (const std::vector< std::string > &ports)

    Return the JACK port latency of ports.
- std::vector< int > **get_port_capture_latency_int** (const std::vector< std::string > &ports)

    Return the JACK port latency of ports.
- std::vector< unsigned int > **get_port_playback_latency** (const std::vector< std::string > &ports)

    Return the JACK port latency of ports.

3.8.1 Function Documentation

3.8.1.1 std::vector< unsigned int > MHAJack::get_port_capture_latency (const std::vector< std::string > & *ports*)

Parameters

ports	Ports to be tested

Returns

Latency vector (one entry for each port)

3.8.1.2 std::vector< int > MHAJack::get_port_capture_latency_int (const std::vector< std::string > & *ports*)

Parameters

ports	Ports to be tested

Returns

Latency vector (one entry for each port)

3.8.1.3 std::vector< unsigned int > MHAJack::get_port_playback_latency (const std::vector< std::string > & *ports*)

Parameters

ports	Ports to be tested

Returns

Latency vector (one entry for each port)

3.9 MHAMultiSrc Namespace Reference

Collection of classes for selecting audio chunks from multiple sources.

Classes

- class **base_t**

  Base class for source selection.

3.9.1 Detailed Description

3.10 MHAOvlFilter Namespace Reference

Namespace for overlapping FFT based filter bank classes and functions.

Namespaces

- **FreqScaleFun**

  Transform functions from linear scale in Hz to new frequency scales.
- **ShapeFun**

  Shape functions for overlapping filters.

Classes

- class **fftfb_t**

  FFT based overlapping filter bank.
- class **fftfb_vars_t**

  Set of configuration variables for FFT-based overlapping filters.
- class **fspacing_t**

  Class for frequency spacing, used by filterbank shape generator class.
- class **overlap_save_filterbank_t**

  A time-domain minimal phase filter bank with frequency shapes from ***MHAOvlFilter::fftfb_t*** *(*p. *196).*

3.11 MHAOvlFilter::FreqScaleFun Namespace Reference

Transform functions from linear scale in Hz to new frequency scales.

Functions

- **mha_real_t hz2hz** (**mha_real_t** x)
  Dummy scale transformation Hz to Hz.
- **mha_real_t hz2bark** (**mha_real_t** x)
  Transformation to bark scale.
- **mha_real_t hz2log** (**mha_real_t** x)
  Third octave frequency scale.

3.11.1 Function Documentation

3.11.1.1 mha_real_t MHAOvlFilter::FreqScaleFun::hz2hz (mha_real_t *x*)

This function implements a dummy scale transformation (linear frequency scale).

Parameters

x	Input frequency in Hz

Returns

Frequency in Hz

3.11.1.2 mha_real_t MHAOvlFilter::FreqScaleFun::hz2bark (mha_real_t *x*)

This function implements a critical band rate (bark) scale.

Parameters

x	Input frequency in Hz

Returns

Critical band rate in Bark

3.11.1.3 mha_real_t MHAOvlFilter::FreqScaleFun::hz2log (mha_real_t *x*)

This function implements a third octave scale. Frequencies below 16 Hz are mapped to 16 Hz.

Parameters

x	Frequency in Hz

Returns

Third octaves relative to 1000 Hz

3.12 MHAOvlFilter::ShapeFun Namespace Reference

Shape functions for overlapping filters.

Functions

- **mha_real_t rect** (**mha_real_t** x)
  Filter shape function for rectangular filters.
- **mha_real_t linear** (**mha_real_t** x)
  Filter shape function for sawtooth filters.
- **mha_real_t hann** (**mha_real_t** x)
  Filter shape function for hanning shaped filters.

3.12.1 Function Documentation

3.12.1.1 mha_real_t MHAOvlFilter::ShapeFun::rect (mha_real_t *x*)

This function creates rectangular filter shapes. The edge is exactly half way between two center frequencies (on a given scale).

Parameters

x	Input value in the range [-1,1].

Returns

Weigth function in the range [0,1]

3.12.1.2 mha_real_t MHAOvlFilter::ShapeFun::linear (mha_real_t *x*)

This function creates sawtooth filter shapes. They rise linearly form 0 to 1 in the interval from the lower neighbor center frequency to the band center frequency and from 1 to 0 in the interval from the band center frequency to the upper neighbour band center frequency. Linear means linear on a given frequency scale.

Parameters

x	Input value in the range [-1,1].

Returns

Weigth function in the range [0,1]

3.12.1.3 mha_real_t MHAOvlFilter::ShapeFun::hann (mha_real_t *x*)

This function creates hanning window shaped filters.

Parameters

x	Input value in the range [-1,1].

Returns

Weigth function in the range [0,1]

3.13 MHAParser Namespace Reference

Name space for the openMHA-Parser configuration language.

Namespaces

- **StrCnv**
  String converter namespace.

Classes

- class **base_t**
  Base class for all parser items.
- class **bool_mon_t**
  Monitor with string value.
- class **bool_t**
  Variable with a boolean value ("yes"/"no")
- class **commit_t**
  Parser variable with event-emission functionality.
- class **complex_mon_t**
  Monitor with complex value.

- class **complex_t**
  Variable with complex value.
- class **float_mon_t**
  Monitor with float value.
- class **float_t**
  Variable with float value.
- class **int_mon_t**
  Monitor variable with int value.
- class **int_t**
  Variable with integer value.
- class **keyword_list_t**
  Keyword list class.
- class **kw_t**
  Variable with keyword list value.
- class **mcomplex_mon_t**
  Matrix of complex numbers monitor.
- class **mcomplex_t**
  Matrix variable with complex value.
- class **mfloat_mon_t**
  Matrix of floats monitor.
- class **mfloat_t**
  Matrix variable with float value.
- class **mhapluginloader_t**
  Class to create a plugin loader in a parser, including the load logic.
- class **monitor_t**
  Base class for monitors and variable nodes.
- class **parser_t**
  Parser node class.
- class **range_var_t**
  Base class for all variables with a numeric value range.
- class **string_mon_t**
  Monitor with string value.
- class **string_t**
  Variable with a string value.
- class **variable_t**
  Base class for variable nodes.
- class **vcomplex_mon_t**
  Monitor with vector of complex values.
- class **vcomplex_t**
  Vector variable with complex value.
- class **vfloat_mon_t**
  Vector of floats monitor.
- class **vfloat_t**
  Vector variable with float value.
- class **vint_mon_t**

Vector of ints monitor.
- class **vint_t**
  Variable with vector<int> value.
- class **vstring_mon_t**
  Vector of monitors with string value.
- class **vstring_t**
  Vector variable with string values.
- class **window_t**
  MHA configuration interface for a window function generator.

Functions

- void **strreplace** (std::string &, const std::string &, const std::string &)
  string replace function

3.13.1 Detailed Description

This namespace contains all classes which are needed for the implementation of the openMHA configuration language. For details on the script language itself please see section **The openMHA configuration language** (p. 33).

3.13.2 List of valid MHAParser items

- **Sub-parser**: **parser_t** (p. 220)
- **Variables**:
  Numeric variables: **int_t** (p. 211), **vint_t** (p. 232), **float_t** (p. 208), **vfloat_t** (p. 229), **mfloat_t** (p. 218)
  Other variables: **string_t** (p. 225), **vstring_t** (p. 234), **kw_t** (p. 214), **bool_t** (p. 204)
- **Monitors**:
  Numeric monitors: **int_mon_t** (p. 210), **vint_mon_t** (p. 231), **float_mon_t** (p. 207), **vfloat_mon_t** (p. 228)
  mfloat_mon_t (p. 217)
  mcomplex_mon_t (p. 215)
  Other monitors: **bool_mon_t** (p. 203), **string_mon_t** (p. 224), **vstring_mon_t** (p. 233)

Members can be inserted into the configuration namespace by using MHAParser::insert_item() or the **insert_member()** (p. 295) macro.

3.13.3 Function Documentation

3.13.3.1 void MHAParser::strreplace (std::string & *s*, const std::string & *arg*, const std::string & *rep*)

Parameters

s	target string
arg	search pattern
rep	replace pattern

3.14 MHAParser::StrCnv Namespace Reference

String converter namespace.

Functions

- int **num_brackets** (const std::string &s)
  Return number of brackets at beginning and end of string.
- void **str2val** (const std::string &, bool &)
  Convert from string.
- void **str2val** (const std::string &, float &)
  Convert from string.
- void **str2val** (const std::string &, **mha_complex_t** &)
  Convert from string.
- void **str2val** (const std::string &, int &)
  Convert from string.
- void **str2val** (const std::string &, **keyword_list_t** &)
  Convert from string.
- void **str2val** (const std::string &, std::string &)
  Convert from string.
- template<class arg_t >
  void **str2val** (const std::string &s, std::vector< arg_t > &val)
  Converter for vector types.
- template<>
  void **str2val**< **mha_real_t** > (const std::string &s, std::vector< **mha_real_t** > &v)
  Converter for vector<mha_real_t> with Matlab-style expansion.
- template<class arg_t >
  void **str2val** (const std::string &s, std::vector< std::vector< arg_t > > &val)
  Converter for matrix types.
- std::string **val2str** (const bool &)
  Convert to string.
- std::string **val2str** (const float &)
  Convert to string.
- std::string **val2str** (const **mha_complex_t** &)
  Convert to string.
- std::string **val2str** (const int &)
  Convert to string.
- std::string **val2str** (const **keyword_list_t** &)

Convert to string.
- std::string **val2str** (const std::string &)
  Convert to string.
- std::string **val2str** (const std::vector< float > &)
  Convert to string.
- std::string **val2str** (const std::vector< **mha_complex_t** > &)
  Convert to string.
- std::string **val2str** (const std::vector< int > &)
  Convert to string.
- std::string **val2str** (const std::vector< std::string > &)
  Convert to string.
- std::string **val2str** (const std::vector< std::vector< float > > &)
  Convert to string.
- std::string **val2str** (const std::vector< std::vector< **mha_complex_t** > > &)
  Convert to string.

3.14.1 Detailed Description

The functions defined in this namespace manage the conversions from C++ variables to strings and back. It was tried to keep a matlab compatible string format for vectors and vectors of vectors.

3.14.2 Function Documentation

3.14.2.1 int MHAParser::StrCnv::num_brackets (const std::string & *s*)

Parameters

s	String

Returns

Number of brackets, or -1 for empty string

3.15 MHAPlugin Namespace Reference

Namespace for openMHA plugin class templates and thread-safe runtime configurations.

Classes

- class **config_t**
  Template class for thread safe configuration.
- class **plugin_t**
  The template class for C++ openMHA plugins.

3.16 MHASignal Namespace Reference

Namespace for audio signal handling and processing classes.

Classes

- class **async_rmslevel_t**

  Class for asynchronous level metering.
- class **delay_t**

  Class to realize a simple delay of waveform streams.
- class **delay_wave_t**

  Delayline containing wave fragments.
- class **doublebuffer_t**

  Double-buffering class.
- class **hilbert_t**

  Hilbert transformation of a waveform segment.
- class **loop_wavefragment_t**

  Copy a fixed waveform fragment to a series of waveform fragments of other size.
- class **matrix_t**

  n-dimensional matrix with real or complex floating point values.
- class **minphase_t**

  Minimal phase function.
- class **quantizer_t**

  Simple simulation of fixpoint quantization.
- class **ringbuffer_t**

  A ringbuffer class for time domain audio signal, which makes no assumptions with respect to fragment size.
- class **schroeder_t**

  Schroeder tone complex class.
- class **spectrum_t**

  *a signal processing class for spectral data (based on **mha_spec_t** (p. 141))*
- class **subsample_delay_t**

  implements subsample delay in spectral domain.
- class **uint_vector_t**

  Vector of unsigned values, used for size and index description of n-dimensional matrixes.
- class **waveform_t**

  *signal processing class for waveform data (based on **mha_wave_t** (p. 154))*

Functions

- void **for_each** (**mha_wave_t** ∗s, **mha_real_t**(∗fun)(**mha_real_t**))
  *Apply a function to each element of a **mha_wave_t** (p. 154).*
- **mha_real_t lin2db** (**mha_real_t** x)
  Conversion from linear scale to dB (no SPL reference)
- **mha_real_t db2lin** (**mha_real_t** x)
  Conversion from dB scale to linear (no SPL reference)
- **mha_real_t pa2dbspl** (**mha_real_t** x)
  Conversion from linear Pascal scale to dB SPL.
- **mha_real_t pa22dbspl** (**mha_real_t** x, **mha_real_t** eps=1e-20f)
  Conversion from squared Pascal scale to dB SPL.
- **mha_real_t dbspl2pa** (**mha_real_t** x)
  Conversion from dB SPL to linear Pascal scale.
- **mha_real_t smp2sec** (**mha_real_t** n, **mha_real_t** srate)
  conversion from samples to seconds
- **mha_real_t sec2smp** (**mha_real_t** sec, **mha_real_t** srate)
  conversion from seconds to samples
- **mha_real_t bin2freq** (**mha_real_t** bin, unsigned fftlen, **mha_real_t** srate)
  conversion from fft bin index to frequency
- **mha_real_t freq2bin** (**mha_real_t** freq, unsigned fftlen, **mha_real_t** srate)
  conversion from frequency to fft bin index
- **mha_real_t smp2rad** (**mha_real_t** samples, unsigned bin, unsigned fftlen)
  conversion from delay in samples to phase shift
- **mha_real_t rad2smp** (**mha_real_t** phase_shift, unsigned bin, unsigned fftlen)
  conversion from phase shift to delay in samples
- template< class elem_type >
  std::vector< elem_type > **dupvec** (std::vector< elem_type > vec, unsigned n)
  Duplicate last vector element to match desired size.
- template< class elem_type >
  std::vector< elem_type > **dupvec_chk** (std::vector< elem_type > vec, unsigned n)
  Duplicate last vector element to match desired size, check for dimension.
- void **copy_channel** (**mha_spec_t** &self, const **mha_spec_t** &src, unsigned sch, unsigned dch)
  Copy one channel of a source signal.
- void **copy_channel** (**mha_wave_t** &self, const **mha_wave_t** &src, unsigned src_channel, unsigned dest_channel)
  Copy one channel of a source signal.
- **mha_real_t rmslevel** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen)
  Return RMS level of a spectrum channel.
- **mha_real_t colored_intensity** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen, **mha_real_t** sqfreq_response[ ])
  Colored spectrum intensity.
- **mha_real_t maxabs** (const **mha_spec_t** &s, unsigned int channel)
  Find maximal absolute value.
- **mha_real_t rmslevel** (const **mha_wave_t** &s, unsigned int channel)

Return RMS level of a waveform channel.
- **mha_real_t maxabs** (const **mha_wave_t** &s, unsigned int channel)
  Find maximal absolute value.
- **mha_real_t maxabs** (const **mha_wave_t** &s)
  Find maximal absolute value.
- **mha_real_t max** (const **mha_wave_t** &s)
  Find maximal value.
- **mha_real_t min** (const **mha_wave_t** &s)
  Find minimal value.
- **mha_real_t sumsqr_channel** (const **mha_wave_t** &s, unsigned int channel)
  Calculate sum of squared values in one channel.
- **mha_real_t sumsqr_frame** (const **mha_wave_t** &s, unsigned int frame)
  Calculate sum over all channels of squared values.
- void **limit** (**mha_wave_t** &s, const **mha_real_t** &**min**, const **mha_real_t** &**max**)
  Limit the singal in the waveform buffer to the range [min, max].
- template<class elem_type >
  elem_type **kth_smallest** (elem_type array[ ], unsigned n, unsigned k)
  Fast search for the kth smallest element of an array.
- template<class elem_type >
  elem_type **median** (elem_type array[ ], unsigned n)
  Fast median search.
- template<class elem_type >
  elem_type **mean** (const std::vector< elem_type > &data, elem_type start_val)
  Calculate average of elements in a vector.
- template<class elem_type >
  std::vector< elem_type > **quantile** (std::vector< elem_type > data, const std::vector< elem_type > &p)
  Calculate quantile of elements in a vector.
- void **saveas_mat4** (const **mha_spec_t** &data, const std::string &varname, FILE *fh)
  Save a openMHA spectrum as a variable in a Matlab4 file.
- void **saveas_mat4** (const **mha_wave_t** &data, const std::string &varname, FILE *fh)
  Save a openMHA waveform as a variable in a Matlab4 file.
- void **saveas_mat4** (const std::vector< **mha_real_t** > &data, const std::string &varname, FILE *fh)
  Save a float vector as a variable in a Matlab4 file.
- void **copy_permuted** (**mha_wave_t** *dest, const **mha_wave_t** *src)
  Copy contents of a waveform to a permuted waveform.

Variables

- unsigned long int **signal_counter** = 0
  Signal counter to produce signal ID strings.

3.16.1 Function Documentation

3.16.1.1 void MHASignal::limit (mha_wave_t & *s*, const mha_real_t & *min*, const mha_real_t & *max*)

Parameters

s	The signal to limit. The signal in this wave buffer is modified.
min	lower limit
max	upper limit

3.16.1.2 template<class elem_type > elem_type MHASignal::kth_smallest (elem_type *array[ ],* unsigned *n,* unsigned *k*)

The order of elements is altered, but not completely sorted. Using the algorithm from N. Wirth, published in "Algorithms + data structures = programs", Prentice-Hall, 1976

Parameters

array	Element array

Postcondition

The order of elements in the array is altered. array[k] then holds the result.

Parameters

n	number of elements in array

Precondition

n >= 1

Parameters

k	The k'th smalles element is returned: k = 0 returns the minimum, k = (n-1)/2 returns the median, k=(n-1) returns the maximum

Precondition

k < n

Returns

The kth smallest array element

3.16.1.3 template<class elem_type > elem_type MHASignal::median (elem_type *array[ ],* unsigned *n*) [inline]

The order of elements is altered, but not completely sorted.

Parameters

array	Element array

Postcondition

The order of elements in the array is altered. array[(n-1)/2] then holds the median.

Parameters

n	number of elements in array

Precondition

n >= 1

Returns

The median of the array elements

3.16.1.4 template< class elem_type > elem_type MHASignal::mean (const std::vector< elem_type > & *data*, elem_type *start_val*) [inline]

Parameters

data	Input vector
start_val	Value for initialization of the return value before sum.

Returns

The average of the vector elements

3.16.1.5 template< class elem_type > std::vector<elem_type> MHASignal::quantile (std::vector< elem_type > *data*, const std::vector< elem_type > & *p*) [inline]

Parameters

data	Input vector
p	Vector of probability values.

Returns

Vector of quantiles of input data, one entry for each probability value.

3.16.1.6 void MHASignal::saveas_mat4 (const mha_spec_t & *data,* const std::string & *varname,* FILE ∗ *fh*)

Parameters

data	openMHA spectrum to be saved.
varname	Matlab variable name (Matlab4 limitations on maximal length are not checked).
fh	File handle to Matlab4 file.

3.16.1.7 void MHASignal::saveas_mat4 (const mha_wave_t & *data,* const std::string & *varname,* FILE ∗ *fh*)

Parameters

data	openMHA waveform to be saved.
varname	Matlab variable name (Matlab4 limitations on maximal length are not checked).
fh	File handle to Matlab4 file.

3.16.1.8 void MHASignal::saveas_mat4 (const std::vector< mha_real_t > & *data,* const std::string & *varname,* FILE ∗ *fh*)

Parameters

data	Float vector to be saved.
varname	Matlab variable name (Matlab4 limitations on maximal length are not checked).
fh	File handle to Matlab4 file.

3.16.1.9 void MHASignal::copy_permuted (mha_wave_t ∗ *dest,* const mha_wave_t ∗ *src*)

Parameters

dest	Destination waveform
src	Source waveform

The total size of src and dest must be the same, num_frames and num_channels must be exchanged in dest.

3.17 MHATableLookup Namespace Reference

Namespace for table lookup classes.

Classes

- class **xy_table_t**
  Class for interpolation with non-equidistant x values.

3.18 MHAWindow Namespace Reference

Collection of Window types.

Classes

- class **bartlett_t**
  Bartlett window.
- class **base_t**
  Common base for window types.
- class **blackman_t**
  Blackman window.
- class **fun_t**
  Generic window based on a generator function.
- class **hamming_t**
  Hamming window.
- class **hanning_t**
  von-Hann window
- class **rect_t**
  Rectangular window.
- class **user_t**
  User defined window.

Functions

- float **rect** (float)
  Rectangular window function.
- float **bartlett** (float)
  Bartlett window function.
- float **hanning** (float)
  Hanning window function.
- float **hamming** (float)
  Hamming window function.
- float **blackman** (float)
  Blackman window function.

4 Class Documentation

4.1 algo_comm_t Struct Reference

A reference handle for algorithm communication variables.

Public Attributes

- void ∗ **handle**
  AC variable control handle.
- int(∗ **insert_var**)(void ∗, const char ∗, **comm_var_t**)
  Register an AC variable.
- int(∗ **insert_var_int**)(void ∗, const char ∗, int ∗)
  Register an int as an AC variable.
- int(∗ **insert_var_float**)(void ∗, const char ∗, float ∗)
  Register a float as an AC variable.
- int(∗ **remove_var**)(void ∗, const char ∗)
  Remove an AC variable.
- int(∗ **remove_ref**)(void ∗, void ∗)
  Remove all AC variable which refer to address.
- int(∗ **is_var**)(void ∗, const char ∗)
  Test if an AC variable exists.
- int(∗ **get_var**)(void ∗, const char ∗, **comm_var_t** ∗)
  Get the variable handle of an AC variable.
- int(∗ **get_var_int**)(void ∗, const char ∗, int ∗)
  Get the value of an int AC variable.
- int(∗ **get_var_float**)(void ∗, const char ∗, float ∗)
  Get the value of a float AC variable.
- int(∗ **get_entries**)(void ∗, char ∗, unsigned int)
  Return a space separated list of all variable names.
- const char ∗(∗ **get_error**)(int)
  Convert AC error codes into human readable error messages.

4.1.1 Detailed Description

This structure contains a coontrol handle and a set of function pointers for sharing variables within one processing chain. See also section **Communication between algorithms** (p. 27).

4.1.2 Member Data Documentation

4.1.2.1 algo_comm_t::insert_var

This function can register a variable to be shared within one chain. If a variable of this name exists it will be overwritten.

Parameters

h	AC handle
n	name of variable. May not be empty. Must not contain space character. The name is copied, therefore it is allowed that the char array pointed to gets invalid after return.
v	variable handle of type **comm_var_t** (p. 95)

Returns

Error code or zero on success

4.1.2.2 algo_comm_t::insert_var_int

This function can register an int variable to be shared with other algorithms. It behaves similar to ac.insert_var.

Parameters

h	AC handle
n	name of variable
v	pointer on the variable

Returns

Error code or zero on success

4.1.2.3 algo_comm_t::insert_var_float

This function can register a float variable to be shared with other algorithms. It behaves similar to ac.insert_var.

Parameters

h	AC handle
n	name of variable
v	pointer on the variable

Returns

Error code or zero on success

4.1.2.4 algo_comm_t::remove_var

Remove (unregister) an AC variable. After calling this function, the variable is not available to ac.is_var or ac.get_var. The data pointer is not affected.

Parameters

h	AC handle
n	name of variable to be removed

Returns

Error code or zero on success

4.1.2.5 algo_comm_t::remove_ref

This function removes all AC variables whos data field points to the given address.

Parameters

h	AC handle
p	address which should not be referred to any more

Returns

Error code or zero on success

4.1.2.6 algo_comm_t::is_var

This function tests if an AC variable of a given name exists. Use ac.get_var to get information about the variables type and dimension.

Parameters

h	AC handle
n	name of variable

Returns

1 if the variable exists, 0 otherwise

4.1.2.7 algo_comm_t::get_var

This function returns the variable handle **comm_var_t** (p. 95) of a variable of the given name. If no variable of that name exists, an error code is returned.

Parameters

h	AC handle
n	name of variable
v	pointer to a AC variable object

Returns

Error code or zero on success

4.1.2.8 algo_comm_t::get_var_int

This function returns the value of an int AC variable of the given name. If no variable exists, the variable type is mismatching or more than one entry is registered, a corresponding error code is returned. This is a special version of ac.get_var.

Parameters

h	AC handle
n	name of variable
v	pointer on an int variable to store the result

Returns

Error code or zero on success

4.1.2.9 algo_comm_t::get_var_float

This function returns the value of a float AC variable of the given name. If no variable exists, the variable type is mismatching or more than one entry is registered, a corresponding error code is returned. This is a special version of ac.get_var.

Parameters

h	AC handle
n	name of variable
v	pointer on a float variable to store the result

Returns

Error code or zero on success

4.1.2.10 algo_comm_t::get_entries

This function returns the names of all registered variables, separated by a single space.

Parameters

h	AC handle

Return values

ret	Character buffer for return value

Parameters

len	length of character buffer

Returns

Error code or zero on success. -1: invalid ac handle. -3: not enough room in character buffer to store all variable names.

4.1.2.11 algo_comm_t::get_error

Parameters

e	Error code

Returns

Error message

4.2 AuditoryProfile::fmap_t Class Reference

A class to store frequency dependent data (e.g., HTL and UCL).

Inherits map< mha_real_t, mha_real_t >.

Public Member Functions

- std::vector< **mha_real_t** > **get_frequencies** () const

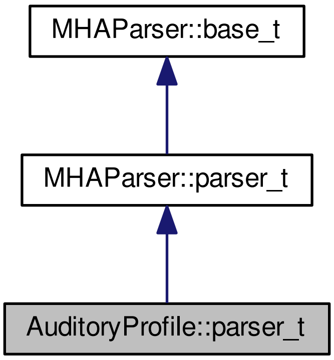
  Return configured frequencies.
- std::vector< **mha_real_t** > **get_values** () const

  Return stored values corresponding to the frequencies.

4.3 AuditoryProfile::parser_t Class Reference

Class to make the auditory profile accessible through the parser interface.

Inheritance diagram for AuditoryProfile::parser_t:



Additional Inherited Members

4.4 AuditoryProfile::profile_t Class Reference

The Auditory Profile class.

Classes

- class **ear_t**

  Class for ear-dependent parameters, e.g., audiograms or unilateral loudness scaling.

Public Member Functions

- **AuditoryProfile::profile_t::ear_t get_ear** (unsigned int channel) const

  Return ear information of channel number.

Public Attributes

- **AuditoryProfile::profile_t::ear_t L**

  Left ear data.
- **AuditoryProfile::profile_t::ear_t R**

  Right ear data.

4.4.1 Detailed Description

See definition of auditory profile

Currently only the audiogram data is stored.

4.5 AuditoryProfile::profile_t::ear_t Class Reference

Class for ear-dependent parameters, e.g., audiograms or unilateral loudness scaling.

4.6 comm_var_t Struct Reference

Algorithm communication variable structure.

Public Attributes

- unsigned int **data_type**

  Type of data.
- unsigned int **num_entries**

  Number of entries.
- unsigned int **stride**

  length of one row (C interpretation) or of one column (Fortran interpretation)
- void ∗ **data**

  Pointer to variable data.

4.6.1 Detailed Description

Algorithm communication variables (AC variables) are objects of this type. The member data is a pointer to the variable 'data'. This pointer has to be valid for the lifetime of this AC variable. The member 'data_type' can be one of the predefined types or any user defined type. The member 'num_entries' describes the number of elements of this base type stored at the pointer address.

```
An AC variable can be registered with the \ref
algo_comm_t::insert_var "insert_var" function.
```

4.6.2 Member Data Documentation

4.6.2.1 comm_var_t::data_type

This can be one of the predefined types

- MHA_AC_CHAR
- MHA_AC_INT
- MHA_AC_MHAREAL
- MHA_AC_FLOAT
- MHA_AC_DOUBLE
- MHA_AC_MHACOMPLEX
- MHA_AC_VEC_FLOAT or any user defined type with a value greater than
- MHA_AC_USER

4.7 DynComp::dc_afterburn_rt_t Class Reference

Real-time class for after burn effect.

Public Member Functions

- void **burn** (float &Gin, float Lin, unsigned int band, unsigned int channel)
  gain modifier method (afterburn).

4.7.1 Detailed Description

The constructor processes the parameters and creates pre-processed variables for efficient realtime processing.

4.7.2 Member Function Documentation

4.7.2.1 void DynComp::dc_afterburn_rt_t::burn (float & *Gin*, float *Lin*, unsigned int *band*, unsigned int *channel*) [inline]

Parameters

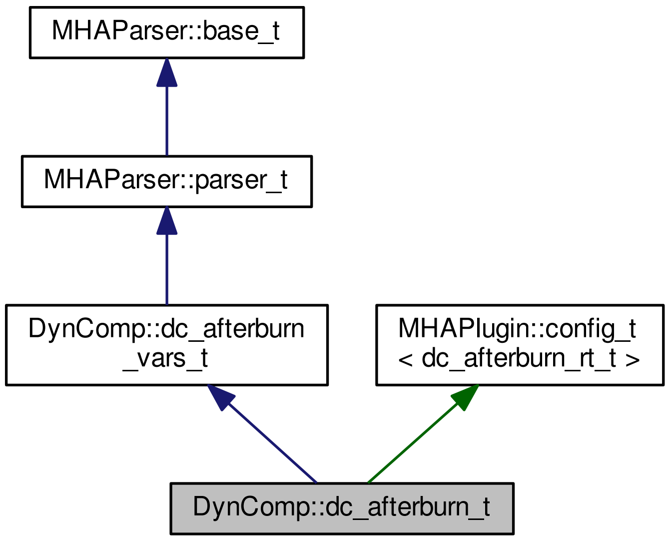
Gin	Linear gain.
Lin	Input level (Pascal).
band	Filter band number.
channel	Channel number.

Output level for MPO is estimated by Gin $*$ Lin.

4.8 DynComp::dc_afterburn_t Class Reference

Afterburn class, to be defined as a member of compressors.

Inheritance diagram for DynComp::dc_afterburn_t:

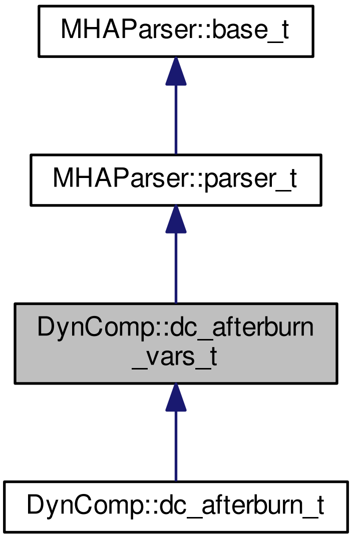


Additional Inherited Members

4.9 DynComp::dc_afterburn_vars_t Class Reference

Variables for **dc_afterburn_t** (p. 97) class.

Inheritance diagram for DynComp::dc_afterburn_vars_t:



Additional Inherited Members

4.10 DynComp::gaintable_t Class Reference

Gain table class.

Public Member Functions

- **gaintable_t** (const std::vector< **mha_real_t** > &LInput, const std::vector< **mha_real_t** > &FCenter, unsigned int **channels**)
  Constructor.
- void **update** (std::vector< std::vector< std::vector< **mha_real_t** > > > newGain)
  Update gains from an external table.
- **mha_real_t get_gain** (**mha_real_t** Lin, **mha_real_t** Fin, unsigned int channel)
  Read Gain from gain table.
- **mha_real_t get_gain** (**mha_real_t** Lin, unsigned int band, unsigned int channel)
  Read Gain from gain table.
- void **get_gain** (const **mha_wave_t** &Lin, **mha_wave_t** &Gain)
  Read Gains from gain table.
- unsigned int **nbands** () const
  Return number of frequency bands.
- unsigned int **nchannels** () const
  Return number of audio channels.
- std::vector< std::vector< **mha_real_t** > > **get_iofun** () const
  Return current input-output function.

4.10.1 Detailed Description

This gain table is intended to efficient table lookup, i.e, interpolation of levels, and optional interpolation of frequencies. Sample input levels and sample frequencies are given in the constructor. The gain entries can be updated with the **update()** (p. 99) member function via a gain prescription rule from an auditory profile.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 gaintable_t::gaintable_t (const std::vector< mha_real_t > & *LInput*, const std::vector< mha_real_t > & *FCenter*, unsigned int *channels*)

Parameters

LInput	Input level samples, in equivalent LTASS_combined dB SPL.
FCenter	Frequency samples in Hz (e.g., center frequencies of filterbank).
channels	Number of audio channels (typically 2).

4.10.3 Member Function Documentation

4.10.3.1 void gaintable_t::update (std::vector< std::vector< std::vector< mha_real_t > > > *newGain*)

Parameters

newGain	New gain table entries.

Dimension change is not allowed. The number of entries are checked.

4.10.3.2 mha_real_t gaintable_t::get_gain (mha_real_t *Lin*, mha_real_t *Fin*, unsigned int *channel*)

Parameters

Lin	Input level
Fin	Input frequency (no match required)
channel	Audio channel

4.10.3.3 mha_real_t gaintable_t::get_gain (mha_real_t *Lin*, unsigned int *band*, unsigned int *channel*)

Parameters

Lin	Input level
band	Input frequency band
channel	Audio channel

4.10.3.4 void gaintable_t::get_gain (const mha_wave_t & *Lin*, mha_wave_t & *Gain*)

Parameters

Lin	Input levels.
Gain	Output gain.

The number of channels in Lin and Gain must match the number of bands times number of channels in the gaintable.

4.11 expression_t Class Reference

Class for separating a string into a left hand value and a right hand value.

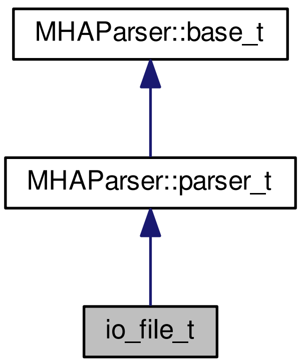
4.11.1 Detailed Description

A list of valid operators can be provided. After construction, the class members lval, rval and op contain the apropriate contents.

4.12 io_file_t Class Reference

File IO.

Inheritance diagram for io_file_t:



Public Member Functions

- void **prepare** (int, int)

  Allocate buffers, activate FILE client and install internal ports.
- void **release** ()

  Remove FILE client and deallocate internal ports and buffers.

Additional Inherited Members

4.12.1 Member Function Documentation

4.12.1.1 void io_file_t::prepare (int *nch_in,* int *nch_out*)

4.12.1.2 void io_file_t::release ( )

4.13 io_lib_t Class Reference

Class for loading MHA sound IO module.

Inherits MHAParser::c_ifc_parser_t.

Public Member Functions

- **io_lib_t** (int fragsize, float samplerate, IOProcessEvent_t proc_event, void ∗proc_handle, IOStartedEvent_t start_event, void ∗start_handle, IOStoppedEvent_t stop_event, void ∗stop_handle, std::string libname)

  load and initialize MHA sound io module.
- ∼**io_lib_t** ()

  Deinitialize and unload this MHA sound io module.
- void **prepare** (unsigned int inch, unsigned int outch)

  Prepare the sound io module.
- void **start** ()

  Tell the sound io module to start sound processing.

4.13.1 Detailed Description

4.13.2 Member Function Documentation

4.13.2.1 void io_lib_t::prepare (unsigned int *inch,* unsigned int *outch*)

After preparation, the sound io module may start the sound processing at any time (external trigger). When the sound processing is started, the sound io module will call the start_event callback.

Parameters

inch	number of input channels
outch	number of output channels

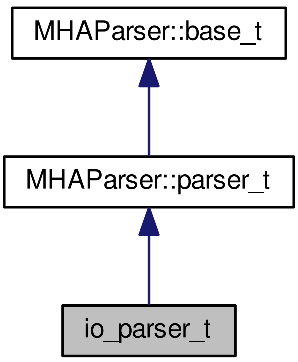
4.13.2.2 void io_lib_t::start ( )

Some io modules need this, for others that wait for external events this method might do nothing.

4.14 io_parser_t Class Reference

Main class for Parser IO.

Inheritance diagram for io_parser_t:



Public Member Functions

- void **prepare** (int, int)

  Allocate buffers, activate JACK client and install internal ports.
- void **release** ()

  Remove JACK client and deallocate internal ports and buffers.

Additional Inherited Members

4.14.1 Detailed Description

4.14.2 Member Function Documentation

4.14.2.1 void io_parser_t::prepare (int *nch_in*, int *nch_out*)

4.14.2.2 void io_parser_t::release ( )

4.15 io_tcp_fwcb_t Class Reference

TCP sound-io library's interface to the framework callbacks.

Public Member Functions

- **io_tcp_fwcb_t** (IOProcessEvent_t **proc_event**, void ∗**proc_handle**, IOStartedEvent_↩
t **start_event**, void ∗start_handle, IOStoppedEvent_t **stop_event**, void ∗stop_handle)

  Constructor stores framework handles and initializes error numbers to 0.
- virtual ∼**io_tcp_fwcb_t** ()

  Do-nothing destructor.
- virtual void **start** ()

  Call the framework's start callback.
- virtual int **process** (**mha_wave_t** ∗sIn, **mha_wave_t** ∗&sOut)

  Call the frameworks processing callback.
- virtual void **set_errnos** (int **proc_err**, int io_err)

  Save error numbers to use during.
- virtual void **stop** ()

  Call the frameworks stop callback.

Private Attributes

- IOProcessEvent_t **proc_event**

  Pointer to signal processing callback function.
- IOStartedEvent_t **start_event**

  Pointer to start notification callback function.
- IOStoppedEvent_t **stop_event**

  Pointer to stop notification callback function.
- void ∗ **proc_handle**

  Handles belonging to framework.
- int **proc_err**

  Errors from the processing callback and from the TCP IO itself are stored here before closing Network handles.

4.15.1 Detailed Description

4.15.2 Constructor & Destructor Documentation

4.15.2.1 io_tcp_fwcb_t::io_tcp_fwcb_t (IOProcessEvent_t *proc_event*, void ∗ *proc_handle*, IOStartedEvent_t *start_event*, void ∗ *start_handle*, IOStoppedEvent_t *stop_event*, void ∗ *stop_handle*)

4.15.2.2 virtual io_tcp_fwcb_t::∼io_tcp_fwcb_t ( ) [inline], [virtual]

4.15.3 Member Function Documentation

4.15.3.1 void io_tcp_fwcb_t::start ( ) [virtual]

4.15.3.2 int io_tcp_fwcb_t::process (mha_wave_t ∗ *sIn*, mha_wave_t ∗& *sOut*) [virtual]

Parameters

sIn	The input sound data just received from TCP.
sOut	A pointer to output sound data. Will point to the output sound data storage when the callback finishes.

Returns

Status, an error number from the signal processing callback. If this is != 0, then the connection should be closed.

4.15.3.3 void io_tcp_fwcb_t::set_errnos (int *proc_err*, int *io_err*) [virtual]

See also

stop (p. 105)

Parameters

proc_err	The error number from the

See also

process (p. 104) callback.

Parameters

io_err	The error number from the io library itself.

4.15.3.4 void io_tcp_fwcb_t::stop ( ) [virtual]

Uses the error numbers set previously with

See also

> **set_errnos** (p. 104).

4.15.4 Member Data Documentation

4.15.4.1 IOProcessEvent_t io_tcp_fwcb_t::proc_event [private]

4.15.4.2 IOStartedEvent_t io_tcp_fwcb_t::start_event [private]

Called when a new TCP connection is established or the user issues the start command while there is a connection.

4.15.4.3 IOStoppedEvent_t io_tcp_fwcb_t::stop_event [private]

Called when the connection is closed.

4.15.4.4 void* io_tcp_fwcb_t::proc_handle [private]

4.15.4.5 int io_tcp_fwcb_t::proc_err [private]

MHAIOTCP is notified by the server when the connection has been taken down, and calls

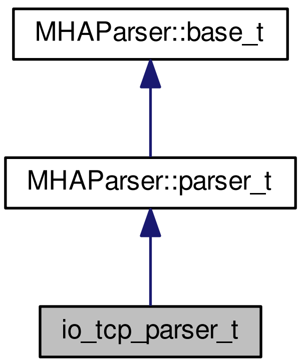
See also

> **stop** (p. 105) from that callback. Within **stop** (p. 105), these error numbers are read again and transmitted to the framework.

4.16 io_tcp_parser_t Class Reference

The parser interface of the IOTCP library.

Inheritance diagram for io_tcp_parser_t:



Public Member Functions

- virtual const std::string & **get_local_address** () const
  Read parser variable local_address, this is the address of the network interface that should listen for incoming connections.
- virtual unsigned short **get_local_port** () const
  Read parser variable local_port, this is the TCP port that should be used for incoming connections.
- virtual void **set_local_port** (unsigned short port)
  Set parser variable local_port.
- virtual bool **get_server_port_open** () const
  Return the status of the server port as it is known to the parser.
- virtual void **set_server_port_open** (bool open)
  Inform the parser of the current status of the server socket.
- virtual bool **get_connected** () const
  Return the parser's knowledge concerning wether there currently exists an established sound data TCP connection or not.
- virtual void **set_connected** (bool **connected**)
  Inform the parser about the existance of a sound data connection.
- virtual void **set_new_peer** (unsigned short port, const std::string &host)
  Set parser monitor variables peer_port and peer_address, and calls set_connected(true).
- **io_tcp_parser_t** ()
  Constructor initializes parser variables.
- virtual ∼**io_tcp_parser_t** ()
  Do-nothing destructor.

Private Attributes

- **MHAParser::string_t local_address**
  Lets the user set the local network interface to listen on.
- **MHAParser::int_t local_port**
  Lets the user choose the local tcp port to listen on.
- **MHAParser::int_mon_t server_port_open**
  Indicates wether the TCP server socket is currently open.
- **MHAParser::int_mon_t connected**
  Indicator if there currently is a sound data connection over TCP.
- **MHAParser::string_mon_t peer_address**
  Display the ip address of the currently connected sound data client.
- **MHAParser::int_mon_t peer_port**
  Display the tcp port used by the current sound data client.
- **MHAParser::string_t debug_filename**
  filename to write debugging info to (if non-empty)
- FILE ∗ **debug_file**
  file handle to write debugging info to

Additional Inherited Members

4.16.1 Detailed Description

4.16.2 Constructor & Destructor Documentation

4.16.2.1 io_tcp_parser_t::io_tcp_parser_t ( )

4.16.2.2 virtual io_tcp_parser_t::~io_tcp_parser_t ( ) [inline], [virtual]

4.16.3 Member Function Documentation

4.16.3.1 virtual const std::string& io_tcp_parser_t::get_local_address ( ) const [inline], [virtual]

Returns

A string containing the address of the local interface as it was set by the user.

4.16.3.2 unsigned short io_tcp_parser_t::get_local_port ( ) const [virtual]

Returns

The local tcp port to listen on as it was chosen by the user. The port number is between MIN_TCP_PORT and MAX_TCP_PORT.

4.16.3.3 void io_tcp_parser_t::set_local_port (unsigned short *port*) [virtual]

This is needed when it was set to 0 before: In this case, the OS chooses a free port for the TCP server socket, and the port that it chose has to be published to the user via the parser interface.

Parameters

port	The TCP port number that is currently used. In the range [MIN_TCP_PORT, MAX_TCP_PORT], excluding 0.

Precondition

get_local_port() (p. 107) currently returns 0.

4.16.3.4 bool io_tcp_parser_t::get_server_port_open ( ) const [virtual]

Returns

false after initialization, or the value most recently set via

See also

set_server_port_open (p. 108).

4.16.3.5 void io_tcp_parser_t::set_server_port_open (bool *open*) [virtual]

Parameters

open	Indicates wether the server socket has just been opened or closed.

Precondition

open may only have the value true if **get_server_port_open()** (p. 107) currently returns false.

Postcondition

See also

get_server_port_open (p. 107) returns the **value** (p. 48) of open.

4.16.3.6 bool io_tcp_parser_t::get_connected ( ) const [virtual]

Returns

false after initialization, or the value most recently set via

See also

set_connected (p. 108).

4.16.3.7 void io_tcp_parser_t::set_connected (bool *connected*) [virtual]

Parameters

connected	Indicates wether there currently is a connection or not.

Precondition

connected must not have the same value that is currently returned by

See also

get_connected (p. 108).

Postcondition

See also

get_connected (p. 108) returns the **value** (p. 48) of open.

4.16.3.8 void io_tcp_parser_t::set_new_peer (unsigned short *port*, const std::string & *host*) [virtual]

This method should be called when a new connection is established.

Parameters

port	The TCP port number used by the peer.
host	The Internet host where the peer is located.

Precondition

See also

get_connected (p. 108) currently returns false.

Postcondition

See also

get_connected (p. 108) returns true.

4.16.4 Member Data Documentation

4.16.4.1 MHAParser::string_t io_tcp_parser_t::local_address [private]

4.16.4.2 MHAParser::int_t io_tcp_parser_t::local_port [private]

4.16.4.3 MHAParser::int_mon_t io_tcp_parser_t::server_port_open [private]

4.16.4.4 MHAParser::int_mon_t io_tcp_parser_t::connected [private]

4.16.4.5 MHAParser::string_mon_t io_tcp_parser_t::peer_address [private]

4.16.4.6 MHAParser::int_mon_t io_tcp_parser_t::peer_port [private]

4.17 io_tcp_sound_t Class Reference

Sound data handling of io tcp library.

Classes

- union **float_union**

  This union helps in conversion of floats from host byte order to network byte order and back again.

Public Member Functions

- **io_tcp_sound_t** (int **fragsize**, float **samplerate**)

  Initialize sound data handling.
- virtual ∼**io_tcp_sound_t** ()

  Do-nothing destructor.
- virtual void **prepare** (int **num_inchannels**, int num_outchannels)

  Called during prepare, sets number of audio channels and allocates sound data storage.
- virtual void **release** ()

  Called during release.
- virtual int **chunkbytes_in** () const

  Number of bytes that constitute one input sound chunk.
- virtual std::string **header** () const

  Create the tcp sound header lines.
- virtual **mha_wave_t** ∗ **ntoh** (const std::string &data)

  *Copy data received from tcp into **mha_wave_t** (p. 154) structure.*
- virtual std::string **hton** (const **mha_wave_t** ∗s_out)

  Copy sound data from the output sound structure to a string.

Static Private Member Functions

- static void **check_sound_data_type** () throw (MHA_Error)
  Check if mha_real_t is a usable 32-bit floating point type.

Private Attributes

- int **fragsize**
  Number of sound samples in each channel expected and returned from processing callback.
- float **samplerate**
  Sampling rate.
- int **num_inchannels**
  Number of input channels.
- **MHASignal::waveform_t** $*$ **s_in**
  Storage for input signal.

4.17.1 Constructor & Destructor Documentation

4.17.1.1 io_tcp_sound_t::io_tcp_sound_t (int *fragsize*, float *samplerate*)

Checks sound data type by calling

See also

check_sound_data_type (p. 111).

Parameters

fragsize	Number of sound samples in each channel expected and returned from processing callback.
samplerate	Number of samples per second in each channel.

4.17.2 Member Function Documentation

4.17.2.1 void io_tcp_sound_t::check_sound_data_type ( ) throw MHA_Error) [static], [private]

Exceptions

MHA_Error *(p. 132)*	if mha_real_t is not compatible to 32-bit float.

4.17.2.2 void io_tcp_sound_t::prepare (int *num_inchannels,* int *num_outchannels*) [virtual]

Parameters

num_inchannels	Number of input audio channels.
num_outchannels	Number of output audio channels.

4.17.2.3 void io_tcp_sound_t::release ( ) [virtual]

Deletes sound data storage.

4.17.2.4 int io_tcp_sound_t::chunkbytes_in ( ) const [virtual]

Returns

Number of bytes to read from TCP connection before invoking signal processing.

4.17.2.5 std::string io_tcp_sound_t::header ( ) const [virtual]

4.17.2.6 mha_wave_t * io_tcp_sound_t::ntoh (const std::string & *data*) [virtual]

Doing network-to-host byte order swapping in the process.

Parameters

data	One chunk (

See also

chunkbytes_in (p. 112)) of sound data to process.

Returns

Pointer to the sound data storage.

4.17.2.7 std::string io_tcp_sound_t::hton (const mha_wave_t * *s_out*) [virtual]

Doing host-to-network byte order swapping while at it.

Parameters

s_out	Pointer to the storage of the sound to put out.

Returns

The sound data in network byte order.

4.17.3 Member Data Documentation

4.17.3.1 int io_tcp_sound_t::fragsize [private]

4.17.3.2 float io_tcp_sound_t::samplerate [private]

Number of samples per second in each channel.

4.17.3.3 int io_tcp_sound_t::num_inchannels [private]

Number of channels expected from and returned by signal processing callback.

4.17.3.4 MHASignal::waveform_t* io_tcp_sound_t::s_in [private]

4.18 io_tcp_sound_t::float_union Union Reference

This union helps in conversion of floats from host byte order to network byte order and back again.

4.18.1 Detailed Description

4.19 io_tcp_t Class Reference

The tcp sound io library.

Public Member Functions

- void **prepare** (int num_inchannels, int num_outchannels)
  Allocate server socket and start thread waiting for sound data exchange.
- void **start** ()
  Call frameworks start callback if there is a sound data connection at the moment.
- void **stop** ()
  Close the current connection if there is one.
- void **release** ()
  Close the current connection and close the server socket.
- virtual void **accept_loop** ()
  IO thread executes this method.
- virtual void **connection_loop** (**MHA_TCP::Connection** *c)
  IO thread executes this method for each connection.
- virtual void **parse** (const char *cmd, char *retval, unsigned int len)
  Parser interface.

4.19.1 Detailed Description

4.19.2 Member Function Documentation

4.19.2.1 void io_tcp_t::prepare (int *num_inchannels,* int *num_outchannels*)

prepare opens the tcp server socket and starts the io thread that listens for audio data on the tcp socket after doing some sanity checks

4.19.2.2 void io_tcp_t::start ( )

4.19.2.3 void io_tcp_t::stop ( )

stop IO thread

4.19.2.4 void io_tcp_t::release ( )

Stop IO thread and close server socket.

4.19.2.5 void io_tcp_t::accept_loop ( ) [virtual]

4.19.2.6 void io_tcp_t::connection_loop (MHA_TCP::Connection * *c*) [virtual]

Parameters

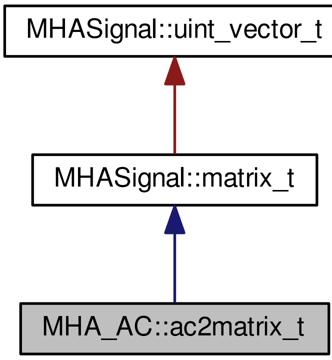
c	pointer to connection. connection_loop deletes connection before exiting.

4.19.2.7 virtual void io_tcp_t::parse (const char * *cmd,* char * *retval,* unsigned int *len*) [inline], [virtual]

4.20 MHA_AC::ac2matrix_t Class Reference

Copy AC variable to a matrix.

Inheritance diagram for MHA_AC::ac2matrix_t:



Public Member Functions

- **ac2matrix_t** (**algo_comm_t** ac, const std::string &name)

  Constructor.
- void **update** ()

  Update contents of the matrix from the AC space.
- const std::string & **getname** () const

  Return name of AC variable/matrix.
- const std::string & **getusername** () const

  Return user specified name of AC variable/matrix.
- void **insert** (**algo_comm_t** ac)

  Insert matrix into an AC space (other than source AC space)

4.20.1 Detailed Description

This class constructs a matrix of same size as an AC variable and can copy the AC variable to itself. The **update()** (p. 116) function is real-time safe.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 MHA_AC::ac2matrix_t::ac2matrix_t (algo_comm_t *ac*, const std::string & *name*)

Parameters

ac	AC handle
name	Name of AC variable to be copied

4.20.3 Member Function Documentation

4.20.3.1 void MHA_AC::ac2matrix_t::update ( )

This function is real-time safe. The copy operation performance is of the order of the number of elements in the matrix.

4.20.3.2 void MHA_AC::ac2matrix_t::insert (algo_comm_t *ac*)

Parameters

ac	AC space handle to insert data

Note

The AC variable data buffer points to the data of the matrix. Modifications of the AC variable directly modify the data of the matrix; after deletion of the matrix, the data buffer is invalid.

4.21 MHA_AC::acspace2matrix_t Class Reference

Copy all or a subset of all numeric AC variables into an array of matrixes.

Public Member Functions

- **acspace2matrix_t** (**algo_comm_t** ac, const std::vector< std::string > &names)
  Constructor.
- **acspace2matrix_t** (const **MHA_AC::acspace2matrix_t** &src)
  Constructor with initialization from an instance.
- **MHA_AC::acspace2matrix_t** & **operator=** (const **MHA_AC::acspace2matrix_t** &src)
  Copy all contents (deep copy).
- **MHA_AC::ac2matrix_t** & **operator[ ]** (unsigned int k)
  Access operator.
- const **MHA_AC::ac2matrix_t** & **operator[ ]** (unsigned int k) const
  Constant access operator.
- void **update** ()
  Update function.
- unsigned int **size** () const
  Number of matrixes in AC space.
- unsigned int **frame** () const
  Actual frame number.
- void **insert** (**algo_comm_t** ac)
  Insert AC space copy into an AC space (other than source AC space)

4.21.1 Constructor & Destructor Documentation

4.21.1.1 MHA_AC::acspace2matrix_t::acspace2matrix_t (algo_comm_t *ac*, const std::vector< std::string > & *names*)

Scan all given AC variables and allocate corresponding matrixes.

Parameters

ac	AC handle.
names	Names of AC variables, or empty for all.

4.21.1.2 MHA_AC::acspace2matrix_t::acspace2matrix_t (const MHA_AC::acspace2matrix_t & *src*)

Parameters

src	Instance to be copied.

4.21.2 Member Function Documentation

4.21.2.1 MHA_AC::acspace2matrix_t & MHA_AC::acspace2matrix_t::operator= (const MHA_AC::acspace2matrix_t & *src*)

Parameters

src	Array of matrixes to be copied.

4.21.2.2 MHA_AC::ac2matrix_t& MHA_AC::acspace2matrix_t::operator[] (unsigned int *k*) [inline]

Parameters

k	index into array; should not exceed **size()** (p. 116)-1.

Return values

Reference	to matrix.

4.21.2.3 const MHA_AC::ac2matrix_t& MHA_AC::acspace2matrix_t::operator[] (unsigned int *k*) const [inline]

Parameters

k	index into array; should not exceed **size()** (p. 116)-1.

Return values

Constant	reference to matrix.

4.21.2.4 void MHA_AC::acspace2matrix_t::update ( ) [inline]

This function updates all matrixes from their corresponding AC variables. It can be called from the MHA Framework prepare function or in the processing callback.

4.21.2.5 void MHA_AC::acspace2matrix_t::insert (algo_comm_t *ac*)

Parameters

ac	AC space handle to insert data

4.22 MHA_AC::double_t Class Reference

Insert a double precision floating point variable into the AC space.

Public Attributes

- double **data**
  Floating point value variable.

4.22.1 Detailed Description

The variable is automatically removed on destruction.

4.23 MHA_AC::float_t Class Reference

Insert a float point variable into the AC space.

Public Member Functions

- **float_t** (**algo_comm_t**, std::string, float=0)
  Constructor.

Public Attributes

- float **data**

  Floating point value variable.

4.23.1 Detailed Description

The variable is automatically removed on destruction.

4.24 MHA_AC::int_t Class Reference

Insert a integer variable into the AC space.

Public Attributes

- int **data**

  Integer value variable.

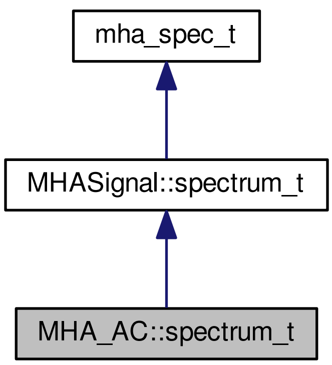
4.24.1 Detailed Description

The variable is automatically removed on destruction.

4.25 MHA_AC::spectrum_t Class Reference

Insert a **MHASignal::spectrum_t** (p. 261) class into the AC space.

Inheritance diagram for MHA_AC::spectrum_t:



Public Member Functions

- **spectrum_t** (**algo_comm_t** ac, std::string name, unsigned int bins, unsigned int **channels**, bool insert_now)

  Create the AC variable.
- void **insert** ()

  Insert AC variable into AC space.

Additional Inherited Members

4.25.1 Detailed Description

The variable is automatically removed on destruction.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 MHA_AC::spectrum_t::spectrum_t (algo_comm_t *ac,* std::string *name,* unsigned int *bins,* unsigned int *channels,* bool *insert_now*)

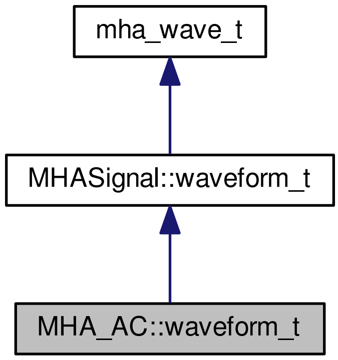
Parameters

ac	AC handle
name	Name of variable in AC space
bins	Number of FFT bins in the **waveform_t** (p. 120) class
channels	Number of audio channels in the **waveform_t** (p. 120) class
insert_now	Insert implicitely in the constructor (true) or explicitely in the **insert()** (p. 120) function (false)

4.26 MHA_AC::waveform_t Class Reference

Insert a **MHASignal::waveform_t** (p. 268) class into the AC space.

Inheritance diagram for MHA_AC::waveform_t:



Public Member Functions

- **waveform_t** (**algo_comm_t** ac, std::string name, unsigned int frames, unsigned int **channels**, bool insert_now)

    Create the AC variable.
- void **insert** ()

    Insert AC variable into AC space.

Additional Inherited Members

4.26.1 Detailed Description

The variable is automatically removed on destruction.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 MHA_AC::waveform_t::waveform_t (**algo_comm_t** *ac,* std::string *name,* unsigned int *frames,* unsigned int *channels,* bool *insert_now*)

Parameters

ac	AC handle
name	Name of variable in AC space
frames	Number of frames in the **waveform_t** (p. 120) class
channels	Number of audio channels in the **waveform_t** (p. 120) class
insert_now	Insert implicitely in the constructor (true) or explicitely in the **insert()** (p. 121) function (false)

4.27 mha_audio_descriptor_t Struct Reference

Description of an audio fragment (planned as a replacement of **mhaconfig_t** (p. 155)).

Public Attributes

- unsigned int **n_samples**

  Number of samples.
- unsigned int **n_channels**

  Number of audio channels.
- unsigned int **n_freqs**

  Number of frequency bands.
- unsigned int **is_complex**

  Flag about sample type.
- **mha_real_t dt**

  Time distance between samples (only equidistant samples allowed)
- **mha_real_t** $*$ **cf**

  Center frequencies of frequency bands.
- **mha_real_t** $*$ **chdir**

  Hint on source direction of channel, values below zero is left, values above zero is right, zero means unknown.

4.28 mha_audio_t Struct Reference

An audio fragment in the openMHA (planned as a replacement of **mha_wave_t** (p. 154) and **mha_spec_t** (p. 141)).

Public Attributes

- **mha_audio_descriptor_t descriptor**

  Dimension and description of the data.
- **mha_real_t** $*$ **rdata**

  *Data pointer if flag **mha_audio_descriptor_t::is_complex** (p. 122) is unset.*
- **mha_complex_t** $*$ **cdata**

  *Data pointer if flag **mha_audio_descriptor_t::is_complex** (p. 122) is set.*

4.28.1 Detailed Description

The data alignment is $(t_0, c_0, f_0), (t_0, c_0, f_1), \ldots, (t_0, c_0, f_{freqs}), (t_0, c_1, f_0), \ldots$. This allows a direct cast of the current **mha_wave_t** (p. 154) and **mha_spec_t** (p. 141) data pointers into corresponding **mha_audio_t** (p. 122) objects.

4.28.2 Member Data Documentation

4.28.2.1 mha_audio_descriptor_t mha_audio_t::descriptor

4.28.2.2 mha_real_t* mha_audio_t::rdata

4.28.2.3 mha_complex_t* mha_audio_t::cdata

4.29 mha_channel_info_t Struct Reference

Channel information structure.

Public Attributes

- int **id**
  channel id
- char **idstr** [32]
  channel id
- unsigned int **side**
  side (left/right)
- **mha_direction_t dir**
  source direction
- **mha_real_t peaklevel**
  Peak level corresponds to this SPL (dB) level.

4.30 mha_complex_t Struct Reference

Type for complex floating point values.

Public Attributes

- **mha_real_t re**
  Real part.
- **mha_real_t im**
  Imaginary part.

4.30.1 Member Data Documentation

4.30.1.1 mha_real_t mha_complex_t::re

4.30.1.2 mha_real_t mha_complex_t::im

4.31 mha_dblbuf_t< FIFO > Class Template Reference

The doublebuffer adapts blocksizes between an outer process, which provides input data and takes output data, and an inner process, which processes the input signal and generates output data using a different block size than the outer process.

Public Types

- typedef FIFO::value_type **value_type**

  The datatype exchanged by the FIFO and this doublebuffer.

Public Member Functions

- **mha_dblbuf_t** (unsigned **outer_size**, unsigned **inner_size**, unsigned **delay**, unsigned **input_channels**, unsigned **output_channels**, const **value_type** &delay_data)

  Constructor creates FIFOs with specified delay.
- virtual void **process** (const **value_type** ∗input_signal, **value_type** ∗output_signal, unsigned count)

  The outer process has to call this method to propagate the input signal to the inner process, and receives back the output signal.
- virtual void **input** (**value_type** ∗input_signal)

  The inner process has to call this method to receive its input signal.
- virtual void **output** (const **value_type** ∗output_signal)

  The outer process has to call this method to deliver its output signal.

Private Attributes

- unsigned **outer_size**

  The block size used by the outer process.
- unsigned **inner_size**

  The block size used by the inner process.
- unsigned **delay**

  The delay introduced by bidirectional buffer size adaptation.
- unsigned **fifo_size**

  The size of each of the FIFOs.
- unsigned **input_channels**

  The number of input channels.

- unsigned **output_channels**
  The number of output channels.
- FIFO **input_fifo**
  The FIFO for transporting the input signal from the outer process to the inner process.
- FIFO **output_fifo**
  The FIFO for transporting the output signal from the inner process to the outer process.
- **MHA_Error** ∗ **inner_error**
  Owned copy of exception to be thrown in inner thread.
- **MHA_Error** ∗ **outer_error**
  Owned copy of exception to be thrown in outer thread.

4.31.1 Detailed Description

template<class FIFO>
class mha_dblbuf_t< FIFO >

This class introduces the channels concept. Input and output may have different channel counts.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 template<class FIFO > mha_dblbuf_t< FIFO >::mha_dblbuf_t (unsigned *outer_size,* unsigned *inner_size,* unsigned *delay,* unsigned *input_channels,* unsigned *output_channels,* const value_type & *delay_data*)

Warning

The doublebuffer may block or raise an exception if the delay is too small. To avoid this, the delay should be

$$delay >= (inner\_size - gcd(inner\_size, outer\_size))$$

.

Parameters

outer_size	The block size used by the outer process.
inner_size	The block size used by the inner process.
delay	The total delay
input_channels	Number of input channels
output_channels	Number of output channels
delay_data	The delay consists of copies of this value.

4.31.3 Member Function Documentation

4.31.3.1 template< class FIFO > void mha_dblbuf_t< FIFO >::process (const value_type * *input_signal*, value_type * *output_signal*, unsigned *count*) [virtual]

Parameters

input_signal	Pointer to the input signal array.
output_signal	Pointer to the output signal array.
count	The number of data instances provided and expected, lower or equal to inner_size given to constructor.

Exceptions

MHA_Error (p. 132)	When count is > outer_size as given to constructor or the underlying fifo implementation detects an error.

4.31.3.2 template< class FIFO > void mha_dblbuf_t< FIFO >::input (value_type * *input_signal*) [virtual]

Parameters

input_signal	Array where the doublebuffer can store the signal.

Exceptions

MHA_Error (p. 132)	When the underlying fifo implementation detects an error.

4.31.3.3 template< class FIFO > void mha_dblbuf_t< FIFO >::output (const value_type * *output_signal*) [virtual]

Parameters

output_signal	Array from which doublebuffer reads outputsignal.

Exceptions

MHA_Error (p. 132)	When the underlying fifo implementation detects an error.

4.31.4 Member Data Documentation

4.31.4.1 template< class FIFO > unsigned mha_dblbuf_t< FIFO >::outer_size [private]

4.31.4.2 template< class FIFO > unsigned mha_dblbuf_t< FIFO >::inner_size [private]

4.31.4.3 template< class FIFO > unsigned mha_dblbuf_t< FIFO >::delay [private]

4.31.4.4 template< class FIFO > FIFO mha_dblbuf_t< FIFO >::input_fifo [private]

4.31.4.5 template< class FIFO > FIFO mha_dblbuf_t< FIFO >::output_fifo [private]

4.32 mha_direction_t Struct Reference

Channel source direction structure.

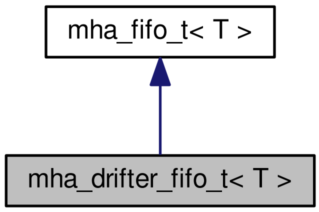
Public Attributes

- **mha_real_t azimuth**
  azimuth in radiants
- **mha_real_t elevation**
  elevation in radiants
- **mha_real_t distance**
  distance in meters

4.33 mha_drifter_fifo_t< T > Class Template Reference

A FIFO class for blocksize adaptation without Synchronization.

Inheritance diagram for mha_drifter_fifo_t< T >:



Public Member Functions

- virtual void **write** (const T ∗data, unsigned count)

  write data to fifo
- virtual void **read** (T ∗**buf**, unsigned count)

  Read data from fifo.
- virtual unsigned **get_fill_count** () const

  Return fill_count, adding ***mha_drifter_fifo_t<T>::startup_zeros*** *(p. 131) to the number of samples actually in the fifo's buffer.*
- virtual unsigned **get_available_space** () const

  Return available space, subtracting number of ***mha_drifter_fifo_t<T>::startup_zeros*** *(p. 131) from the available_space actually present in the fifo's buffer.*
- virtual unsigned **get_des_fill_count** () const

  The desired fill count of this fifo.
- virtual unsigned **get_min_fill_count** () const

  The minimum fill count of this fifo.
- virtual void **stop** ()

  Called by ***mha_drifter_fifo_t<T>::read*** *(p. 130) or* ***mha_drifter_fifo_t<T>::write*** *(p. 129) when their xrun in succession counter exceeds its limit.*
- virtual void **starting** ()

  Called by ***mha_drifter_fifo_t<T>::read*** *(p. 130) or* ***mha_drifter_fifo_t<T>::write*** *(p. 129) when the respective flag (****mha_drifter_fifo_t<T>::reader_started*** *(p. 131) or* ***mha_drifter↩_fifo_t<T>::writer_started*** *(p. 131)) is about to be toggled from false to true.*
- **mha_drifter_fifo_t** (unsigned min_fill_count, unsigned **desired_fill_count**, unsigned **max_fill_count**)

  Create drifter FIFO.
- **mha_drifter_fifo_t** (unsigned min_fill_count, unsigned **desired_fill_count**, unsigned **max_fill_count**, const T &t)

  Create drifter FIFO where all (initially unused) copies of T are initialized as copies of t.

Private Attributes

- const unsigned **minimum_fill_count**

  The minimum fill count of this fifo.
- const unsigned **desired_fill_count**

  The desired fill count of the fifo.
- bool **writer_started**

  Flag set to true when write is called the first time.
- bool **reader_started**

  Flag set to true when read is called for the first time.
- unsigned **writer_xruns_total**

  The number of xruns seen by the writer since object instantiation.
- unsigned **reader_xruns_total**

  The number of xruns seen by the reader since object instantiation.
- unsigned **writer_xruns_since_start**

  The number of xruns seen by the writer since the last start of processing.

- unsigned **reader_xruns_since_start**

  The number of xruns seen by the reader since the last start of processing.
- unsigned **writer_xruns_in_succession**

  The number of xruns seen by the writer in succession.
- unsigned **reader_xruns_in_succession**

  The number of xruns seen by the reader in succession.
- unsigned **maximum_writer_xruns_in_succession_before_stop**

  A limit to the number of xruns seen in succession during write before the data transmission through the FIFO is stopped.
- unsigned **maximum_reader_xruns_in_succession_before_stop**

  A limit to the number of xruns seen in succession during read before the data transmission through the FIFO is stopped.
- **mha_fifo_t**< T >::**value_type null_data**

  The value used in place of missing data.
- unsigned **startup_zeros**

  *When processing starts, that is when both **mha_drifter_fifo_t<T>::reader_started** (p. 131) and **mha_drifter_fifo_t<T>::writer_started** (p. 131) are true, then first **mha_drifter_fifo_t<T>::desired_fill_count** (p. 131) instances of **mha_drifter_fifo_t<T>::null_data** (p. 129) are delivered to the reader.*

Additional Inherited Members

4.33.1 Detailed Description

template<class T>
class mha_drifter_fifo_t< T >

Features: delay concept (desired, minimum and maximum delay), drifting support by throwing away data or inserting zeroes.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 template<class T > mha_drifter_fifo_t< T >::mha_drifter_fifo_t (unsigned *min_fill_count*, unsigned *desired_fill_count*, unsigned *max_fill_count*)

4.33.3 Member Function Documentation

4.33.3.1 template<class T > void mha_drifter_fifo_t< T >::write (const T ∗ *data*, unsigned *count*) [virtual]

Sets **writer_started** (p. 131) to true.

When processing has started, i.e. both **reader_started** (p. 131) and **writer_started** (p. 131) are true, write specified ammount of data to the fifo. If there is not enough space available, then the exceeding data is lost and the writer xrun counters are increased.

Processing is stopped when **writer_xruns_in_succession** (p. 131) exceeds **maximum_writer_xruns_in_succession_before_stop** (p. 131).

Parameters

data	Pointer to source data.
count	Number of instances to copy

Reimplemented from **mha_fifo_t**< **T** > (p. 136).

4.33.3.2 template<class T > void mha_drifter_fifo_t< T >::read (T * *buf*, unsigned *count*) [virtual]

Sets **reader_started** (p. 131) to true.

When processing has started, i.e. both **reader_started** (p. 131) and **writer_started** (p. 131) are true, then read specified ammount of data from the fifo. As long as **startup_zeros** (p. 131) is > 0, **null_data** (p. 129) is delivered to the reader and **startup_zeros** (p. 131) is diminished. Only when **startup_zeros** (p. 131) has reached 0, data is actually read from the fifo's buffer.

If the read would cause the fifo's fill count to drop below **minimum_fill_count** (p. 131), then only so much data are read that **minimum_fill_count** (p. 131) entries remain in the fifo, the missing data is replaced with **null_data** (p. 129), and the reader xrun counters are increased.

Processing is stopped when **reader_xruns_in_succession** (p. 131) exceeds **maximum_↩ reader_xruns_in_succession_before_stop** (p. 131).

Parameters

buf	Pointer to the target buffer
count	Number of instances to copy

Reimplemented from **mha_fifo_t**< **T** > (p. 137).

4.33.3.3 template<class T > unsigned mha_drifter_fifo_t< T >::get_fill_count ( ) const [virtual]

Reimplemented from **mha_fifo_t**< **T** > (p. 135).

4.33.3.4 template<class T > unsigned mha_drifter_fifo_t< T >::get_available_space ( ) const [virtual]

TODO: uncertain if this is a good idea.

Reimplemented from **mha_fifo_t**< **T** > (p. 135).

4.33.3.5 template<class T > void mha_drifter_fifo_t< T >::stop ( ) [virtual]

Called by **read** (p. 130) or **write** (p. 129) when their xrun in succession counter exceeds its limit.

May also be called explicitly.

4.33.3.6 template<class T > void mha_drifter_fifo_t< T >::starting ( ) [virtual]

The fifo's buffer is emptied, this method resets **startup_zeros** (p. 131) to **desired_fill_count** (p. 131), and it also resets **reader_xruns_since_start** (p. 129) and **writer_xruns_since_start** (p. 131) to 0.

4.33.4 Member Data Documentation

4.33.4.1 template<class T > const unsigned mha_drifter_fifo_t< T >::minimum_fill_count [private]

4.33.4.2 template<class T > const unsigned mha_drifter_fifo_t< T >::desired_fill_count [private]

The fifo is initialized with this ammount of data when data transmission starts.

4.33.4.3 template<class T > bool mha_drifter_fifo_t< T >::writer_started [private]

4.33.4.4 template<class T > bool mha_drifter_fifo_t< T >::reader_started [private]

4.33.4.5 template<class T > unsigned mha_drifter_fifo_t< T >::writer_xruns_since_start [private]

4.33.4.6 template<class T > unsigned mha_drifter_fifo_t< T >::writer_xruns_in_succession [private]

Reset to 0 every time a write succeeds without xrun.

4.33.4.7 template<class T > unsigned mha_drifter_fifo_t< T >::reader_xruns_in_succession [private]

Reset to 0 every time a read succeeds without xrun.

4.33.4.8 template<class T > unsigned mha_drifter_fifo_t< T >::maximum_writer_xruns_in_↩ succession_before_stop [private]

4.33.4.9 template<class T > unsigned mha_drifter_fifo_t< T >::maximum_reader_xruns_in_↩ succession_before_stop [private]

4.33.4.10 template<class T > unsigned mha_drifter_fifo_t< T >::startup_zeros [private]

These **null_data** (p. 129) instances are not transmitted through the fifo because filling the fifo with enough **null_data** (p. 129) might not be realtime safe and this filling has to be initiated by **starting** (p. 131) or **stop** (p. 130) (this implementation: **starting** (p. 131)) which are be called with realtime constraints.

4.34 MHA_Error Class Reference

Error reporting exception class.

Inherits exception.

Public Member Functions

- **MHA_Error** (const char ∗file, int line, const char ∗fmt,...)

    Create an instance of a ***MHA_Error*** *(p. 132).*
- const char ∗ **get_msg** () const

    Return the error message without source position.
- const char ∗ **get_longmsg** () const

    Return the error message with source position.
- const char ∗ **what** () const throw ()

    overwrite std::execption::what()

4.34.1 Detailed Description

This class is used for error handling in the openMHA. It is used by the openMHA kernel and by the openMHA toolbox library. Please note that exceptions should not be used accross ANSI-C interfaces. It is necessary to catch exceptions within the library.

The **MHA_Error** (p. 132) class holds source file name, line number and an error message.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 MHA_Error::MHA_Error (const char ∗ *s_file,* int *l,* const char ∗ *fmt,* ...)

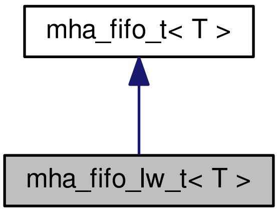
Parameters

s_file	source file name (`FILE`)
l	source line (`LINE`)
fmt	format string for error message (as in printf)

4.35 mha_fifo_lw_t< T > Class Template Reference

This FIFO uses locks to synchronize access.

Inheritance diagram for mha_fifo_lw_t< T >:



Public Member Functions

- virtual void **write** (const T ∗data, unsigned count)
  write specified ammount of data to the fifo.
- virtual void **read** (T ∗**buf**, unsigned count)
  read data from fifo.
- **mha_fifo_lw_t** (unsigned **max_fill_count**)
  Create FIFO with fixed buffer size.
- virtual ∼**mha_fifo_lw_t** ()
  release synchronization object
- virtual void **set_error** (unsigned index, **MHA_Error** ∗**error**)
  Process waiting for more data or space should bail out, throwing this error.

Private Attributes

- **mha_fifo_thread_platform_t** ∗ **sync**
  platform specific thread synchronization
- **MHA_Error** ∗ **error** [2]
  If waiting for synchronization should be aborted then exception to be thrown by reader process (index 0) or writer process (index 1) has to be placed here.

Additional Inherited Members

4.35.1 Detailed Description

template<class T>
class mha_fifo_lw_t< T >

Reading and writing can block until the operation can be executed.

4.35.2 Member Function Documentation

4.35.2.1 template< class T > void mha_fifo_lw_t< T >::write (const T * *data*, unsigned *count*) [virtual]

If there is not enough space, then wait for more space.

Parameters

data	Pointer to source data.
count	Number of instances to copy.

Exceptions

MHA_Error *(p. 132)*	when detecting a deadlock situation.

Reimplemented from **mha_fifo_t< T >** (p. 136).

4.35.2.2 template< class T > void mha_fifo_lw_t< T >::read (T * *buf*, unsigned *count*) [virtual]

If there is not enough data, then wait for more data.

Parameters

buf	Pointer to the target buffer.
count	Number of instances to copy.

Exceptions

MHA_Error *(p. 132)*	when detecting a deadlock situation.

Reimplemented from **mha_fifo_t< T >** (p. 137).

4.35.2.3 template< class T > void mha_fifo_lw_t< T >::set_error (unsigned *index*, MHA_Error * *error*) [virtual]

Parameters

index	Use 0 for terminating reader, 1 for terminating writer.
error	**MHA_Error** (p. 132) to be thrown

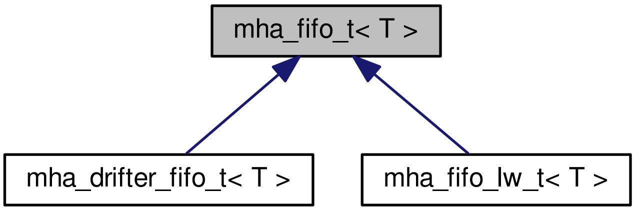
4.35.3 Member Data Documentation

4.35.3.1 template<class T > MHA_Error∗ mha_fifo_lw_t< T >::error[2] [private]

4.36 mha_fifo_t< T > Class Template Reference

A FIFO class for blocksize adaptation Synchronization: None.

Inheritance diagram for mha_fifo_t< T >:



Public Types

- typedef T **value_type**
  The data type exchanged by this fifo.

Public Member Functions

- virtual void **write** (const T ∗data, unsigned count)
  write specified ammount of data to the fifo.
- virtual void **read** (T ∗**buf**, unsigned count)
  read data from fifo
- virtual unsigned **get_fill_count** () const
  Read-only access to fill_count.
- virtual unsigned **get_available_space** () const
  Read-only access to available_space.
- virtual unsigned **get_max_fill_count** () const
  The capacity of this fifo.
- **mha_fifo_t** (unsigned **max_fill_count**)
  Create FIFO with fixed buffer size.
- **mha_fifo_t** (unsigned **max_fill_count**, const T &t)
  Create FIFO with fixed buffer size, where all (initially unused) copies of T are initialized as copies of t.
- **mha_fifo_t** (const **mha_fifo_t** &src)
  Copy constructor.
- virtual ∼**mha_fifo_t** ()
  Destroy FIFO.
- **mha_fifo_t**< T > & **operator=** (const **mha_fifo_t**< T > &)
  Assignment operator.

Protected Member Functions

- void **clear** ()
  Empty the fifo at once.

Private Attributes

- const unsigned **max_fill_count**
  The maximum fill count of this FIFO.
- T ∗ **buf**
  The memory allocated to store the data.
- T ∗ **write_ptr**
  points to location where to write next
- const T ∗ **read_ptr**
  points to location where to read next
- bool **buf_uses_placement_new**
  wether buf was allocated using placement new or array new.

4.36.1 Detailed Description

template<class T>
class mha_fifo_t< T >

Use external synchronisation or synchronization in inheriting class.

4.36.2 Member Function Documentation

4.36.2.1 template<class T > void mha_fifo_t< T >::write (const T ∗ *data,* unsigned *count*) [virtual]

Parameters

data	Pointer to source data.
count	Number of instances to copy

Exceptions

MHA_Error *(p. 132)*	when there is not enough space available.

Reimplemented in **mha_fifo_lw_t< T >** (p. 134), and **mha_drifter_fifo_t< T >** (p. 129).

4.36.2.2 template< class T > void mha_fifo_t< T >::read (T * *buf*, unsigned *count*) [virtual]

Parameters

buf	Pointer to the target buffer
count	Number of instances to copy

Exceptions

MHA_Error *(p. 132)*	when there is not enough data available.

Reimplemented in **mha_fifo_lw_t< T >** (p. 134), and **mha_drifter_fifo_t< T >** (p. 130).

4.36.2.3 template< class T > void mha_fifo_t< T >::clear ( ) [inline], [protected]

Should be called by the reader, or when the reader is inactive.

4.36.3 Member Data Documentation

4.36.3.1 template< class T > const unsigned mha_fifo_t< T >::max_fill_count [private]

4.36.3.2 template< class T > T* mha_fifo_t< T >::buf [private]

max_fill_count + 1 locations are allocated: At least one location is always unused, because we have max_fill_count + 1 possible fillcounts [0:max_fill_count] that we need to distinguish.

4.36.3.3 template< class T > bool mha_fifo_t< T >::buf_uses_placement_new [private]

4.37 mha_fifo_thread_guard_t Class Reference

Simple Mutex Guard Class.

4.38 mha_fifo_thread_platform_t Class Reference

Abstract base class for synchronizing multithreaded (producer/consumer) fifo operations.

Inherited by mha_fifo_posix_threads_t.

Public Member Functions

- virtual void **aquire_mutex** ()=0
  Calling thread waits until it aquires the lock.
- virtual void **release_mutex** ()=0
  Calling thread releases the lock.
- virtual void **wait_for_decrease** ()=0
  Calling producer thread must own the lock.
- virtual void **wait_for_increase** ()=0
  Calling consumer thread must own the lock.
- virtual void **increment** ()=0
  To be called by producer thread after producing.
- virtual void **decrement** ()=0
  To be called by consumer thread after consuming.
- virtual ∼**mha_fifo_thread_platform_t** ()
  Make destructor virtual.
- **mha_fifo_thread_platform_t** ()
  Make default constructor accessible.

4.38.1 Detailed Description

Works only with single producer and single consumer.

4.38.2 Member Function Documentation

4.38.2.1 virtual void mha_fifo_thread_platform_t::aquire_mutex ( ) `[pure virtual]`

Must not be called when the lock is already aquired.

4.38.2.2 virtual void mha_fifo_thread_platform_t::release_mutex ( ) `[pure virtual]`

May only be called when lock is owned.

4.38.2.3 virtual void mha_fifo_thread_platform_t::wait_for_decrease ( ) `[pure virtual]`

Method releases lock, and waits for consumer thread to call decrease(). Then reaquires lock and returns

4.38.2.4 virtual void mha_fifo_thread_platform_t::wait_for_increase ( ) `[pure virtual]`

Method releases lock, and waits for producer thread to call increase(). Then reaquires lock and returns

4.38.2.5 virtual void mha_fifo_thread_platform_t::increment ( ) [pure virtual]

Producer thread needs to own the lock to call this method.

4.38.2.6 virtual void mha_fifo_thread_platform_t::decrement ( ) [pure virtual]

Consumer thread needs to own the lock to call this method.

4.39 mha_rt_fifo_element_t< T > Class Template Reference

Object wrapper for **mha_rt_fifo_t** (p. 140).

Public Member Functions

- **mha_rt_fifo_element_t** (T ∗**data**)
  Constructor.

Public Attributes

- **mha_rt_fifo_element_t**< T > ∗ **next**
  Pointer to next fifo element. NULL for the last (newest) fifo element.
- bool **abandonned**
  Indicates that this element will no longer be used and may be deleted.
- T ∗ **data**
  Pointer to user data.

4.39.1 Constructor & Destructor Documentation

4.39.1.1 template<class T > mha_rt_fifo_element_t< T >::mha_rt_fifo_element_t (T ∗ *data*) [inline]

This element assumes ownership of user data.

Parameters

data	User data. Has to be allocated on the heap with standard operator new, because it will be deleted in this element's destructor.

4.40 mha_rt_fifo_t< T > Class Template Reference

Template class for thread safe, half real time safe fifo without explixit locks.

Public Member Functions

- **mha_rt_fifo_t** ()
  Construct empty fifo.
- ∼**mha_rt_fifo_t** ()
  Destructor will delete all data currently in the fifo.
- T ∗ **poll** ()
  Retrieve the latest element in the Fifo.
- T ∗ **poll_1** ()
  Retrieve the next element in the Fifo, if there is one, and mark the previous element as abandonned.
- void **push** (T ∗data)
  Add element to the Fifo.

Private Member Functions

- void **remove_abandonned** ()
  Deletes abandonned elements.
- void **remove_all** ()
  Deletes all elements.

Private Attributes

- **mha_rt_fifo_element_t**< T > ∗ **root**
  The first element in the fifo. Deleting elements starts here.
- **mha_rt_fifo_element_t**< T > ∗ **current**
  *The element most recently returned by **poll** (p. 141) or **poll_1** (p. 141).*

4.40.1 Detailed Description

template<class T>
class mha_rt_fifo_t< T >

Reading from this fifo is realtime safe, writing to it is not. This fifo is designed for objects that were constructed on the heap. It assumes ownership of these objects and calls delete on them when they are no longer used. Objects remain inside the Fifo while being used by the reader.

A new fifo element is inserted by using **push** (p. 141). The push operation is not real time safe, it allocates and deallocates memory. The latest element is retrieved by calling **poll** (p. 141). This operation will skip fifo elements if more than one **push** (p. 141) has been occured since the last poll. To avoid skipping, call the **poll_1** (p. 141) operation instead.

4.40.2 Member Function Documentation

4.40.2.1 template<class T > T* mha_rt_fifo_t< T >::poll ( ) [inline]

Will skip fifo elements if more than one element has been added since last poll invocation. Will return the same element as on last call if no elements have been added in the mean time. Marks former elements as abandonned.

Returns

The latest element in this Fifo. Returns NULL if the Fifo is empty.

4.40.2.2 template<class T > T* mha_rt_fifo_t< T >::poll_1 ( ) [inline]

Else, if there is no newer element, returns the same element as on last **poll()** (p. 141) or **poll_1()** (p. 141) invocation.

Returns

The next element in this Fifo, if there is one, or the same as before. Returns NULL if the Fifo is empty.

4.40.2.3 template<class T > void mha_rt_fifo_t< T >::push (T * *data*) [inline]

Deletes abandonned elements in the fifo.

Parameters

data	The new user data to place at the end of the fifo. After this invocation, the fifo is the owner of this object and will delete it when it is no longer used. data must have been allocated on the heap with standard operator new.

4.40.3 Member Data Documentation

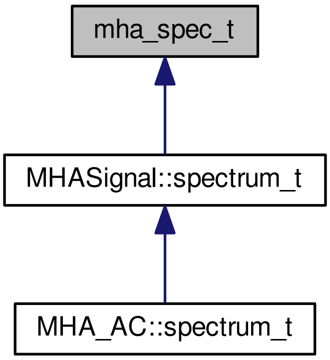
4.40.3.1 template<class T > mha_rt_fifo_element_t<T>* mha_rt_fifo_t< T >::current [private]

Searching for new elements starts here.

4.41 mha_spec_t Struct Reference

Spectrum signal structure.

Inheritance diagram for mha_spec_t:



Public Attributes

- **mha_complex_t** ∗ **buf**

  signal buffer
- unsigned int **num_channels**

  number of channels
- unsigned int **num_frames**

  number of frames in each channel
- **mha_channel_info_t** ∗ **channel_info**

  detailed channel description

4.41.1 Detailed Description

This structure contains the short time fourier transform output of the windowed input signal. The member `num_frames` describes the number of frequency bins in each channel. For an even FFT length N, this is $N/2+1$. With odd FFT lengths, it is $(N+1)/2$. The imaginary part of the first bin is zero. For even FFT lengths, also the imaginary part at the Nyquist frequency is zero.

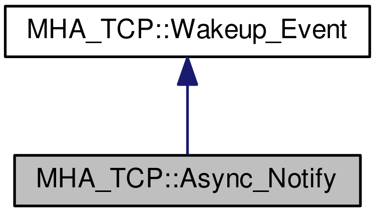
buf[k].re	*Re(0)*	*Re(1)*	*Re(2)*	*Re(3)*	*Re(4)*	...	*Re(n/2−1)*	*Re(n/2)*
buf[k].im		*Im(1)*	*Im(2)*	*Im(3)*	*Im(4)*	...	*Im(n/2−1)*	
k	*0*	*1*	*2*	*3*	*4*		*n/2−1*	*n/2*

Figure 4 Data order of FFT spectrum.

4.42 MHA_TCP::Async_Notify Class Reference

Portable Multiplexable cross-thread notification.

Inheritance diagram for MHA_TCP::Async_Notify:

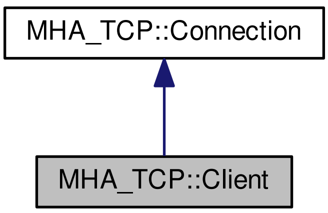


Additional Inherited Members

4.43 MHA_TCP::Client Class Reference

A portable class for a tcp client connections.

Inheritance diagram for MHA_TCP::Client:



Public Member Functions

- **Client** (const std::string &host, unsigned short port)

  Constructor connects to host, port via TCP.
- **Client** (const std::string &host, unsigned short port, **Timeout_Watcher** &timeout_↩ watcher)

  Constructor connects to host, port via TCP, using a timeout.

Additional Inherited Members

4.43.1 Constructor & Destructor Documentation

4.43.1.1 Client::Client (const std::string & *host*, unsigned short *port*)

Parameters

host	The hostname of the TCP Server.
port	The port or the TCP Server.

4.43.1.2 Client::Client (const std::string & *host*, unsigned short *port*, Timeout_Watcher & *timeout_watcher*)

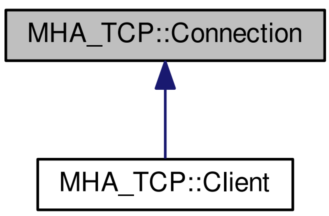
Parameters

host	The hostname of the TCP Server.
port	The port or the TCP Server.
timeout_watcher	an Event watcher that implements a timeout.

4.44 MHA_TCP::Connection Class Reference

Connection (p. 144) handles Communication between client and server, is used on both sides.

Inheritance diagram for MHA_TCP::Connection:



Public Member Functions

- std::string **get_peer_address** ()

  Get peer's IP Address.
- unsigned short **get_peer_port** ()

Get peer's TCP port.

- SOCKET **get_fd** () const

  Return the (protected) file descriptor of the connection.
- virtual ~**Connection** ()

  Destructor closes the underlying file descriptor.
- bool **eof** ()

  Checks if the peer has closed the connection.
- bool **can_read_line** (char delim= '\n')

  Checks if a full line of text has arrived by now.
- bool **can_read_bytes** (unsigned howmany)

  Checks if the specified ammount of data can be read.
- std::string **read_line** (char delim= '\n')

  Reads a single line of data from the socket.
- std::string **read_bytes** (unsigned howmany)

  Reads the specified ammount of dat from the socket.
- void **try_write** (const std::string &data="")

  Adds data to the internal "outgoing" buffer, and then tries to write as much data from that buffer to the socket as possible without blocking.
- void **write** (const std::string &data="")

  Adds data to the internal "outgoing" buffer, and then writes that that buffer to the socket, regardless of blocking.
- bool **needs_write** ()

  Checks if the internal "outgoing" buffer contains data.
- unsigned **buffered_incoming_bytes** () const

  Returns the number of bytes in the internal "incoming" buffer.
- unsigned **buffered_outgoing_bytes** () const

  Returns the number of bytes in the internal "outgoing" buffer.

Protected Member Functions

- **Connection** (SOCKET _fd) throw (MHA_Error)

  Create a connection instance from a socket filedescriptor.

Protected Attributes

- SOCKET **fd**

  The file descriptor of the TCP Socket.

Private Member Functions

- void **init_peer_data** ()
  determine peer address and port
- bool **can_sysread** ()
  Determine wether at least 1 byte can be read without blocking.
- bool **can_syswrite** ()
  Determine wether at least 1 byte can be written without blocking.
- std::string **sysread** (unsigned bytes)
  Call the system's read function and try to read bytes.
- std::string **syswrite** (const std::string &data)
  Call the system's write function and try to write all characters in the string data.

4.44.1 Constructor & Destructor Documentation

4.44.1.1 MHA_TCP::Connection::Connection (SOCKET *_fd*) throw **MHA_Error)** [protected]

Parameters

↩ _↩ *fd*	The file descriptor of the TCP Socket. This file descriptor is closed again in the destructor.

Exceptions

MHA_Error *(p. 132)*	If the file descriptor is < 0.

4.44.2 Member Function Documentation

4.44.2.1 std::string Connection::sysread (unsigned *bytes*) [private]

This will block in a situation where can_sysread returns false.

Parameters

bytes	The desired number of characters.

Returns

The characters read from the socket. The result may have fewer characters than specified by bytes. If the result is an empty string, then the socket has been closed by the peer.

4.44.2.2 std::string Connection::syswrite (const std::string & *data*) [private]

May write fewer characters, but will at least write one character.

Parameters

data	A string of characters to write to the socket.

Returns

The rest of the characters that have not yet been written.

4.44.2.3 SOCKET MHA_TCP::Connection::get_fd ( ) const [inline]

Will be required for SSL.

4.44.2.4 bool Connection::eof ( )

As a side effect, this method fills the internal "incoming" buffer if it was empty and the socket is readable and not eof.

4.44.2.5 bool Connection::can_read_line (char *delim* = '\n')

This method reads data from the socket into the internal "incoming" buffer if it can be done without blocking.

Parameters

delim	The line delimiter.

Returns

true if at least one full line of text is present in the internal buffer after this method call, false otherwise.

4.44.2.6 bool Connection::can_read_bytes (unsigned *howmany*)

This method reads data from the socket into an internal "incoming" buffer if it can be done without blocking.

Parameters

howmany	The number of bytes that the caller wants to have checked.

Returns

true if at least the specified ammount of data is present in the internal buffer after this method call, false otherwise

4.44.2.7 std::string Connection::read_line (char *delim* = '\n')

Blocks if necessary.

Parameters

delim	The line delimiter.

Returns

The string of characters in this line, including the trailing delimiter. The delimiter may be missing if the last line before EOF does not have a delimiter.

4.44.2.8 std::string Connection::read_bytes (unsigned *howmany*)

Blocks if necessary.

Parameters

howmany	The number of bytes to read.

Returns

The string of characters read. The string may be shorter if EOF is encountered.

4.44.2.9 void Connection::try_write (const std::string & *data* = "")

Parameters

data	data to send over the socket.

4.44.2.10 void Connection::write (const std::string & *data* = "")

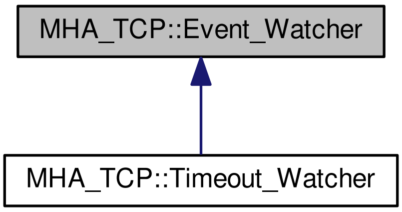
Parameters

data	data to send over the socket.

4.45 MHA_TCP::Event_Watcher Class Reference

OS-independent event watcher, uses select on Unix and WaitForMultipleObjects on Windows.

Inheritance diagram for MHA_TCP::Event_Watcher:



Public Member Functions

- void **observe** (**Wakeup_Event** ∗event)
  Add an event to this observer.
- void **ignore** (**Wakeup_Event** ∗event)
  Remove an event from this observer.
- **std::set**< **Wakeup_Event** ∗ > **wait** ()
  \ Wait for some event to occur.

Private Attributes

- **std::set**< **Wakeup_Event** ∗ > **events**
  The list of events to watch.

4.45.1 Member Function Documentation

4.45.1.1 void Event_Watcher::observe (Wakeup_Event ∗ *event*)

4.45.1.2 std::set< Wakeup_Event ∗ > Event_Watcher::wait ( )

Return all events that are ready

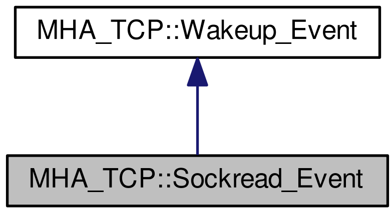
4.45.2 Member Data Documentation

4.45.2.1 std::set<Wakeup_Event∗> MHA_TCP::Event_Watcher::events [private]

4.46 MHA_TCP::Sockread_Event Class Reference

Watch socket for incoming data.

Inheritance diagram for MHA_TCP::Sockread_Event:



Public Member Functions

- **Sockread_Event** (SOCKET s)
  Set socket to watch for.

4.46.1 Constructor & Destructor Documentation

4.46.1.1 MHA_TCP::Sockread_Event::Sockread_Event (SOCKET *s*)

Parameters

s	The socket to observe incoming data on.

4.47 MHA_TCP::Thread Class Reference

A very simple class for portable threads.

Public Types

- typedef void ∗(∗ **thr_f**) (void ∗)
  The thread function signature to use with this class.

Public Member Functions

- **Thread** (**thr_f** func, void ∗**arg**=0)

  Constructor starts a new thread.
- virtual ∼**Thread** ()

  *The destructor should only be called when the **Thread** (p. 150) is finished.*
- virtual void **run** ()

  *The internal method that delegated the new thread to the registered **Thread** (p. 150) function.*

Public Attributes

- **Async_Notify thread_finish_event**

  Event will be triggered when the thread exits.
- enum MHA_TCP::Thread:: { **...** } **state**

  The current state of the thread.
- **thr_f thread_func**

  The thread function that the client has registered.
- void ∗ **thread_arg**

  The argument that the client wants to be handed through to the thread function.
- **MHA_Error** ∗ **error**

  *The **MHA_Error** (p. 132) that caused the thread to abort, if any.*

Protected Member Functions

- **Thread** ()

  Default constructor may only be used by derived classes that want to start the thread themselves.

Protected Attributes

- void ∗ **arg**

  The argument for the client's thread function.
- void ∗ **return_value**

  The return value from the client's thread function is stored here When that function returns.

Private Attributes

- pthread_t **thread_handle**

  The posix thread handle.
- pthread_attr_t **thread_attr**

  The posix thread attribute structure.

4.47.1 Member Typedef Documentation

4.47.1.1 typedef void*(* MHA_TCP::Thread::thr_f) (void *)

Derive from this class and call protected standard constructor to start threads differently.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 Thread::Thread (Thread::thr_f *func*, void * *arg* = 0)

Parameters

func	The function to be executed by the thread.
arg	The argument given to pass to the thread function.

4.47.2.2 Thread::∼Thread ( ) [virtual]

There is preliminary support for forceful thread cancellation in the destructor, but probably not very robust or portable..

4.47.3 Member Data Documentation

4.47.3.1 pthread_attr_t MHA_TCP::Thread::thread_attr [private]

Required for starting a thread in detached state. Detachment is required to eliminate the need for joining this thread.

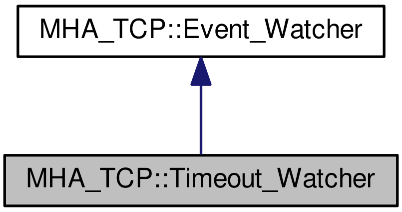
4.47.3.2 void* MHA_TCP::Thread::arg [protected]

4.47.3.3 void* MHA_TCP::Thread::thread_arg

4.48 MHA_TCP::Timeout_Watcher Class Reference

OS-independent event watcher with internal fixed-end-time timeout.

Inheritance diagram for MHA_TCP::Timeout_Watcher:

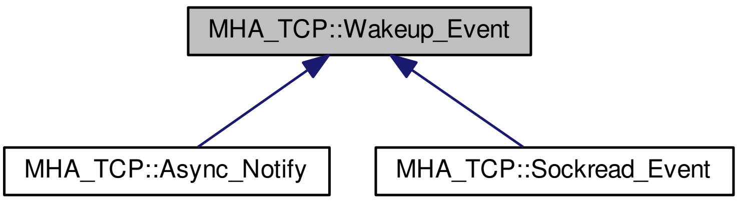


Additional Inherited Members

4.49 MHA_TCP::Wakeup_Event Class Reference

A base class for asynchronous wakeup events.

Inheritance diagram for MHA_TCP::Wakeup_Event:



Public Member Functions

- **Wakeup_Event** ()

  Event Constructor.
- virtual void **observed_by** (**Event_Watcher** ∗observer)

  Called by the ***Event_Watcher*** (p. 149) *when this event is added to its list of observed events.*
- virtual void **ignored_by** (**Event_Watcher** ∗observer)

  Called by the ***Event_Watcher*** (p. 149) *when this event is removed from its list of observed events.*
- virtual ∼**Wakeup_Event** ()

  Destructor deregisters from observers.

- virtual OS_EVENT_TYPE **get_os_event** ()

  Get necessary information for the Event Watcher.
- virtual void **reset** ()

  For pure notification events, reset the "signalled" status.
- virtual bool **status** ()

  Query wether the event is in signalled state now.

Private Attributes

- **std::set**< class **Event_Watcher** ∗ > **observers**

  A list of all ***Event_Watcher*** (p. 149) *instances that this* ***Wakeup_Event*** (p. 153) *is observed by (stored here for proper deregistering).*

4.49.1 Constructor & Destructor Documentation

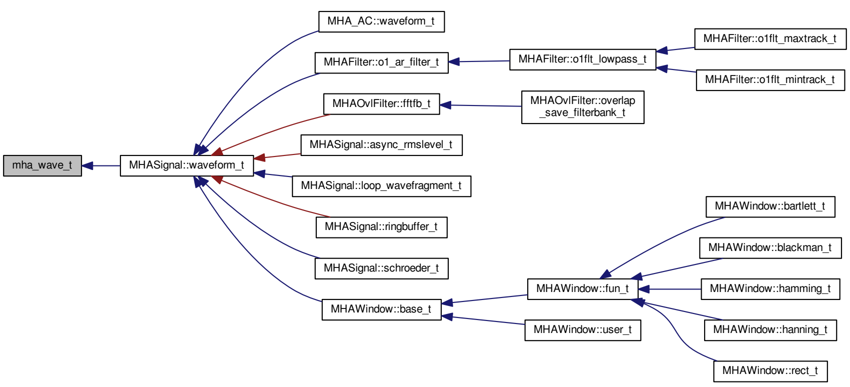
4.49.1.1 Wakeup_Event::Wakeup_Event ( )

The new event has invalid state.

4.50 mha_wave_t Struct Reference

Waveform signal structure.

Inheritance diagram for mha_wave_t:



Public Attributes

- **mha_real_t** ∗ **buf**
  signal buffer
- unsigned int **num_channels**
  number of channels
- unsigned int **num_frames**
  number of frames in each channel
- **mha_channel_info_t** ∗ **channel_info**
  detailed channel description

4.50.1 Detailed Description

This structure contains one fragment of a waveform signal. The member num_frames describes the number of audio samples in each audio channel.

The field channel_info must be an array of num_channels entries or NULL.

4.51 mhaconfig_t Struct Reference

MHA prepare configuration structure.

Public Attributes

- unsigned int **channels**
  Number of audio channels.
- unsigned int **domain**
  Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)
- unsigned int **fragsize**
  Fragment size of waveform data.
- unsigned int **wndlen**
  Window length of spectral data.
- unsigned int **fftlen**
  FFT length of spectral data.
- **mha_real_t** **srate**
  Sampling rate in Hz.

4.51.1 Detailed Description

This structure contains information about channel number and domain for input and output signals of a openMHA Plugin. Each plugin can change any of these parameters, e.g. by resampling of the signal. The only limitation is that the callback frequency is fixed (except for the plugins `db` and `dbasync`).

4.52 MHAEvents::emitter_t Class Reference

Class for emitting openMHA events.

Public Member Functions

- void **operator()** ()
  Emit an event without parameter.
- void **operator()** (const std::string &)
  Emit an event with string parameter.
- void **operator()** (const std::string &, unsigned int, unsigned int)
  Emit an event with string parameter and two unsigned int parameters.

4.52.1 Detailed Description

Use the template claas **MHAEvents::patchbay_t** (p. 156) for connecting to an emitter.

4.53 MHAEvents::patchbay_t< receiver_t > Class Template Reference

Patchbay which connects any event emitter with any member function of the parameter class.

Public Member Functions

- void **connect** (**emitter_t** ∗, receiver_t ∗, void(receiver_t::∗)())
  Connect a receiver member function void (receiver_t::∗)() with an event emitter.
- void **connect** (**emitter_t** ∗, receiver_t ∗, void(receiver_t::∗)(const std::string &))
  Connect a receiver member function void (receiver_t::∗)(const std::string&) with an event emitter.

4.53.1 Detailed Description

template<class receiver_t>
class MHAEvents::patchbay_t< receiver_t >

The connections created by the **connect()** (p. 156) function are hold until the destructor is called. To avoid access to invalid function pointers, it is required to destruct the patchbay before the receiver, usually by declaring the patchbay as a member of the receiver.

The receiver can be any claas or structure; the event callback can be either a member function without arguments or with const std::string& argument.

4.53.2 Member Function Documentation

4.53.2.1 template<class receiver_t> void MHAEvents::patchbay_t< receiver_t >::connect (emitter_t ∗ *e*, receiver_t ∗ *r*, void(receiver_t::∗)() *rfun*)

Create a connection.

The connection is removed when the patchbay is destructed.

Parameters

e	Pointer to an event emitter
r	Pointer to the receiver
rfun	Pointer to a member function of the receiver class

4.53.2.2 template<class receiver_t> void MHAEvents::patchbay_t< receiver_t >::connect (emitter_t ∗ *e*, receiver_t ∗ *r*, void(receiver_t::∗)(const std::string &) *rfun*)

Create a connection.

The connection is removed when the patchbay is destructed.

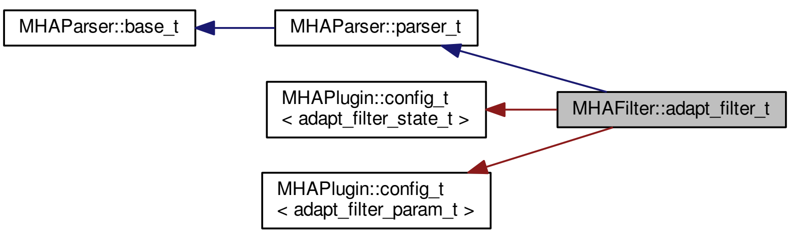
Parameters

e	Pointer to an event emitter
r	Pointer to the receiver
rfun	Pointer to a member function of the receiver class

4.54 MHAFilter::adapt_filter_t Class Reference

Adaptive filter.

Inheritance diagram for MHAFilter::adapt_filter_t:



Additional Inherited Members

4.55 MHAFilter::blockprocessing_polyphase_resampling_t Class Reference

A class that does polyphase resampling and takes into account block processing.

Public Member Functions

- **blockprocessing_polyphase_resampling_t** (float source_srate, unsigned source_↩ fragsize, float target_srate, unsigned target_fragsize, float nyquist_ratio, float irslen, unsigned nchannels, bool add_delay)

  Contructs a polyphase resampling filter that can be used for blockprocessing with the given parameters.
- void **write** (**mha_wave_t** &signal)

  Write signal to the ringbuffer.
- void **read** (**mha_wave_t** &signal)

  Read resampled signal.
- bool **can_read** () const

  Checks if the resampling ring buffer can produce another output signal block.

4.55.1 Detailed Description

4.55.2 Constructor & Destructor Documentation

4.55.2.1 MHAFilter::blockprocessing_polyphase_resampling_t::blockprocessing_polyphase_↩ resampling_t (float *source_srate,* unsigned *source_fragsize,* float *target_srate,* unsigned *target_fragsize,* float *nyquist_ratio,* float *irslen,* unsigned *nchannels,* bool *add_delay*)

Parameters

source_srate	Source sampling rate / Hz
source_fragsize	Fragment size of incoming audio blocks / frames at source_srate
target_srate	Target sampling rate / Hz
target_fragsize	Fragment size of produced audio blocks / frames at target_srate
nyquist_ratio	Low pass filter cutoff frequency relative to the nyquist frequency of the smaller of the two sampling rates. Example values: 0.8, 0.9
irslen	Impulse response length used for low pass filtering / s
nchannels	Number of audio channels
add_delay	To avoid underruns, a delay is generally necessary for round trip block size adaptations. It is only necessary to add this delay to one of the two resampling chains. Set this parameter to true for the first resampling object of a round trip pair. It will add the necessary delay, and calculate the size of the ring buffer appropriately, When set to false, only the ringbuffer size will be set sufficiently.

4.55.3 Member Function Documentation

4.55.3.1 void MHAFilter::blockprocessing_polyphase_resampling_t::write (mha_wave_t & *signal*)

Parameters

signal	input signal in original sampling rate

Exceptions

MHA_Error *(p. 132)*	Raises exception if there is not enough room, if the number of channels does not match, or if the number of frames is not equal to the number specified in the constructor

4.55.3.2 void MHAFilter::blockprocessing_polyphase_resampling_t::read (**mha_wave_t** & *signal*)

Will perform the resampling and remove no longer needed samples from the input buffer.

Parameters

signal	buffer to write the resampled signal to.

Exceptions

MHA_Error *(p. 132)*	Raises exception if there is not enough input signal, if the number of channels of frames does not match.

4.56 MHAFilter::complex_bandpass_t Class Reference

Complex bandpass filter.

Public Member Functions

- **complex_bandpass_t** (std::vector< **mha_complex_t** > A, std::vector< **mha_complex_t** > B)

  Constructor with filter coefficients (one per channel)
- void **set_weights** (std::vector< **mha_complex_t** > new_B)

  Allow to modify the input weights at a later stage.
- void **filter** (const **mha_wave_t** &X, **mha_spec_t** &Y)

  Filter method for real value input.
- void **filter** (const **mha_wave_t** &X, **mha_wave_t** &Yre, **mha_wave_t** &Yim)

  Filter method for real value input.
- void **filter** (const **mha_spec_t** &X, **mha_spec_t** &Y)

  Filter method for complex value input.
- void **filter** (const **mha_wave_t** &Xre, const **mha_wave_t** &Xim, **mha_wave_t** &Yre, **mha_wave_t** &Yim)

  Filter method for complex value input.

4.56.1 Constructor & Destructor Documentation

4.56.1.1 MHAFilter::complex_bandpass_t::complex_bandpass_t (std::vector< mha_complex_t > *A*, std::vector< mha_complex_t > *B*)

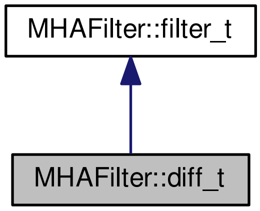
Parameters

A	complex filter coefficients, one per band
B	complex weights

4.57 MHAFilter::diff_t Class Reference

Differentiator class (non-normalized)

Inheritance diagram for MHAFilter::diff_t:

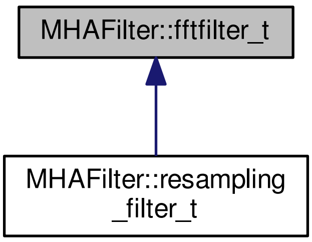


Additional Inherited Members

4.58 MHAFilter::fftfilter_t Class Reference

FFT based FIR filter implementation.

Inheritance diagram for MHAFilter::fftfilter_t:



Public Member Functions

- **fftfilter_t** (unsigned int fragsize, unsigned int channels, unsigned int fftlen)
  Constructor.
- void **update_coeffs** (const **mha_wave_t** ∗pwIRS)
  Update the set of coefficients.
- void **filter** (const **mha_wave_t** ∗pwIn, **mha_wave_t** ∗∗ppwOut, const **mha_wave_t** ∗pw←IRS)
  Apply filter with changing coefficients to a waveform fragment.
- void **filter** (const **mha_wave_t** ∗pwIn, **mha_wave_t** ∗∗ppwOut)
  Apply filter to waveform fragment, without changing the coefficients.
- void **filter** (const **mha_wave_t** ∗pwIn, **mha_wave_t** ∗∗ppwOut, const **mha_spec_t** ∗ps←Weights)
  Apply filter with changing coefficients to a waveform fragment.

4.58.1 Detailed Description

The maximal number of coefficients can be FFT length - fragsize + 1.

4.58.2 Constructor & Destructor Documentation

4.58.2.1 MHAFilter::fftfilter_t::fftfilter_t (unsigned int *fragsize,* unsigned int *channels,* unsigned int *fftlen*)

Parameters

fragsize	Number of frames expected in input signal (each cycle).
channels	Number of channels expected in input signal.
fftlen	FFT length of filter.

4.58.3 Member Function Documentation

4.58.3.1 void MHAFilter::fftfilter_t::update_coeffs (const mha_wave_t ∗ *pwIRS*)

Parameters

pwIRS	Coefficients structure

Note

The number of channels in h must match the number of channels given in the constructor. The filter length is limited to fftlen-fragsize+1 (longer IRS will be shortened).

4.58.3.2 void MHAFilter::fftfilter_t::filter (const mha_wave_t * pwIn, mha_wave_t ** ppwOut, const mha_wave_t * pwIRS)

Parameters

pwIn	Input signal pointer.

Return values

ppwOut	Pointer to output signal pointer, will be set to a valid signal.

Parameters

pwIRS	Pointer to FIR coefficients structure.

4.58.3.3 void MHAFilter::fftfilter_t::filter (const mha_wave_t * pwIn, mha_wave_t ** ppwOut)

Parameters

pwIn	Input signal pointer.

Return values

ppwOut	Pointer to output signal pointer, will be set to a valid signal

4.58.3.4 void MHAFilter::fftfilter_t::filter (const mha_wave_t * pwIn, mha_wave_t ** ppwOut, const mha_spec_t * psWeights)

Parameters

pwIn	Input signal pointer.

Return values

ppwOut	Pointer to output signal pointer, will be set to a valid signal.

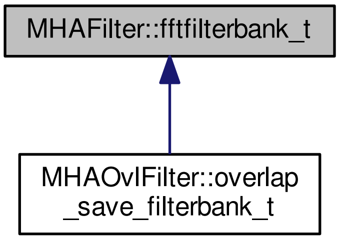
Parameters

psWeights	Pointer to filter weights structure.

4.59 MHAFilter::fftfilterbank_t Class Reference

FFT based FIR filterbank implementation.

Inheritance diagram for MHAFilter::fftfilterbank_t:



Public Member Functions

- **fftfilterbank_t** (unsigned int fragsize, unsigned int inputchannels, unsigned int firchannels, unsigned int fftlen)
    Constructor.
- void **update_coeffs** (const **mha_wave_t** ∗h)
    Update the set of coefficients.
- void **filter** (const **mha_wave_t** ∗s_in, **mha_wave_t** ∗∗s_out, const **mha_wave_t** ∗h)
    Apply filter with changing coefficients to a waveform fragment.
- void **filter** (const **mha_wave_t** ∗s_in, **mha_wave_t** ∗∗s_out)
    Apply filter to waveform fragment, without changing the coefficients.
- const **mha_wave_t** ∗ **get_irs** () const
    Return the current IRS.

4.59.1 Detailed Description

This class convolves n input channels with m filter coefficient sets and returns n∗m output channels.

The maximal number of coefficients can be FFT length - fragsize + 1.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 MHAFilter::fftfilterbank_t::fftfilterbank_t (unsigned int *fragsize,* unsigned int *inputchannels,* unsigned int *firchannels,* unsigned int *fftlen*)

Parameters

fragsize	Number of frames expected in input signal (each cycle).
inputchannels	Number of channels expected in input signal.
firchannels	Number of channels expected in FIR filter coefficients (= number of bands).
fftlen	FFT length of filter.

The number of output channels is inputchannels∗firchannels.

4.59.3 Member Function Documentation

4.59.3.1 void MHAFilter::fftfilterbank_t::update_coeffs (const mha_wave_t ∗ *h*)

Parameters

h	Coefficients structure

Note

The number of channels in h must match the number of channels given in the constructor, and the number of frames can not be more than fftlen-fragsize+1.

4.59.3.2 void MHAFilter::fftfilterbank_t::filter (const mha_wave_t ∗ *s_in,* mha_wave_t ∗∗ *s_out,* const mha_wave_t ∗ *h*)

Parameters

s_in	Input signal pointer.

Return values

s_out	Pointer to output signal pointer, will be set to a valid signal

Parameters

h	FIR coefficients

4.59.3.3 void MHAFilter::fftfilterbank_t::filter (const mha_wave_t ∗ *s_in,* mha_wave_t ∗∗ *s_out*)

Parameters

s_in	Input signal pointer.

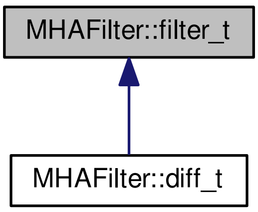
Return values

s_out	Pointer to output signal pointer, will be set to a valid signal

4.60 MHAFilter::filter_t Class Reference

Generic IIR filter class.

Inheritance diagram for MHAFilter::filter_t:



Public Member Functions

- **filter_t** (unsigned int ch, unsigned int lena, unsigned int lenb)

  Constructor.
- **filter_t** (unsigned int ch, const std::vector< **mha_real_t** > &vA, const std::vector< **mha_real_t** > &vB)

  Constructor with initialization of coefficients.
- void **filter** (**mha_wave_t** ∗out, const **mha_wave_t** ∗in)

  Filter all channels in a waveform structure.
- void **filter** (**mha_real_t** ∗dest, const **mha_real_t** ∗src, unsigned int dframes, unsigned int frame_dist, unsigned int channel_dist, unsigned int channel_begin, unsigned int channel_end)

  Filter parts of a waveform structure.
- **mha_real_t filter** (**mha_real_t** x, unsigned int ch)

  Filter one sample.
- unsigned int **get_len_A** () const

  Return length of recursive coefficients.
- unsigned int **get_len_B** () const

  Return length of non-recursive coefficients.

Public Attributes

- double $*$ **A**

  Pointer to recursive coefficients.
- double $*$ **B**

  Pointer to non-recursive coefficients.

4.60.1 Detailed Description

This class implements a generic multichannel IIR filter. It is realized as direct form II. It can work on any float array or on **mha_wave_t** (p. 154) structs. The filter coefficients can be directly accessed.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 MHAFilter::filter_t::filter_t (unsigned int *ch*, unsigned int *lena*, unsigned int *lenb*)

Parameters

ch	Number of channels
lena	Number of recursive coefficients
lenb	Number of non-recursive coefficients

4.60.2.2 MHAFilter::filter_t::filter_t (unsigned int *ch*, const std::vector< **mha_real_t** > & *vA*, const std::vector< **mha_real_t** > & *vB*)

Parameters

ch	Number of channels.
vA	Recursive coefficients.
vB	Non-recursive coefficients.

4.60.3 Member Function Documentation

4.60.3.1 void MHAFilter::filter_t::filter (**mha_wave_t** $*$ *out*, const **mha_wave_t** $*$ *in*)

Parameters

out	Output signal
in	Input signal

4.60.3.2 void MHAFilter::filter_t::filter (**mha_real_t** ∗ *dest,* const **mha_real_t** ∗ *src,* unsigned int *dframes,* unsigned int *frame_dist,* unsigned int *channel_dist,* unsigned int *channel_begin,* unsigned int *channel_end*)

Parameters

dest	Output signal.
src	Input signal.
dframes	Number of frames to be filtered.
frame_dist	Index distance between frames of one channel
channel_dist	Index distance between audio channels
channel_begin	Number of first channel to be processed
channel_end	Number of last channel to be processed

4.60.3.3 **mha_real_t** MHAFilter::filter_t::filter (**mha_real_t** *x,* unsigned int *ch*)

Parameters

x	Input value
ch	Channel number to use in filter state

4.61 MHAFilter::gamma_flt_t Class Reference

Class for gammatone filter.

Public Member Functions

- **gamma_flt_t** (std::vector< **mha_real_t** > cf, std::vector< **mha_real_t** > bw, **mha_real↩_t** srate, unsigned int order)
  Constructor.
- void **operator()** (**mha_wave_t** &X, **mha_spec_t** &Y)
  Filter method.
- void **operator()** (**mha_wave_t** &X, **mha_wave_t** &Yre, **mha_wave_t** &Yim)
  Filter method.
- void **operator()** (**mha_wave_t** &Yre, **mha_wave_t** &Yim, unsigned int stage)
  Filter method for specific stage.

4.61.1 Constructor & Destructor Documentation

4.61.1.1 MHAFilter::gamma_flt_t::gamma_flt_t (std::vector< **mha_real_t** > *cf,* std::vector< **mha_real_t** > *bw,* **mha_real_t** *srate,* unsigned int *order*)

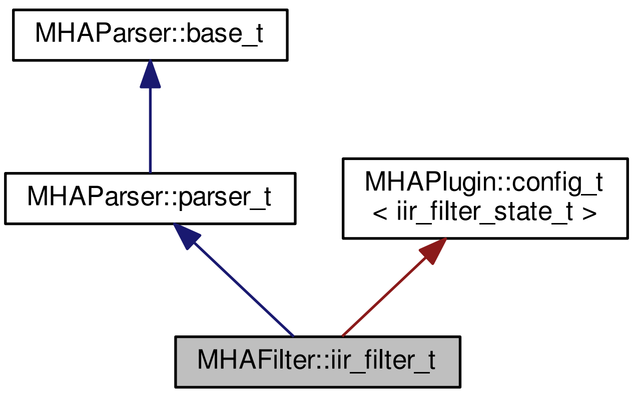
Parameters

cf	Center frequency in Hz.
bw	Bandwidth in Hz (same number of entries as in cf).
srate	Sampling frequency in Hz.
order	Filter order.

4.62 MHAFilter::iir_filter_t Class Reference

IIR filter class wrapper for integration into parser structure.

Inheritance diagram for MHAFilter::iir_filter_t:



Public Member Functions

- **iir_filter_t** (std::string help="IIR **filter** structure", std::string def_A="[1]", std::string def_B="[1]", unsigned int **channels**=1)

  Constructor of the IIR filter.
- void **filter** (**mha_wave_t** ∗y, const **mha_wave_t** ∗x)

  The filter processes the audio signal.
- **mha_real_t filter** (**mha_real_t** x, unsigned int ch)

  Filter a single audio sample.
- void **resize** (unsigned int **channels**)

  Change the number of channels after object creation.

Additional Inherited Members

4.62.1 Detailed Description

This class implements an infinite impulse response filter. Since it inherits from **MHAParser::parser_t** (p. 220), it can easily be integrated in the openMHA configuration tree. It provides the configuration language variables "A" (vector of recursive filter coefficients) and "B" (vector of non-recursive filter coefficients).

The filter instance reacts to changes in filter coefficients through the openMHA configuration language, and uses the updated coefficients in the next invocation of the filter method.

Update of the coefficients is thread-safe and non-blocking. Simply add this subparser to your parser items and use the "filter" member function. Filter states are reset to all 0 on update.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 MHAFilter::iir_filter_t::iir_filter_t (std::string *help* = "IIR filter structure", std::string *def_A* = "[1]", std::string *def_B* = "[1]", unsigned int *channels* = 1)

Initialises the sub-parser structure and the memory for holding the filter's state.

Parameters

help	The help string for the parser that groups the configuration variables of this filter. Could be used to describe the purpose of this IIR filter.
def_A	The initial value of the vector of the recursive filter coefficients, represented as string.
def_B	The initial value of the vector of the non-recursive filter coefficients, represented as string.
channels	The number of indipendent audio channels to process with this filter. Needed to allocate a state vector for each audio channel.

4.62.3 Member Function Documentation

4.62.3.1 void MHAFilter::iir_filter_t::filter (mha_wave_t * *y*, const mha_wave_t * *x*)

All channels in the audio signal are processed using the same filter coefficients. Indipendent state is stored between calls for each audio channel.

Parameters

y	Pointer to output signal holder. The output signal is stored here. Has to have the same signal dimensions as the input signal x. In-place processing (y and x pointing to the same signal holder) is possible.
x	Pointer to input signal holder. Number of channels has to be the same as given to the constructor, or to the **resize** (p. 170) method.

4.62.3.2 mha_real_t MHAFilter::iir_filter_t::filter (mha_real_t *x*, unsigned int *ch*)

Parameters

x	The single audio sample
ch	Zero-based channel index. Use and change the state of channel ch. ch has to be less than the number of channels given to the constructor or the **resize** (p. 170) method.

Returns

the filtered result sample.

4.62.3.3 void MHAFilter::iir_filter_t::resize (unsigned int *channels*)

Parameters

channels	The new number of channels. Old filter states are lost.

4.63 MHAFilter::iir_ord1_real_t Class Reference

First order recursive filter.

Public Member Functions

- **iir_ord1_real_t** (std::vector< **mha_real_t** > A, std::vector< **mha_real_t** > B)
  Constructor with filter coefficients (one per channel)
- **iir_ord1_real_t** (std::vector< **mha_real_t** > tau, **mha_real_t** srate)
  Constructor for low pass filter (one time constant per channel)
- **mha_real_t operator()** (unsigned int ch, **mha_real_t** x)
  Filter method for real value input, one element.
- **mha_complex_t operator()** (unsigned int ch, **mha_complex_t** x)
  Filter method for complex input, one element.
- void **operator()** (const **mha_wave_t** &X, **mha_wave_t** &Y)
  Filter method for real value input.
- void **operator()** (const **mha_spec_t** &X, **mha_spec_t** &Y)

Filter method for complex value input.

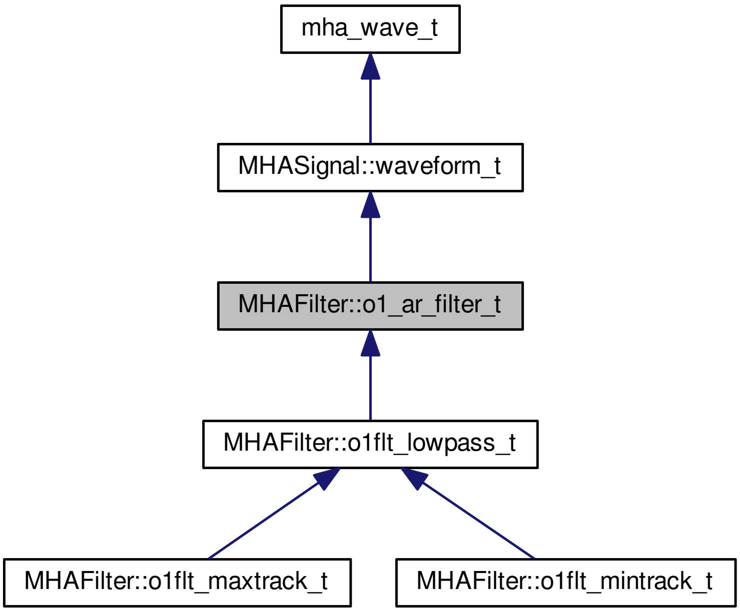
- void **operator()** (const **mha_wave_t** &Xre, const **mha_wave_t** &Xim, **mha_wave_t** &Yre, **mha_wave_t** &Yim)

  Filter method for complex value input.

4.64 MHAFilter::o1_ar_filter_t Class Reference

First order attack-release lowpass filter.

Inheritance diagram for MHAFilter::o1_ar_filter_t:



Public Member Functions

- **o1_ar_filter_t** (unsigned int **channels**, **mha_real_t** fs=1.0f, std::vector< mha_real_t > tau_a=std::vector< float >(1, 0.0f), std::vector< mha_real_t > tau_r=std::vector< float >(1, 0.0f))

  Constructor, setting all taus to zero.
- void **set_tau_attack** (unsigned int ch, **mha_real_t** tau)

  Set the attack time constant.
- void **set_tau_release** (unsigned int ch, **mha_real_t** tau)

  Set the release time constant.
- **mha_real_t operator()** (unsigned int ch, **mha_real_t** x)

  Apply filter to value x, using state channel ch.
- void **operator()** (const **mha_wave_t** &in, **mha_wave_t** &out)

  *Apply filter to a **mha_wave_t** (p. 154) data.*

Additional Inherited Members

4.64.1 Detailed Description

This filter is the base of first order lowpass filter, maximum tracker and minimum tracker.

4.64.2 Constructor & Destructor Documentation

4.64.2.1 MHAFilter::o1_ar_filter_t::o1_ar_filter_t (unsigned int *channels*, mha_real_t *fs* = `1.0f`, std::vector< mha_real_t > *tau_a* = `std::vector<float>(1,0.0f)`, std::vector< mha_real_t > *tau_r* = `std::vector<float>(1,0.0f)`)

The filter state can be accessed through the member functions of **MHASignal::waveform_t** (p. 268).

Parameters

channels	Number of independent filters
fs	Sampling rate (optional, default = 1)
tau_a	Attack time constants (optional, default = 0)
tau_r	Release time constants (optional, default = 0)

4.64.3 Member Function Documentation

4.64.3.1 void MHAFilter::o1_ar_filter_t::set_tau_attack (unsigned int *ch*, mha_real_t *tau*)

Parameters

ch	Channel number
tau	Time constant

4.64.3.2 void MHAFilter::o1_ar_filter_t::set_tau_release (unsigned int *ch*, mha_real_t *tau*)

Parameters

ch	Channel number
tau	Time constant

4.64.3.3 mha_real_t MHAFilter::o1_ar_filter_t::operator() (unsigned int *ch*, mha_real_t *x*) `[inline]`

Parameters

ch	Cannel number
x	Input value

Returns

Output value

4.64.3.4 void MHAFilter::o1_ar_filter_t::operator() (const mha_wave_t & *in*, mha_wave_t & *out*) [inline]

Parameters

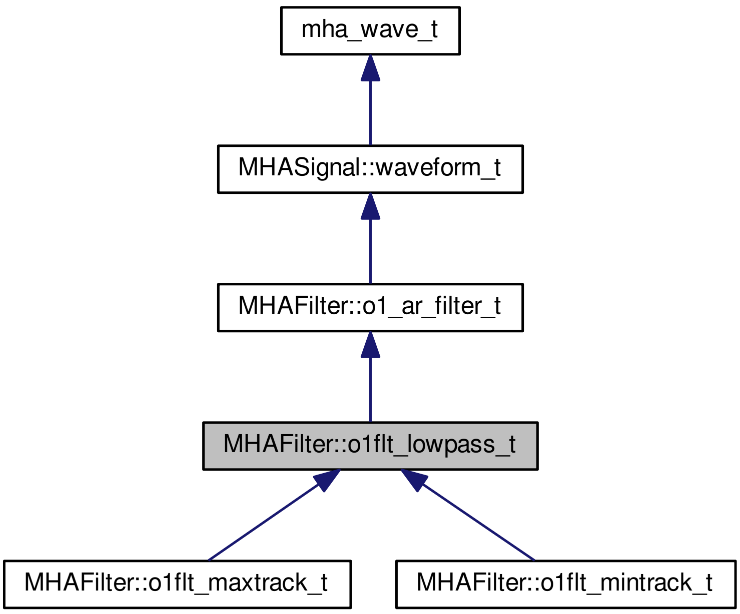
in	Input signal
out	Output signal

The number of channels must match the number of filter bands.

4.65 MHAFilter::o1flt_lowpass_t Class Reference

First order low pass filter.

Inheritance diagram for MHAFilter::o1flt_lowpass_t:



Public Member Functions

- **o1flt_lowpass_t** (const std::vector< **mha_real_t** > &, **mha_real_t**, **mha_real_t**=0)
  Constructor of low pass filter, sets sampling rate and time constants.
- void **set_tau** (unsigned int ch, **mha_real_t** tau)
  change the time constant in one channel
- void **set_tau** (**mha_real_t** tau)
  set time constant in all channels to tau

Additional Inherited Members

4.65.1 Constructor & Destructor Documentation

4.65.1.1 MHAFilter::o1flt_lowpass_t::o1flt_lowpass_t (const std::vector< mha_real_t > & *tau*, mha_real_t *fs*, mha_real_t *startval* = 0)

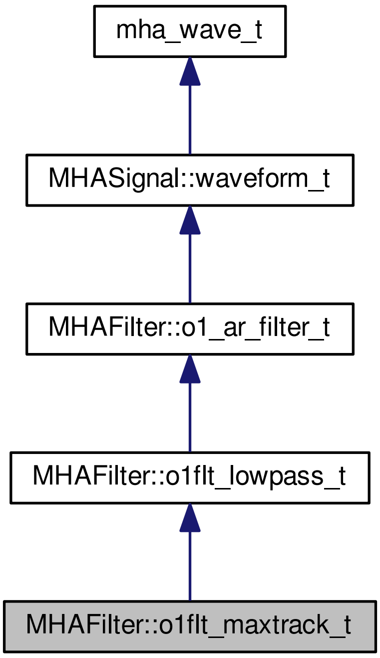
Parameters

tau	Vector of time constants
fs	Sampling rate
startval	Initial internal state value

4.66 MHAFilter::o1flt_maxtrack_t Class Reference

First order maximum tracker.

Inheritance diagram for MHAFilter::o1flt_maxtrack_t:



Public Member Functions

- **o1flt_maxtrack_t** (const std::vector< **mha_real_t** > &, **mha_real_t**, **mha_real_t**=0)
  Constructor of low pass filter, sets sampling rate and time constants.
- void **set_tau** (unsigned int ch, **mha_real_t** tau)
  change the time constant in one channel
- void **set_tau** (**mha_real_t** tau)
  set time constant in all channels to tau

Additional Inherited Members

4.66.1 Detailed Description

4.66.2 Constructor & Destructor Documentation

4.66.2.1 MHAFilter::o1flt_maxtrack_t::o1flt_maxtrack_t (const std::vector< mha_real_t > & *tau*, mha_real_t *fs*, mha_real_t *startval* = 0)

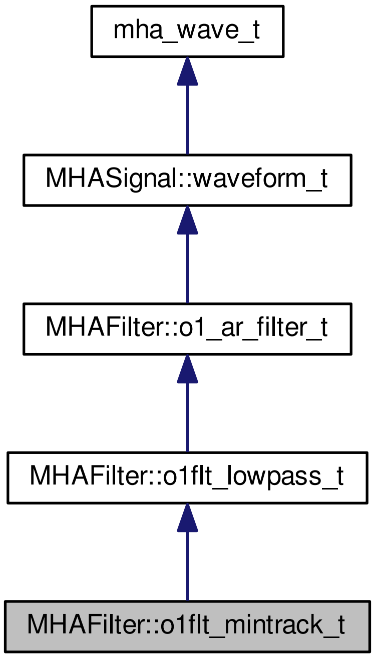
Parameters

tau	Vector of time constants
fs	Sampling rate
startval	Initial internal state value

4.67 MHAFilter::o1flt_mintrack_t Class Reference

First order minimum tracker.

Inheritance diagram for MHAFilter::o1flt_mintrack_t:



Public Member Functions

- void **set_tau** (unsigned int ch, **mha_real_t** tau)
  change the time constant in one channel
- void **set_tau** (**mha_real_t** tau)
  set time constant in all channels to tau

Additional Inherited Members

4.67.1 Detailed Description

4.68 MHAFilter::partitioned_convolution_t Class Reference

A filter class for partitioned convolution.

Classes

- struct **index_t**
  Bookkeeping class.

Public Member Functions

- **partitioned_convolution_t** (unsigned int **fragsize**, unsigned int **nchannels_in**, unsigned int **nchannels_out**, const **transfer_matrix_t** &transfer)
  Create a new partitioned convolver.
- ∼**partitioned_convolution_t** ()
  Free fftw resource allocated in constructor.
- **mha_wave_t** ∗ **process** (const **mha_wave_t** ∗s_in)
  processing

Public Attributes

- unsigned int **fragsize**
  Audio fragment size, always equal to partition size.
- unsigned int **nchannels_in**
  Number of audio input channels.
- unsigned int **nchannels_out**
  Number of audio output channels.
- unsigned int **output_partitions**
  The maximum number of partitions in any of the impulse responses.
- unsigned int **filter_partitions**
  The total number of non-zero impulse response partitions.
- **MHASignal::waveform_t input_signal_wave**
  Buffer for input signal.
- unsigned int **current_input_signal_buffer_half_index**
  A counter modulo 2.
- **MHASignal::spectrum_t input_signal_spec**
  Buffer for FFT transformed input signal.
- **MHASignal::spectrum_t frequency_response**
  Buffers for frequency response spectra of impulse response partitions.

- std::vector< **index_t** > **bookkeeping**

  Keeps track of input channels, output channels, impulse response partition, and delay.
- std::vector< **MHASignal::spectrum_t** > **output_signal_spec**

  Buffers for FFT transformed output signal.
- unsigned int **current_output_partition_index**

  A counter modulo output_partitions, indexing the "current" output partition.
- **MHASignal::waveform_t output_signal_wave**

  Buffer for the wave output signal.
- **mha_fft_t fft**

  The FFT transformer.

4.68.1 Detailed Description

Impulse responses are partitioned into sections of fragment size. Audio signal is convolved with every partition and delayed as needed. Convolution is done according to overlap-save. FFT length used is 2 times fragment size.

4.68.2 Constructor & Destructor Documentation

4.68.2.1 MHAFilter::partitioned_convolution_t::partitioned_convolution_t (unsigned int *fragsize*, unsigned int *nchannels_in*, unsigned int *nchannels_out*, const transfer_matrix_t & *transfer*)

Parameters

fragsize	Audio fragment size, equal to partition size.
nchannels_in	Number of input audio channels.
nchannels_out	Number of output audio channels.
transfer	A sparse matrix of impulse responses.

4.68.3 Member Data Documentation

4.68.3.1 unsigned int MHAFilter::partitioned_convolution_t::fragsize

4.68.3.2 unsigned int MHAFilter::partitioned_convolution_t::nchannels_in

4.68.3.3 unsigned int MHAFilter::partitioned_convolution_t::nchannels_out

4.68.3.4 unsigned int MHAFilter::partitioned_convolution_t::output_partitions

Determines the size if the delay line.

4.68.3.5 unsigned int MHAFilter::partitioned_convolution_t::filter_partitions

4.68.3.6 MHASignal::waveform_t MHAFilter::partitioned_convolution_t::input_signal_wave

Has nchannels_in channels and fragsize∗2 frames

4.68.3.7 unsigned int MHAFilter::partitioned_convolution_t::current_input_signal_buffer_half_index

Indicates the buffer half in input signal wave into which to copy the current input signal.

4.68.3.8 MHASignal::spectrum_t MHAFilter::partitioned_convolution_t::input_signal_spec

Has nchannels_in channels and fragsize+1 frames (fft bins).

4.68.3.9 MHASignal::spectrum_t MHAFilter::partitioned_convolution_t::frequency_response

Each "channel" contains another partition of some impulse response. The bookkeeping array is used to keep track what to do with these frequency responses. This container has filter_↩ partitions channels and fragsize+1 frames (fft bins).

4.68.3.10 std::vector<index_t> MHAFilter::partitioned_convolution_t::bookkeeping

The index into this array is the same as the "channel" index into the frequency_response array. Array has filter_partitions entries.

4.68.3.11 std::vector<MHASignal::spectrum_t> MHAFilter::partitioned_convolution_t::output_↩ signal_spec

For each array member, Number of channels is equal to nchannels_out, number of frames (fft bins) is equal to fragsize+1. Array size is equal to output_partitions.

4.68.3.12 unsigned int MHAFilter::partitioned_convolution_t::current_output_partition_index

4.68.3.13 MHASignal::waveform_t MHAFilter::partitioned_convolution_t::output_signal_wave

Number of channels is equal to nchannels_out, number of frames is equal to fragsize

4.69 MHAFilter::partitioned_convolution_t::index_t Struct Reference

Bookkeeping class.

Public Member Functions

- **index_t** (unsigned int src, unsigned int tgt, unsigned int dly)
  Data constructor.
- **index_t** ()
  Default constructor for STL compatibility.

Public Attributes

- unsigned int **source_channel_index**
  The input channel index to apply the current partition to.
- unsigned int **target_channel_index**
  The index of the output channel to which the filter result should go.
- unsigned int **delay**
  The delay (in blocks) of this partition.

4.69.1 Detailed Description

For each impulse response partition, keeps track of which input to filter, which output channel to filter to, and the delay in blocks. Objects of class Index should be kept in an array with the same indices as the corresponding inpulse response partitions.

4.69.2 Constructor & Destructor Documentation

4.69.2.1 MHAFilter::partitioned_convolution_t::index_t::index_t (unsigned int *src*, unsigned int *tgt*, unsigned int *dly*) [inline]

Parameters

src	The input channel index to apply the current partition to.
tgt	The index of the output channel to which the filter result should go.
dly	The delay (in blocks) of this partition

4.69.3 Member Data Documentation

4.69.3.1 unsigned int MHAFilter::partitioned_convolution_t::index_t::source_channel_index

4.69.3.2 unsigned int MHAFilter::partitioned_convolution_t::index_t::target_channel_index

4.70 MHAFilter::polyphase_resampling_t Class Reference

A class that does polyphase resampling.

Public Member Functions

- **polyphase_resampling_t** (unsigned n_up, unsigned n_down, **mha_real_t** nyquist_ratio, unsigned n_irs, unsigned n_ringbuffer, unsigned n_channels, unsigned n_prefill)

  Initialize a polyphase resampler.
- void **write** (**mha_wave_t** &signal)

  Write signal to the ringbuffer.
- void **read** (**mha_wave_t** &signal)

  Read resampled signal.
- unsigned **readable_frames** () const

  Number of frames at target sampling rate that can be produced.

Private Attributes

- unsigned **upsampling_factor**

  Interpolation rate / source rate.
- unsigned **downsampling_factor**

  Interpolation rate / target rate.
- unsigned **now_index**

  points to "now" in the interpolated sampling rate
- bool **underflow**

  indicates if an underflow has occurred. Object cannot be used then.
- **MHAWindow::hanning_t impulse_response**

  contains the lowpass impulse response at interpolation rate
- **MHASignal::ringbuffer_t ringbuffer**

  storage of input signal

4.70.1 Detailed Description

4.70.2 Constructor & Destructor Documentation

4.70.2.1 MHAFilter::polyphase_resampling_t::polyphase_resampling_t (unsigned *n_up,* unsigned *n_down,* mha_real_t *nyquist_ratio,* unsigned *n_irs,* unsigned *n_ringbuffer,* unsigned *n_channels,* unsigned *n_prefill*)

Parameters

n_up	upsampling factor
n_down	downsampling factor
nyquist_ratio	low pass filter cutoff frequency relative to the nyquist frequency of the smaller of the two sampling rates. Example values: 0.8, 0.9
n_irs	length of impulse response (in samples at interpolation rate)
n_ringbuffer	length of ringbuffer, in samples at source sampling rate
n_channels	audio channels count
n_prefill	Prefill the ringbuffer with this many zero frames in samples at source sampling rate

4.70.3 Member Function Documentation

4.70.3.1 void MHAFilter::polyphase_resampling_t::write (mha_wave_t & *signal*)

Parameters

signal	input signal in original sampling rate

Exceptions

MHA_Error *(p. 132)*	Raises exception if there is not enough room or if the number of channels does not match.

4.70.3.2 void MHAFilter::polyphase_resampling_t::read (mha_wave_t & *signal*)

Will perform the resampling and remove no longer needed samples from the input buffer.

Parameters

signal	buffer to write the resampled signal to.

Exceptions

MHA_Error *(p. 132)*	Raises exception if there is not enough input signal or if the number of channels is too high.

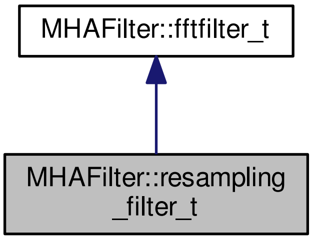
4.70.3.3 unsigned MHAFilter::polyphase_resampling_t::readable_frames ( ) const `[inline]`

Warning: This method only checks for enough future samples present, therefore, this number can be positive and a read operation can still fail if there are not enough past samples present to perform the filtering for the first output sample.

4.71 MHAFilter::resampling_filter_t Class Reference

Hann shaped low pass filter for resampling.

Inheritance diagram for MHAFilter::resampling_filter_t:



Public Member Functions

- **resampling_filter_t** (unsigned int fftlen, unsigned int irslen, unsigned int channels, unsigned int Nup, unsigned int Ndown, double fCutOff)

    Constructor.

4.71.1 Detailed Description

This class uses FFT filter at upsampled rate.

4.71.2 Constructor & Destructor Documentation

4.71.2.1 MHAFilter::resampling_filter_t::resampling_filter_t (unsigned int *fftlen,* unsigned int *irslen,* unsigned int *channels,* unsigned int *Nup,* unsigned int *Ndown,* double *fCutOff*)

Parameters

fftlen	FFT length.
irslen	Length of filter.
channels	Number of channels to be filtered.
Nup	Upsampling ratio.
Ndown	Downsampling ratio.
fCutOff	Cut off frequency (relative to lower Nyquist Frequency)

4.72 MHAFilter::smoothspec_t Class Reference

Smooth spectral gains, create a windowed impulse response.

Public Member Functions

- **smoothspec_t** (unsigned int fftlen, unsigned int nchannels, const **MHAWindow::base↩_t** &window, bool minphase, bool linphase_asym=false)
  Constructor.
- void **smoothspec** (const **mha_spec_t** &s_in, **mha_spec_t** &s_out)
  Create a smoothed spectrum.
- void **smoothspec** (**mha_spec_t** &spec)
  Create a smoothed spectrum (in place)
- void **spec2fir** (const **mha_spec_t** &spec, **mha_wave_t** &fir)
  Return FIR coefficients.

4.72.1 Detailed Description

Spectral gains are smoothed by multiplicating the impulse response with a window function.

If a minimal phase is used, then the original phase is discarded and replaced by the minimal phase function. In this case, the window is applied to the beginning of the inverse Fourier transform of the input spectrum, and the remaining signal set to zero. If the original phase is kept, the window is applied symmetrical arround zero, i.e. to the first and last samples of the inverse Fourier transform of the input spectrum. The **spec2fir()** (p. 185) function creates a causal impulse response by circular shifting the impulse response by half of the window length.

The signal dimensions of the arguments of **smoothspec()** (p. 184) must correspond to the FFT length and number of channels provided in the constructor. The function **spec2fir()** (p. 185) can fill signal structures with more than window length frames.

4.72.2 Constructor & Destructor Documentation

4.72.2.1 MHAFilter::smoothspec_t::smoothspec_t (unsigned int *fftlen*, unsigned int *nchannels*, const MHAWindow::base_t & *window*, bool *minphase*, bool *linphase_asym* = `false`)

Parameters

fftlen	FFT length of input spectrum (fftlen/2+1 bins)
nchannels	Number of channels in input spectrum
window	Window used for smoothing
minphase	Use minimal phase (true) or original phase (false)
linphase_asym	Keep phase, but apply full window at beginning of IRS

4.72.3 Member Function Documentation

4.72.3.1 void MHAFilter::smoothspec_t::smoothspec (const mha_spec_t & *s_in*, mha_spec_t & *s_out*)

Parameters

s↩ _in	Input spectrum

Return values

s_out	Output spectrum

4.72.3.2 void MHAFilter::smoothspec_t::smoothspec (mha_spec_t & *spec*) [inline]

Parameters

spec	Spectrum to be smoothed.

4.72.3.3 void MHAFilter::smoothspec_t::spec2fir (const mha_spec_t & *spec*, mha_wave_t & *fir*)

Parameters

spec	Input spectrum

Return values

fir	FIR coefficients, minimum length is window length

4.73 MHAFilter::transfer_function_t Struct Reference

a structure containing a source channel number, a target channel number, and an impulse response.

Public Member Functions

- **transfer_function_t** ()
  Default constructor for STL conformity.
- **transfer_function_t** (unsigned int **source_channel_index**, unsigned int **target_↩ channel_index**, const std::vector< float > &**impulse_response**)
  Data constructor.
- unsigned int **partitions** (unsigned int fragsize) const
  for the given partition size, return the number of partitions of the impulse response.
- unsigned int **non_empty_partitions** (unsigned int fragsize) const
  for the given partition size, return the number of non-empty partitions of the impulse response.
- bool **isempty** (unsigned int fragsize, unsigned int index) const
  checks if the partition contains only zeros

Public Attributes

- unsigned int **source_channel_index**
  Source audio channel index for this transfer function.
- unsigned int **target_channel_index**
  Target audio channel index for this transfer function.
- std::vector< float > **impulse_response**
  Impulse response of transfer from source to target channel.

4.73.1 Constructor & Destructor Documentation

4.73.1.1 MHAFilter::transfer_function_t::transfer_function_t ( ) [inline]

Not used.

4.73.1.2 MHAFilter::transfer_function_t::transfer_function_t (unsigned int *source_channel_index*, unsigned int *target_channel_index*, const std::vector< float > & *impulse_response*)

Parameters

source_channel_index	Source audio channel index for this transfer function
target_channel_index	Target audio channel index for this transfer function
impulse_response	Impulse response of transfer from source to target channel

4.73.2 Member Function Documentation

4.73.2.1 unsigned int MHAFilter::transfer_function_t::partitions (unsigned int *fragsize*) const [inline]

Parameters

fragsize	partition size

Returns

number of partitions occupied by the impulse response

4.73.2.2 unsigned int MHAFilter::transfer_function_t::non_empty_partitions (unsigned int *fragsize*) const [inline]

Parameters

fragsize	partition size

Returns

the number of non-empty partitions of the impulse response, i.e. partitions containing only zeros are not counted.

4.73.2.3 bool MHAFilter::transfer_function_t::isempty (unsigned int *fragsize*, unsigned int *index*) const [inline]

Parameters

fragsize	partition size
index	partition index

Returns

true when this partition of the impulse response contains only zeros.

4.74 MHAFilter::transfer_matrix_t Struct Reference

A sparse matrix of transfer function partitionss.

Inherits vector< transfer_function_t >.

Public Member Functions

- std::valarray< unsigned int > **partitions** (unsigned fragsize) const

  Returns an array of the results of calling the ***partitions()*** *(p. 187) method on every matrix member.*
- std::valarray< unsigned int > **non_empty_partitions** (unsigned int fragsize) const

  Returns an array of the results of calling the ***non_empty_partitions()*** *(p. 187) method on every matrix member.*

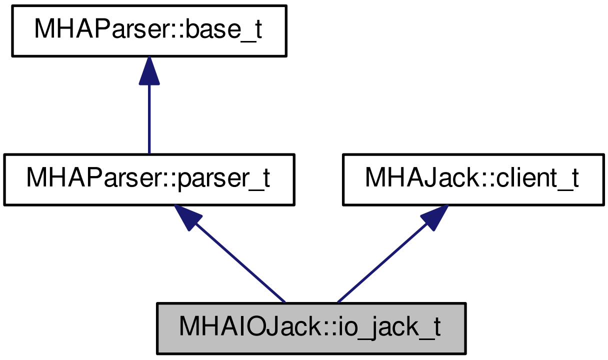
4.74.1 Detailed Description

Each matrix element knows its position in the matrix, so they can be stored as a vector.

4.75 MHAIOJack::io_jack_t Class Reference

Main class for JACK IO.

Inheritance diagram for MHAIOJack::io_jack_t:



Public Member Functions

- void **prepare** (int, int)

  Allocate buffers, activate JACK client and install internal ports.

Private Member Functions

- void **reconnect_inports** ()

  Connect the input ports when connection variable is accessed.
- void **reconnect_outports** ()

  Connect the output ports when connection variable is accessed.

Additional Inherited Members

4.75.1 Detailed Description

This class registers a JACK client. JACK and framework states are managed by this class.

4.75.2 Member Function Documentation

4.75.2.1 void io_jack_t::prepare (int *nch_in,* int *nch_out*)

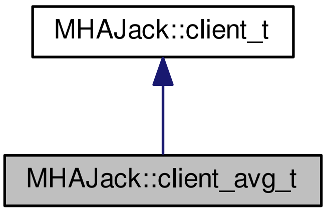
4.75.2.2 void io_jack_t::reconnect_inports ( ) [private]

4.75.2.3 void io_jack_t::reconnect_outports ( ) [private]

4.76 MHAJack::client_avg_t Class Reference

Generic JACK client for averaging a system response across time.

Inheritance diagram for MHAJack::client_avg_t:



Public Member Functions

- **client_avg_t** (const std::string &name, const unsigned int &nrep_)

  Constructor for averaging client.
- void **io** (**mha_wave_t** ∗s_out, **mha_wave_t** ∗s_in, const std::vector< std::string > &p_↩
  out, const std::vector< std::string > &p_in, float ∗srate=NULL, unsigned int ∗fragsize=N↩
  ULL)

  Recording function.

4.76.1 Detailed Description

4.76.2 Constructor & Destructor Documentation

4.76.2.1 MHAJack::client_avg_t::client_avg_t (const std::string & *name_,* const unsigned int & *nrep_*)

Parameters

*name_*	Name of JACK client
*nrep_*	Number of repetitions

4.76.3 Member Function Documentation

4.76.3.1 void MHAJack::client_avg_t::io (mha_wave_t * *is_out*, mha_wave_t * *is_in*, const std::vector< std::string > & *p_out*, const std::vector< std::string > & *p_in*, float * *srate* = NULL, unsigned int * *fragsize* = NULL)

long-description

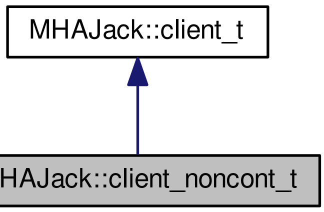
Parameters

is_out	Input (test) signal, which will be repeated
is_in	System response (averaged, same length as input required)
p_out	Ports to play back the test signal
p_in	Ports to record from the system response
srate	Pointer to sampling rate variable, will be filled with server sampling rate
fragsize	Pointer to fragment size variable, will be filled with server fragment size

4.77 MHAJack::client_noncont_t Class Reference

Generic client for synchronous playback and recording of waveform fragments.

Inheritance diagram for MHAJack::client_noncont_t:

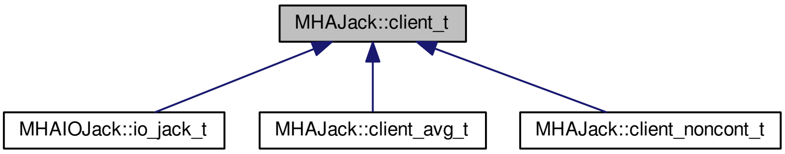


Additional Inherited Members

4.78 MHAJack::client_t Class Reference

Generic asynchronous JACK client.

Inheritance diagram for MHAJack::client_t:



Public Member Functions

- void **prepare** (const std::string &client_name, const unsigned int &nchannels_in, const unsigned int &nchannels_out)

  Allocate buffers, activate JACK client and install internal ports.
- void **prepare** (const std::string &server_name, const std::string &client_name, const unsigned int &nchannels_in, const unsigned int &nchannels_out)

  Allocate buffers, ports, and activates JACK client.
- void **release** ()

  Remove JACK client and deallocate internal ports and buffers.
- void **connect_input** (const std::vector< std::string > &)

  Connect the input ports when connection variable is accessed.
- void **connect_output** (const std::vector< std::string > &)

  Connect the output ports when connection variable is accessed.
- void **get_ports** (std::vector< std::string > &, unsigned long jack_flags)

  Get a list of Jack ports.

Private Member Functions

- void **prepare_impl** (const char ∗server_name, const char ∗client_name, const unsigned int &nchannels_in, const unsigned int &nchannels_out)

  Allocate buffers, activate JACK client and allocates jack ports Registers the jack client with the given server and activates it.
- int **jack_proc_cb** (jack_nframes_t)

  This is the main processing callback.

4.78.1 Detailed Description

4.78.2 Member Function Documentation

4.78.2.1 void MHAJack::client_t::prepare (const std::string & *client_name,* const unsigned int & *nch_in,* const unsigned int & *nch_out*)

Registers the jack client with the default jack server and activates it.

Parameters

client_name	Name of this jack client
nch_in	Input ports to register
nch_out	Output ports to register

4.78.2.2 void MHAJack::client_t::prepare (const std::string & *server_name,* const std::string & *client_name,* const unsigned int & *nch_in,* const unsigned int & *nch_out*)

Registers the jack client with specified jack server and activates it.

Parameters

server_name	Name of the jack server to register with
client_name	Name of this jack client
nch_in	Input ports to register
nch_out	Output ports to register

4.78.2.3 void MHAJack::client_t::release ( )

4.78.2.4 void MHAJack::client_t::connect_input (const std::vector< std::string > & *con*)

4.78.2.5 void MHAJack::client_t::connect_output (const std::vector< std::string > & *con*)

4.78.2.6 void MHAJack::client_t::get_ports (std::vector< std::string > & *res,* unsigned long *jack_flags*)

Parameters

res	Result string vector
jack_flags	Jack port flags (JackPortInput etc.)

4.78.2.7 void MHAJack::client_t::prepare_impl (const char ∗ *server_name,* const char ∗ *client_name,* const unsigned int & *nch_in,* const unsigned int & *nch_out*) [private]

Parameters

server_name	Name of the jack server to register with
client_name	Name of this jack client
nch_in	Input ports to register
nch_out	Output ports to register

4.78.2.8 int MHAJack::client_t::jack_proc_cb (jack_nframes_t *n*) [private]

Here happens double buffering and downsampling.

4.79 MHAJack::port_t Class Reference

Class for one channel/port.

Public Member Functions

- **port_t** (jack_client_t ∗jc, dir_t dir, int id)
- **port_t** (jack_client_t ∗jc, dir_t dir, const std::string &id)

  Constructor to create port with specific name.
- void **read** (**mha_wave_t** ∗s, unsigned int ch)
- void **write** (**mha_wave_t** ∗s, unsigned int ch)
- void **mute** (unsigned int n)
- void **connect_to** (const char ∗pn)
- const char ∗ **get_short_name** ()

  Return the port name.

4.79.1 Detailed Description

This class represents one JACK port. Double buffering for asynchronous process callbacks is managed by this class.

4.79.2 Constructor & Destructor Documentation

4.79.2.1 MHAJack::port_t::port_t (jack_client_t ∗ *jc,* dir_t *dir,* int *id*)

Parameters

jc	JACK client.
dir	Direction (input/output).
id	Number in port name (starting with 1).

4.79.2.2 MHAJack::port_t::port_t (jack_client_t * *jc*, dir_t *dir*, const std::string & *id*)

Parameters

jc	JACK client.
dir	Direction (input/output).
id	Port name.

4.79.3 Member Function Documentation

4.79.3.1 void MHAJack::port_t::read (mha_wave_t * *s*, unsigned int *ch*)

Parameters

s	Signal structure to store the audio data.
ch	Channel number in audio data structure to be used.

4.79.3.2 void MHAJack::port_t::write (mha_wave_t * *s*, unsigned int *ch*)

Parameters

s	Signal structure from which the audio data is read.
ch	Channel number in audio data structure to be used.

4.79.3.3 void MHAJack::port_t::mute (unsigned int *n*)

Parameters

n	Number of samples to be muted (must be the same as reported by Jack processing callback).

4.79.3.4 void MHAJack::port_t::connect_to (const char * *pn*)

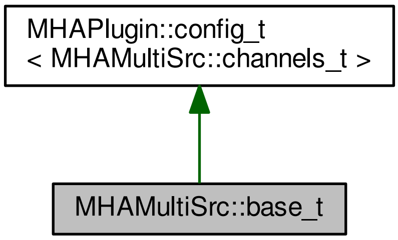
Parameters

pn	Port name to connect to

4.80 MHAMultiSrc::base_t Class Reference

Base class for source selection.

Inheritance diagram for MHAMultiSrc::base_t:



Public Member Functions

- void **select_source** (const std::vector< std::string > &src, int in_channels)
  Change the selection of input sources.

4.80.1 Detailed Description

See also

MHAMultiSrc::channel_t
MHAMultiSrc::channels_t

4.80.2 Member Function Documentation

4.80.2.1 void MHAMultiSrc::base_t::select_source (const std::vector< std::string > & *src,* int *in_channels*)

This function is real-time and thread safe.

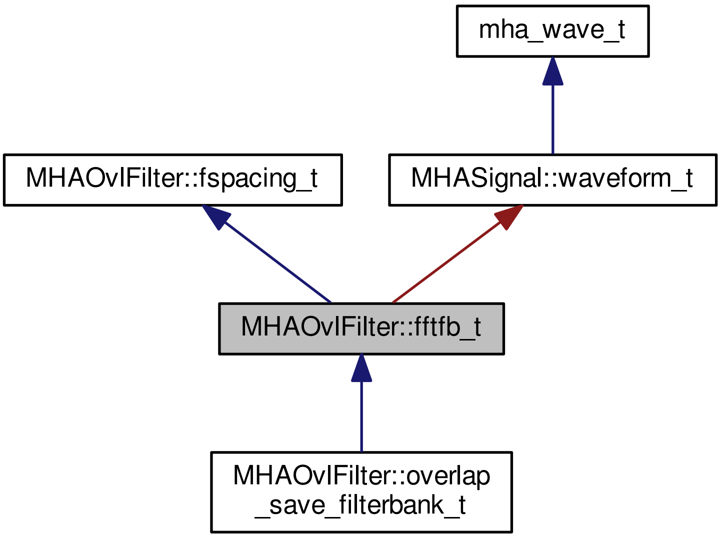
Parameters

src	List of input sources
in_channels	Number of input channels in direct input (the processed signal)

4.81 MHAOvlFilter::fftfb_t Class Reference

FFT based overlapping filter bank.

Inheritance diagram for MHAOvlFilter::fftfb_t:



Public Member Functions

- **fftfb_t** (**MHAOvlFilter::fftfb_vars_t** &par, unsigned int nfft, **mha_real_t** fs)
  Constructor for a FFT-based overlapping filter bank.
- unsigned int **bin1** (unsigned int band) const
  Return index of first non-zero filter shape window.
- unsigned int **bin2** (unsigned int band) const
  Return index of first zero filter shape window above center frequency.
- unsigned int **get_fftlen** () const
  Return fft length.
- **mha_real_t** **w** (unsigned int k, unsigned int b) const
  Return filter shape window at index k in band b.

Additional Inherited Members

4.81.1 Constructor & Destructor Documentation

4.81.1.1 MHAOvlFilter::fftfb_t::fftfb_t (**MHAOvlFilter::fftfb_vars_t** & *par*, unsigned int *nfft*, **mha_real_t** *fs*)

Parameters

par	Parameters for the FFT filterbank that can not be deduced from the signal dimensions are taken from this set of configuration variables.
nfft	FFT length
fs	Sampling rate / Hz

4.81.2 Member Function Documentation

4.81.2.1 mha_real_t MHAOvlFilter::fftfb_t::w (unsigned int *k*, unsigned int *b*) const [inline]

Parameters

k	Frequency index
b	Band index

4.82 MHAOvlFilter::fftfb_vars_t Class Reference

Set of configuration variables for FFT-based overlapping filters.

Inherited by MHAOvlFilter::overlap_save_filterbank_t::vars_t.

Public Member Functions

- **fftfb_vars_t** (**MHAParser::parser_t** &p)
  construct a set of openMHA configuration language variables suitable for configuring the FFT-based overlapping filterbank.

Public Attributes

- scale_var_t **fscale**
  Frequency scale type (lin/bark/log/erb).
- scale_var_t **ovltype**
  Filter shape (rect/lin/hann).
- **MHAParser::float_t plateau**
  relative plateau width.
- **MHAParser::kw_t ftype**
  Flag to decide wether edge or center frequencies are used.
- fscale_t **f**
  Frequency.
- **MHAParser::bool_t normalize**
  Normalize sum of channels.
- **MHAParser::bool_t fail_on_nonmonotonic**

Fail if frequency entries are non-monotonic (otherwise sort)
- **MHAParser::bool_t fail_on_unique_bins**

  Fail if center frequencies share the same FFT bin.
- **MHAParser::vfloat_mon_t cf**

  Final center frequencies in Hz.
- **MHAParser::vfloat_mon_t ef**

  Final edge frequencies in Hz.
- **MHAParser::vfloat_mon_t cLTASS**

  Bandwidth correction for LTASS noise (level of 0 dB RMS LTASS noise)

4.82.1 Detailed Description

This class enables easy configuration of the FFT-based overlapping filterbank. An instance of **fftfb_vars_t** (p. 197) creates openMHA configuration language variables needed for configuring the filterbank, and inserts these variables in the openMHA configuration tree.

This way, the variables are visible to the user and can be configured using the openMHA configuration language.

4.82.2 Constructor & Destructor Documentation

4.82.2.1 MHAOvlFilter::fftfb_vars_t::fftfb_vars_t (MHAParser::parser_t & *p*)

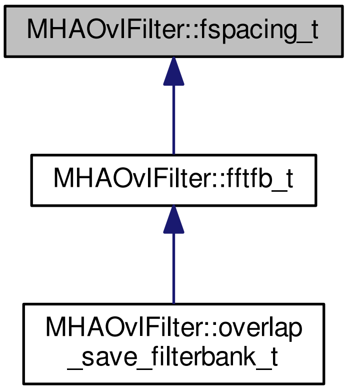
Parameters

p	The node of the configuration tree where the variables created by this instance are inserted.

4.83 MHAOvlFilter::fspacing_t Class Reference

Class for frequency spacing, used by filterbank shape generator class.

Inheritance diagram for MHAOvlFilter::fspacing_t:



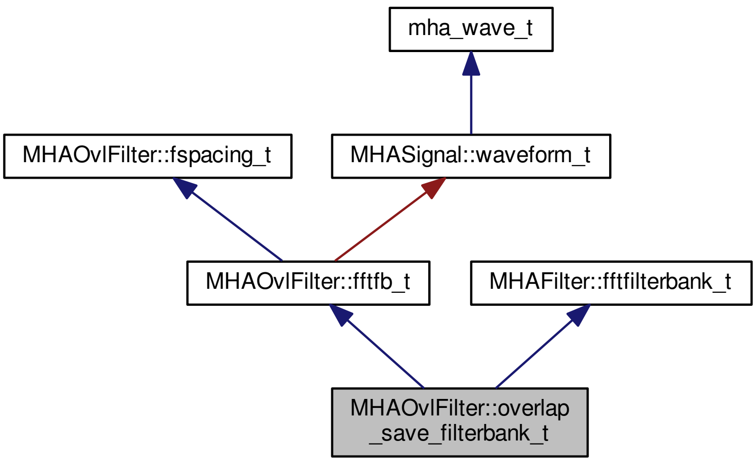
Public Member Functions

- unsigned int **nbands** () const
  Return number of bands in filter bank.

4.84 MHAOvlFilter::overlap_save_filterbank_t Class Reference

A time-domain minimal phase filter bank with frequency shapes from **MHAOvlFilter::fftfb_t** (p. 196).

Inheritance diagram for MHAOvlFilter::overlap_save_filterbank_t:

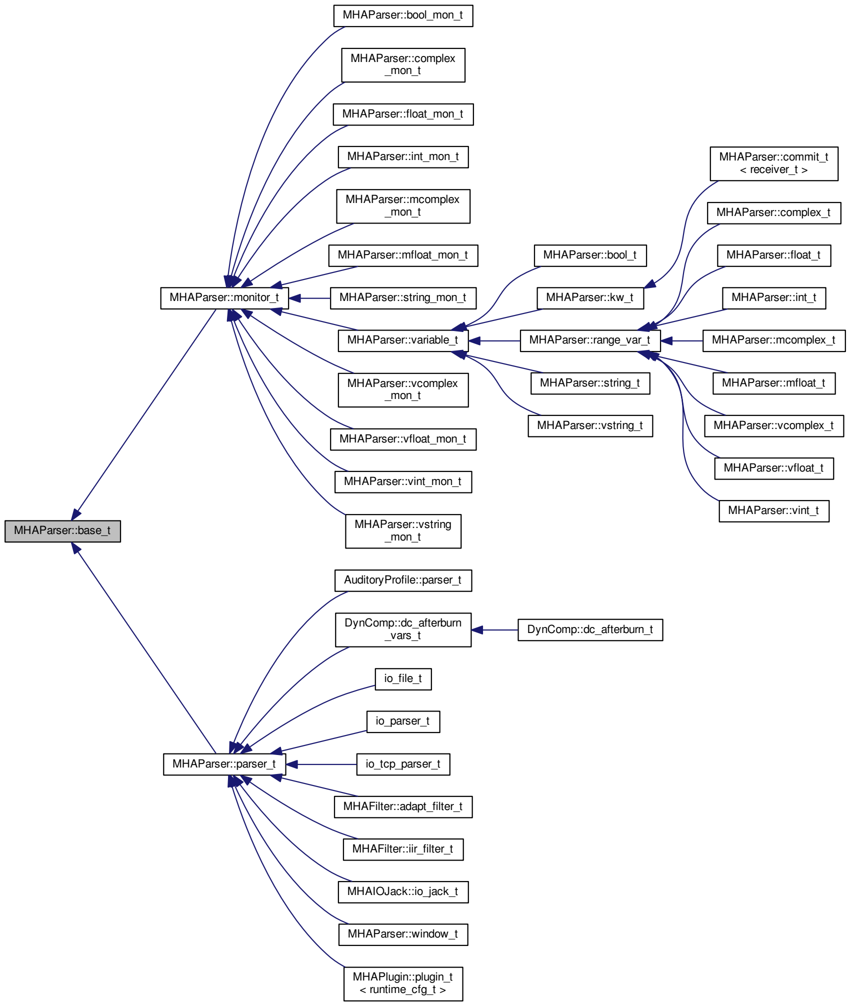


Additional Inherited Members

4.85 MHAParser::base_t Class Reference

Base class for all parser items.

Inheritance diagram for MHAParser::base_t:



Public Member Functions

- **base_t** (const std::string &)
  Constructor for base class of all parser nodes.
- virtual std::string **parse** (const std::string &)
  Causes this node to process a command in the openMHA configuration language.
- virtual void **parse** (const char ∗, char ∗, unsigned int)
  This function parses a command and writes the parsing result into a C character array.

- void **set_node_id** (const std::string &)
  Set the identification string of this parser node.
- void **set_help** (const std::string &)
  Set the help comment of a variable or parser.
- const std::string & **fullname** () const
  Return the full dot-separated path name of this parser node in the openMHA configuration tree.

Public Attributes

- **MHAEvents::emitter_t writeaccess**
  Event emitted on write access.
- **MHAEvents::emitter_t valuechanged**
  Event emitted if the value has changed.
- **MHAEvents::emitter_t readaccess**
  Event emitted on read access.
- **MHAEvents::emitter_t prereadaccess**
  Event emitted on read access, before the data field is accessed.

4.85.1 Detailed Description

The key method of the parser base class is the std::string **parse(const std::string&)** (p. 201) method. Parser proxy derivatives which overwrite any of the other **parse()** (p. 201) methods to be the key method must make sure that the original **parse()** (p. 201) method utilizes the new key method.

4.85.2 Constructor & Destructor Documentation

4.85.2.1 MHAParser::base_t::base_t (const std::string & *h*)

Parameters

h	Help text describing this parser node. This help text is accessible to the configuration language through the "?help" query command.

4.85.3 Member Function Documentation

4.85.3.1 std::string MHAParser::base_t::parse (const std::string & *cs*) [virtual]

Parameters

cs	The command to parse

Returns

The response to the command, if successful

Exceptions

MHA_Error *(p. 132)*	If the command cannot be executed successfully. The reason for failure is given in the message string of the exception.

4.85.3.2 void MHAParser::base_t::parse (const char * *cmd,* char * *retv,* unsigned int *len*) [virtual]

This base class implementation delegates to **parse(const std::string &)** (p. 201).

Parameters

cmd	Command to be parsed
retv	Buffer for the result
len	Length of buffer

4.85.3.3 void MHAParser::base_t::set_node_id (const std::string & *s*)

The id can be queried from the configuration language using the ?id query command. Nodes can be found by id using the ?listid query command on a containing parser node.

Parameters

s	The new identification string.

4.85.3.4 void MHAParser::base_t::set_help (const std::string & *s*)

Parameters

s	New help comment.

4.85.3.5 const std::string & MHAParser::base_t::fullname ( ) const

4.85.4 Member Data Documentation

4.85.4.1 MHAEvents::emitter_t MHAParser::base_t::writeaccess

To connect a callback that is invoked on write access to this parser variable, use MHAEvents::patchbay_t<receiver_t> method connect(&writeaccess,&receiver_t::callback) where callback is a method that expects no parameters and returns void.

4.85.4.2 MHAEvents::emitter_t MHAParser::base_t::valuechanged

To connect a callback that is invoked when write access to this parser variable actually changes its value, use MHAEvents::patchbay_t<receiver_t> method connect(&valuechanged,&receiver↩ _t::callback) where callback is a method that expects no parameters and returns void.

4.85.4.3 MHAEvents::emitter_t MHAParser::base_t::readaccess

To connect a callback that is invoked after the value of this variable has been read through the configuration interface, use MHAEvents::patchbay_t<receiver_t> method connect(&readaccess,&receiver_t::callback) where callback is a method that expects no parameters and returns void.

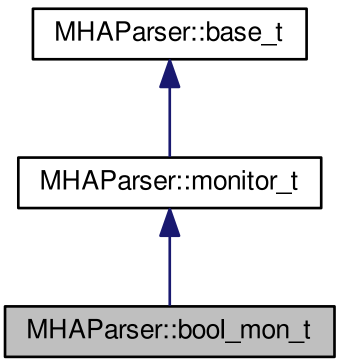
4.85.4.4 MHAEvents::emitter_t MHAParser::base_t::prereadaccess

To connect a callback that is invoked when the value of this variable is about to be read through the configuration interface, so that the callback can influence the value that is reported, use MHAEvents::patchbay_t<receiver_t> method connect(&prereadaccess,&receiver_t::callback) where callback is a method that expects no parameters and returns void.

4.86 MHAParser::bool_mon_t Class Reference

Monitor with string value.

Inheritance diagram for MHAParser::bool_mon_t:



Public Member Functions

- **bool_mon_t** (const std::string &hlp)

    Create a monitor variable for string values.

Public Attributes

- bool **data**
  Data field.

4.86.1 Constructor & Destructor Documentation

4.86.1.1 MHAParser::bool_mon_t::bool_mon_t (const std::string & *hlp*)

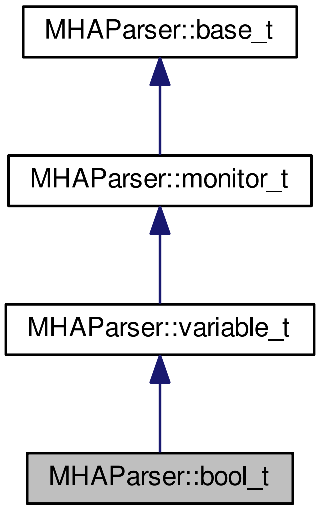
Parameters

hlp	A help text describing this monitor variable.

4.87 MHAParser::bool_t Class Reference

Variable with a boolean value ("yes"/"no")

Inheritance diagram for MHAParser::bool_t:



Public Member Functions

- **bool_t** (const std::string &help_text, const std::string &initial_value)
  Constructor for a configuration language variable for boolean values.

Public Attributes

- bool **data**
  Data field.

4.87.1 Constructor & Destructor Documentation

4.87.1.1 MHAParser::bool_t::bool_t (const std::string & *help_text,* const std::string & *initial_value*)

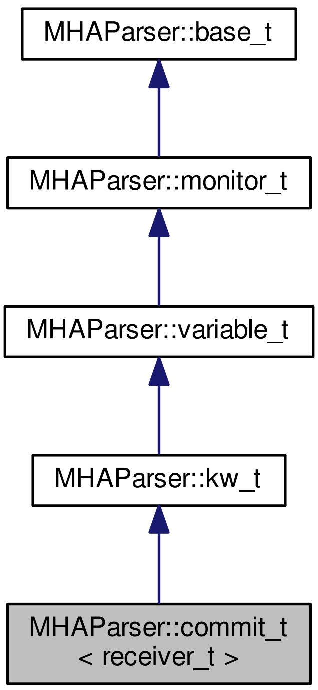
Parameters

help_text	A human-readable text describing the purpose of this configuration variable.
initial_value	The initial value for this variable as a string. The string representation of 'true' is either "yes" or "1". The string representation of 'false' is either "no" or "0".

4.88 MHAParser::commit_t< receiver_t > Class Template Reference

Parser variable with event-emission functionality.

Inheritance diagram for MHAParser::commit_t< receiver_t >:



Additional Inherited Members

4.88.1 Detailed Description

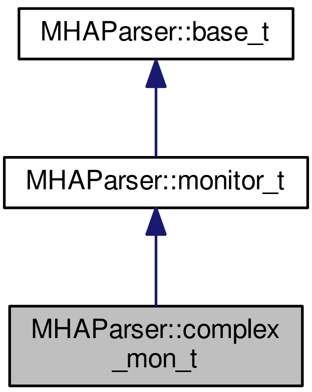
template<class receiver_t>
class MHAParser::commit_t< receiver_t >

The **commit_t** (p. 205) variable can register an event receiver in its constructor, which is called whenever the variable is set to "commit".

4.89 MHAParser::complex_mon_t Class Reference

Monitor with complex value.

Inheritance diagram for MHAParser::complex_mon_t:



Public Member Functions

- **complex_mon_t** (const std::string &hlp)
  Create a complex monitor variable.

Public Attributes

- **mha_complex_t data**
  Data field.

4.89.1 Constructor & Destructor Documentation

4.89.1.1 MHAParser::complex_mon_t::complex_mon_t (const std::string & *hlp*)

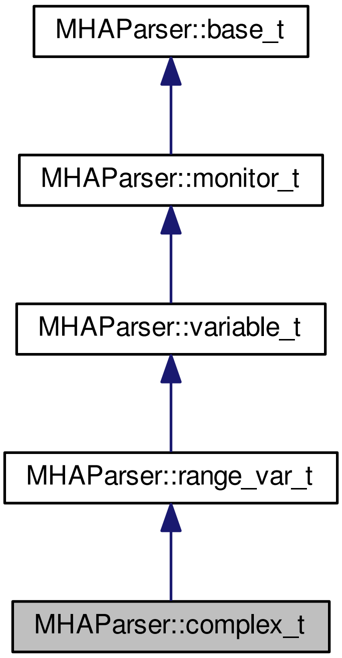
Parameters

hlp	A help text describing this monitor variable.

4.90 MHAParser::complex_t Class Reference

Variable with complex value.

Inheritance diagram for MHAParser::complex_t:



Public Attributes

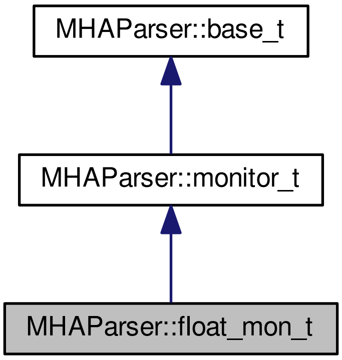
- **mha_complex_t data**
  Data field.

Additional Inherited Members

4.91 MHAParser::float_mon_t Class Reference

Monitor with float value.

Inheritance diagram for MHAParser::float_mon_t:



Public Member Functions

- **float_mon_t** (const std::string &hlp)
  Initialize a floating point (32 bits) monitor variable.

Public Attributes

- float **data**
  Data field.

4.91.1 Constructor & Destructor Documentation

4.91.1.1 MHAParser::float_mon_t::float_mon_t (const std::string & *hlp*)

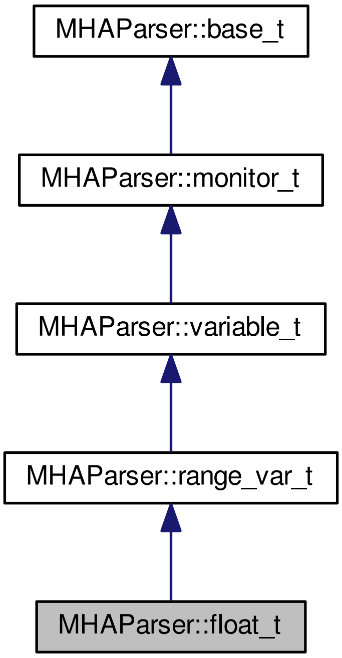
Parameters

hlp	A help text describing this monitor variable.

4.92 MHAParser::float_t Class Reference

Variable with float value.

Inheritance diagram for MHAParser::float_t:



Public Member Functions

- **float_t** (const std::string &help_text, const std::string &initial_value, const std::string &**range**="")

  Constructor for a configuration language variable for 32bit ieee floating-point values.

Public Attributes

- float **data**

  Data field.

Additional Inherited Members

4.92.1 Constructor & Destructor Documentation

4.92.1.1 MHAParser::float_t::float_t (const std::string & *help_text,* const std::string & *initial_value,* const std::string & *range =* "")

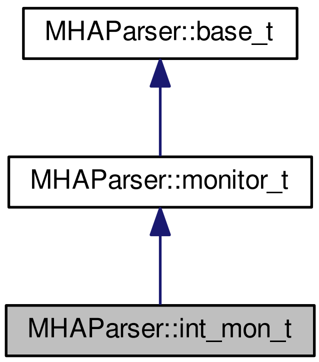
Parameters

help_text	A human-readable text describing the purpose of this configuration variable.
initial_value	The initial value for this variable as a string (decimal representation of the floating-point variable). If a range is given in the third parameter, then the initial value has to be within the range. A human-readable text describing the purpose of this configuration variable.
range	The range of values that this variable can hold can be restricted. A range is a string of the form "[a,b]", where a and b are decimal representations of the inclusive boundaries of the range. a<=b. In a range of the form "]a,b[", both boundaries are excluded. Mixed forms are permitted. a or b can also be omitted if there is no lower or upper limit. The range of values is always restricted by the representable range of the underlying C data type.

4.93 MHAParser::int_mon_t Class Reference

Monitor variable with int value.

Inheritance diagram for MHAParser::int_mon_t:



Public Member Functions

- **int_mon_t** (const std::string &hlp)
  Create a monitor variable for integral values.

Public Attributes

- int **data**
  Data field.

4.93.1 Detailed Description

Monitor variables can be of many types. These variables can be queried through the parser. The public data element contains the monitored state. Write access is only possible from the C++ code by direct access to the data field.

4.93.2 Constructor & Destructor Documentation

4.93.2.1 MHAParser::int_mon_t::int_mon_t (const std::string & *hlp*)

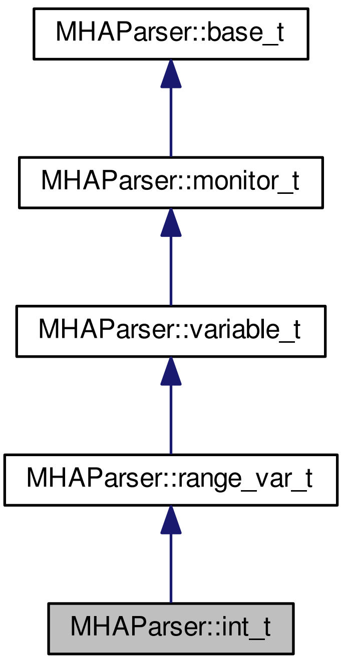
Parameters

hlp	A help text describing this monitor variable.

4.94 MHAParser::int_t Class Reference

Variable with integer value.

Inheritance diagram for MHAParser::int_t:



Public Member Functions

- **int_t** (const std::string &help_text, const std::string &initial_value, const std::string &**range**="")
  Constructor for a configuration language variable for integral values.

Public Attributes

- int **data**
  Data field.

Additional Inherited Members

4.94.1 Constructor & Destructor Documentation

4.94.1.1 MHAParser::int_t::int_t (const std::string & *help_text,* const std::string & *initial_value,* const std::string & *range =* "")

Parameters

help_text	A human-readable text describing the purpose of this configuration variable.
initial_value	The initial value for this variable as a string (decimal representation of the integer variable). If a range is given in the third parameter, then the initial value has to be within the range.
range	The range of values that this variable can hold can be restricted. A range is a string of the form "[a,b]", where a and b are decimal representations of the integral inclusive boundaries of the range. a<=b. In a range of the form "]a,b[", both boundaries are excluded. Mixed forms are permitted. a or b can also be omitted if there is no lower or upper limit. The range of values is always restricted by the representable range of the underlying C data type (usually 32 bits, [-2147483648,2147483647]).

4.95 MHAParser::keyword_list_t Class Reference

Keyword list class.

Public Member Functions

- void **set_value** (const std::string &)
  Select a value from keyword list.
- void **set_entries** (const std::string &)
  Set keyword list entries.

- const std::string & **get_value** () const
  Return selected value.
- const std::vector< std::string > & **get_entries** () const
  Return keyword list.
- const size_t & **get_index** () const
  Return index of selected value.
- void **validate** () const
  Check if index of selected value is valid.
- **keyword_list_t** ()
  Constructor.

Private Attributes

- size_t **index**
  Index into list.
- std::vector< std::string > **entries**
  List of valid entries.

4.95.1 Detailed Description

The stucture **keyword_list_t** (p. 212) defines a keyword list (vector of strings) with an index into the list. Used as **MHAParser::kw_t** (p. 214), it can be used to access a set of valid keywords through the parser (i.e. one of "pear apple banana").

4.95.2 Member Function Documentation

4.95.2.1 void MHAParser::keyword_list_t::set_value (const std::string & *s*)

This function selects a value from the keyword list. The index is set to the last matching entry.

Parameters

s	Value to be selected.

4.95.2.2 void MHAParser::keyword_list_t::set_entries (const std::string & *s*)

With this function, the keyword list can be set from a space separated string list.

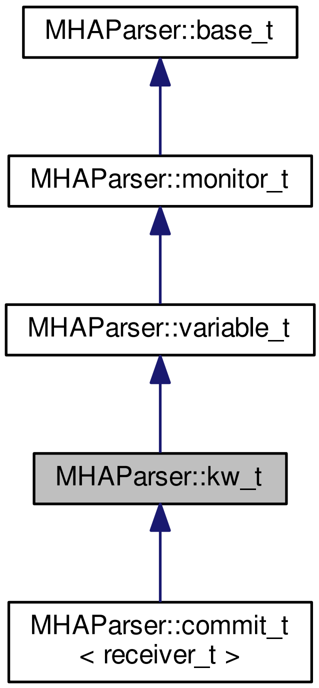
Parameters

s	Space separated entry list.

4.96 MHAParser::kw_t Class Reference

Variable with keyword list value.

Inheritance diagram for MHAParser::kw_t:



Public Member Functions

- **kw_t** (const std::string &, const std::string &, const std::string &)
  Constructor of a keyword list openMHA configuration variable.
- **kw_t** (const **kw_t** &)
  Copy constructor.
- void **set_range** (const std::string &)
  Set/change the list of valid entries.
- bool **isval** (const std::string &) const
  Test if the given value is selected.

Public Attributes

- **keyword_list_t data**
  Variable data in its native type.

4.96.1 Constructor & Destructor Documentation

4.96.1.1 MHAParser::kw_t::kw_t (const std::string & *h*, const std::string & *v*, const std::string & *rg*)

Parameters

h	A help string describing the purpose of this variable.
v	The initial value, has to be a value from the list of possible values given in the last parameter.
rg	A string containing the list of valid entries. The entries have to be separated by spaces. The list of entries has to be delimited by brackets "[", "]".

4.96.2 Member Function Documentation

4.96.2.1 void MHAParser::kw_t::set_range (const std::string & *r*)

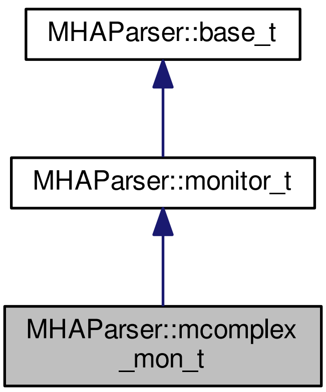
Parameters

r	A string containing the list of valid entries. The entries have to be separated by spaces. The list of entries has to be delimited by brackets "[", "]".

4.97 MHAParser::mcomplex_mon_t Class Reference

Matrix of complex numbers monitor.

Inheritance diagram for MHAParser::mcomplex_mon_t:



Public Member Functions

- **mcomplex_mon_t** (const std::string &hlp)
  Create a matrix of complex floating point monitor values.

Public Attributes

- std::vector< std::vector< **mha_complex_t** > > **data**
  Data field.

4.97.1 Constructor & Destructor Documentation

4.97.1.1 MHAParser::mcomplex_mon_t::mcomplex_mon_t (const std::string & *hlp*)

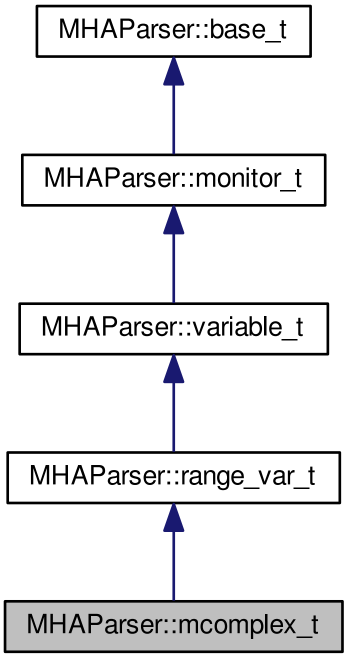
Parameters

hlp	A help text describing this monitor variable.

4.98 MHAParser::mcomplex_t Class Reference

Matrix variable with complex value.

Inheritance diagram for MHAParser::mcomplex_t:



Public Attributes

- std::vector< std::vector< **mha_complex_t** > > **data**

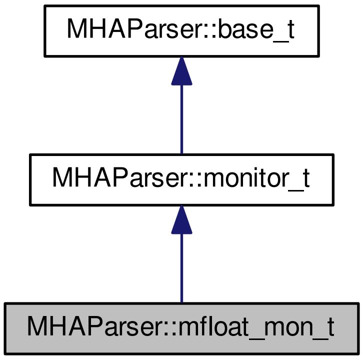
  Data field.

Additional Inherited Members

4.99 MHAParser::mfloat_mon_t Class Reference

Matrix of floats monitor.

Inheritance diagram for MHAParser::mfloat_mon_t:



Public Member Functions

- **mfloat_mon_t** (const std::string &hlp)
  Create a matrix of floating point monitor values.

Public Attributes

- std::vector< std::vector< float > > **data**
  Data field.

4.99.1 Constructor & Destructor Documentation

4.99.1.1 MHAParser::mfloat_mon_t::mfloat_mon_t (const std::string & *hlp*)

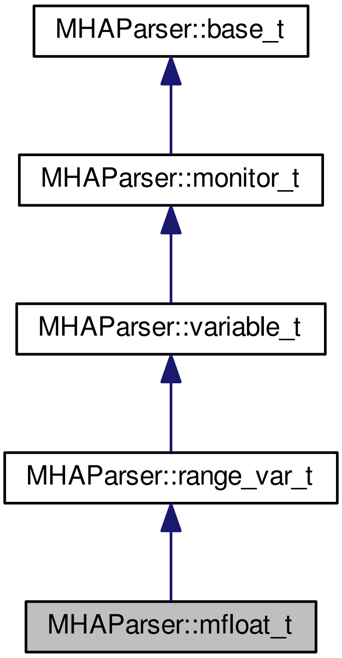
Parameters

hlp	A help text describing this monitor variable.

4.100 MHAParser::mfloat_t Class Reference

Matrix variable with float value.

Inheritance diagram for MHAParser::mfloat_t:



Public Member Functions

- **mfloat_t** (const std::string &, const std::string &, const std::string &="")
  Create a float matrix parser variable.

Public Attributes

- std::vector< std::vector< float > > **data**
  Data field.

Additional Inherited Members

4.100.1 Constructor & Destructor Documentation

4.100.1.1 MHAParser::mfloat_t::mfloat_t (const std::string & *h*, const std::string & *v*, const std::string & *rg* = "")

Parameters

h	A human-readable text describing the purpose of this configuration variable.
v	The initial value of the variable, as a string, in openMHA configuration language: (e.g. "[[0 1]; [2 3]]" for a matrix), described in the "Multidimensional Variables" s2.1.3 section of the openMHA User Manual.
rg	The numeric range to enforce on all members of the matrix.

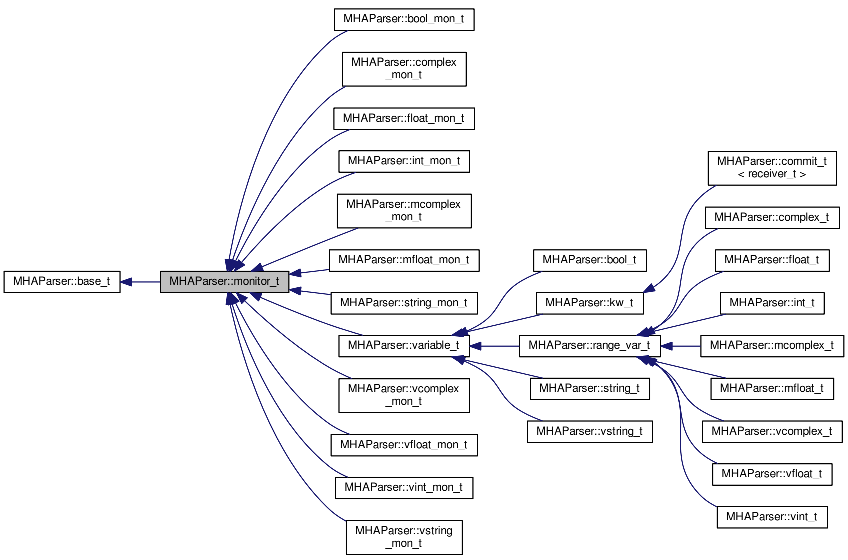
4.101 MHAParser::mhapluginloader_t Class Reference

Class to create a plugin loader in a parser, including the load logic.

4.102 MHAParser::monitor_t Class Reference

Base class for monitors and variable nodes.

Inheritance diagram for MHAParser::monitor_t:

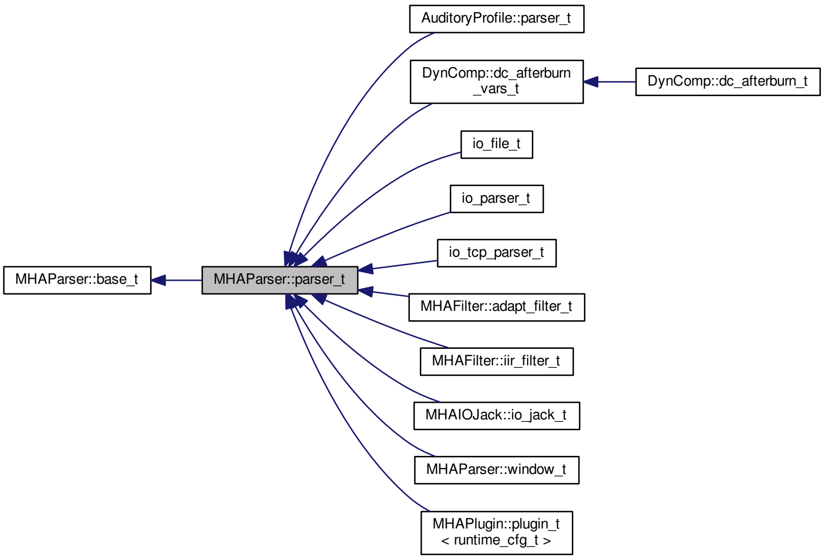


Additional Inherited Members

4.103 MHAParser::parser_t Class Reference

Parser node class.

Inheritance diagram for MHAParser::parser_t:



Public Member Functions

- **parser_t** (const std::string &help_text="")
  Construct detached node to be used in the configuration tree.
- void **insert_item** (const std::string &, **base_t** ∗)
  Register a parser item into this sub-parser.
- void **remove_item** (const std::string &)
  Remove an item by name.
- void **force_remove_item** (const std::string &)
  Remove an item by name.
- void **remove_item** (const **base_t** ∗)
  Remove an item by address.

Private Attributes

- std::string **id_string**
  identification string

Additional Inherited Members

4.103.1 Detailed Description

A **parser_t** (p. 220) instance is a node in the configuration tree. A parser node can contain any number of other **parser_t** (p. 220) instances or configuration language variables. These items are inserted into a parser node using the **parser_t::insert_item** (p. 222) method.

4.103.2 Constructor & Destructor Documentation

4.103.2.1 MHAParser::parser_t::parser_t (const std::string & *help_text* = "")

Parameters

help_text	A text describing this node. E.g. if this node lives at the root of some openMHA plugin, then the help text should describe the functionality of the plugin.

4.103.3 Member Function Documentation

4.103.3.1 void MHAParser::parser_t::insert_item (const std::string & *n*, MHAParser::base_t ∗ *e*)

This function registers an item under a given name into this sub-parser and makes it accessible to the parser interface.

Parameters

n	Name of the item in the configuration tree
e	C++ pointer to the item instance. e can either point to a variable, to a monitor, or to another sub-parser.

4.103.3.2 void MHAParser::parser_t::remove_item (const std::string & *n*)

If the item does not exist, an error is being reported.

Parameters

n	Name of parser item to be removed from list.

4.103.3.3 void MHAParser::parser_t::force_remove_item (const std::string & *n*)

Non-existing items are ignored.

Parameters

n	Name of parser item to be removed from list.

4.103.3.4 void MHAParser::parser_t::remove_item (const base_t ∗ *addr*)

The item belonging to an address is being removed from the list of items.

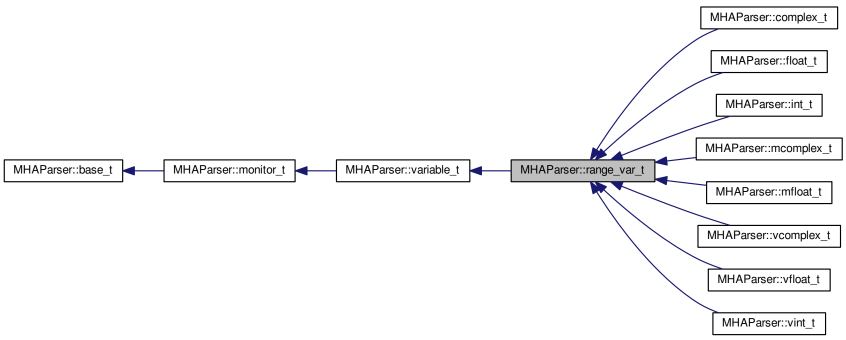
Parameters

addr	Address of parser item to be removed.

4.104 MHAParser::range_var_t Class Reference

Base class for all variables with a numeric value range.

Inheritance diagram for MHAParser::range_var_t:



Public Member Functions

- void **set_range** (const std::string &r)

  Change the valid range of a variable.

Protected Attributes

- float **low_limit**

  Lower limit of range.
- float **up_limit**

  Upper limit of range.
- bool **low_incl**

  Lower limit is included (or excluded) in range.
- bool **up_incl**

  Upper limit is included (or excluded) in range.
- bool **check_low**

  Check lower limit.
- bool **check_up**

  Check upper limit.
- bool **check_range**

  Range checking is active.

Additional Inherited Members

4.104.1 Member Function Documentation

4.104.1.1 void MHAParser::range_var_t::set_range (const std::string & *r*)

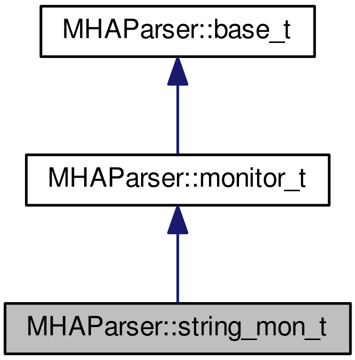
Parameters

r	New range of the variable (string representation)

4.105 MHAParser::string_mon_t Class Reference

Monitor with string value.

Inheritance diagram for MHAParser::string_mon_t:



Public Member Functions

- **string_mon_t** (const std::string &hlp)
  Create a monitor variable for string values.

Public Attributes

- std::string **data**
  Data field.

4.105.1 Constructor & Destructor Documentation

4.105.1.1 MHAParser::string_mon_t::string_mon_t (const std::string & *hlp*)

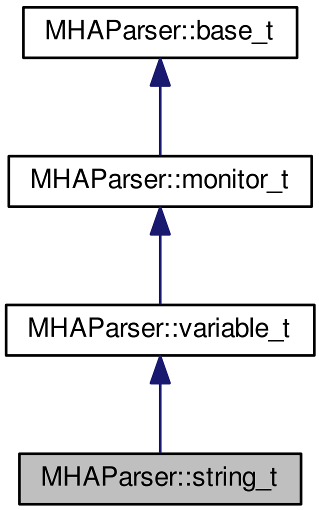
Parameters

hlp	A help text describing this monitor variable.

4.106 MHAParser::string_t Class Reference

Variable with a string value.

Inheritance diagram for MHAParser::string_t:



Public Member Functions

- **string_t** (const std::string &, const std::string &)
  Constructor of a openMHA configuration variable for string values.

Public Attributes

- std::string **data**
  Data field.

4.106.1 Constructor & Destructor Documentation

4.106.1.1 MHAParser::string_t::string_t (const std::string & *h*, const std::string & *v*)

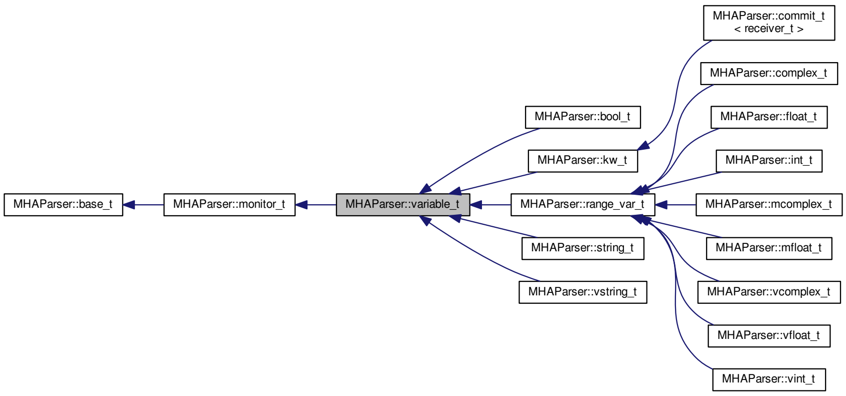
Parameters

h	A help string describing the purpose of this variable.
v	The initial string value

4.107 MHAParser::variable_t Class Reference

Base class for variable nodes.

Inheritance diagram for MHAParser::variable_t:



Public Member Functions

- void **setlock** (const bool &)

  Lock a variable against write access.

Additional Inherited Members

4.107.1 Member Function Documentation

4.107.1.1 void MHAParser::variable_t::setlock (const bool & *b*)

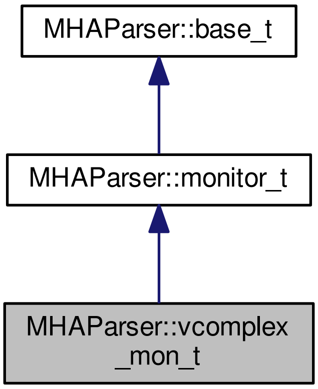
Parameters

b	Lock state

4.108 MHAParser::vcomplex_mon_t Class Reference

Monitor with vector of complex values.

Inheritance diagram for MHAParser::vcomplex_mon_t:



Public Member Functions

- **vcomplex_mon_t** (const std::string &hlp)
  Create a vector of complex monitor values.

Public Attributes

- std::vector< **mha_complex_t** > **data**
  Data field.

4.108.1 Constructor & Destructor Documentation

4.108.1.1 MHAParser::vcomplex_mon_t::vcomplex_mon_t (const std::string & *hlp*)

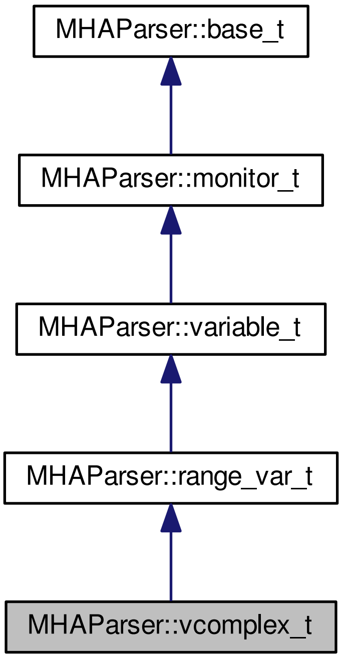
Parameters

hlp	A help text describing this monitor variable.

4.109 MHAParser::vcomplex_t Class Reference

Vector variable with complex value.

Inheritance diagram for MHAParser::vcomplex_t:



Public Attributes

- std::vector< **mha_complex_t** > **data**

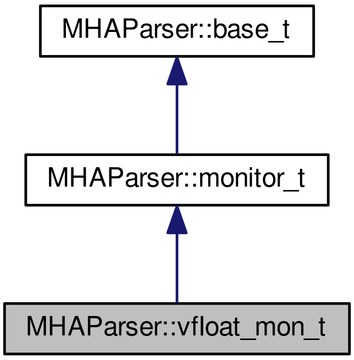
  Data field.

Additional Inherited Members

4.110 MHAParser::vfloat_mon_t Class Reference

Vector of floats monitor.

Inheritance diagram for MHAParser::vfloat_mon_t:



Public Member Functions

- **vfloat_mon_t** (const std::string &hlp)
  Create a vector of floating point monitor values.

Public Attributes

- std::vector< float > **data**
  Data field.

4.110.1 Constructor & Destructor Documentation

4.110.1.1 MHAParser::vfloat_mon_t::vfloat_mon_t (const std::string & *hlp*)

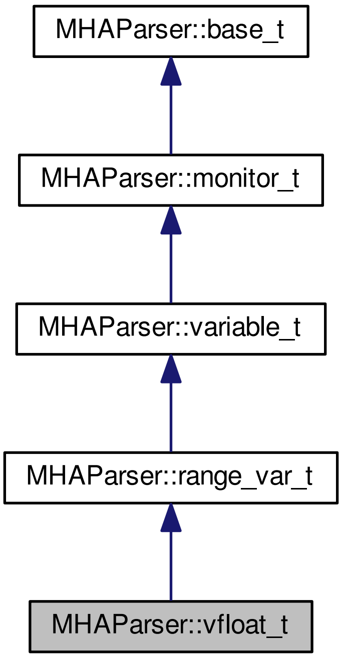
Parameters

hlp	A help text describing this monitor variable.

4.111 MHAParser::vfloat_t Class Reference

Vector variable with float value.

Inheritance diagram for MHAParser::vfloat_t:



Public Member Functions

- **vfloat_t** (const std::string &, const std::string &, const std::string &="")
  Create a float vector parser variable.

Public Attributes

- std::vector< float > **data**
  Data field.

Additional Inherited Members

4.111.1 Constructor & Destructor Documentation

4.111.1.1 MHAParser::vfloat_t::vfloat_t (const std::string & *h*, const std::string & *v*, const std::string & *rg* = "")

Parameters

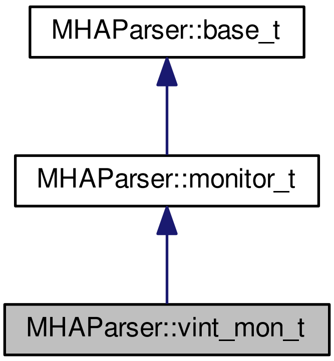
h	A human-readable text describing the purpose of this configuration variable.
v	The initial value of the variable, as a string, in openMHA configuration language: (e.g. "[0 1 2.1 3]" for a vector), described in the "Multidimensional Variables" s2.1.3 section of the openMHA User Manual.
rg	The numeric range to enforce on all members of the vector.

•

4.112 MHAParser::vint_mon_t Class Reference

Vector of ints monitor.

Inheritance diagram for MHAParser::vint_mon_t:



Public Member Functions

- **vint_mon_t** (const std::string &hlp)
  Create a vector of integer monitor values.

Public Attributes

- std::vector< int > **data**
  Data field.

4.112.1 Constructor & Destructor Documentation

4.112.1.1 MHAParser::vint_mon_t::vint_mon_t (const std::string & *hlp*)

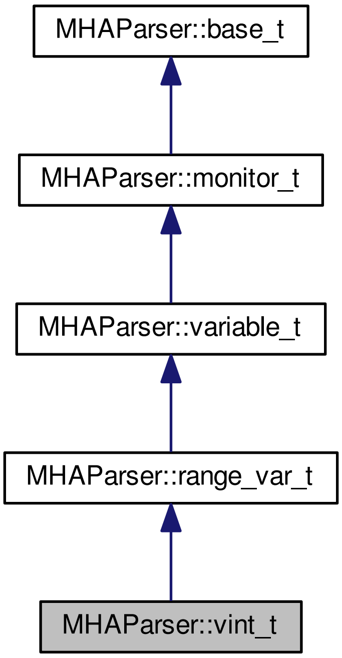
Parameters

hlp	A help text describing this monitor variable.

4.113 MHAParser::vint_t Class Reference

Variable with vector<int> value.

Inheritance diagram for MHAParser::vint_t:



Public Member Functions

- **vint_t** (const std::string &, const std::string &, const std::string &="")
  Constructor.

Public Attributes

- std::vector< int > **data**
  Data field.

Additional Inherited Members

4.113.1 Constructor & Destructor Documentation

4.113.1.1 MHAParser::vint_t::vint_t (const std::string & *h*, const std::string & *v*, const std::string & *rg* = "")

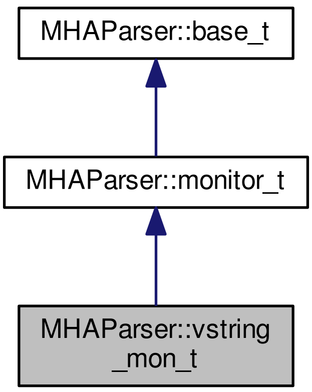
Parameters

h	help string
v	initial value
rg	optional: range constraint for all elements

4.114 MHAParser::vstring_mon_t Class Reference

Vector of monitors with string value.

Inheritance diagram for MHAParser::vstring_mon_t:



Public Member Functions

- **vstring_mon_t** (const std::string &hlp)
  Create a vector of string monitor values.

Public Attributes

- std::vector< std::string > **data**
  Data field.

4.114.1 Constructor & Destructor Documentation

4.114.1.1 MHAParser::vstring_mon_t::vstring_mon_t (const std::string & *hlp*)

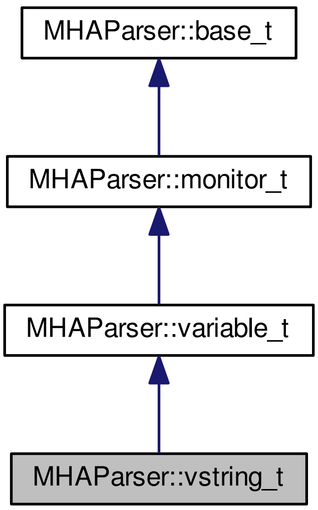
Parameters

hlp	A help text describing this monitor variable.

4.115 MHAParser::vstring_t Class Reference

Vector variable with string values.

Inheritance diagram for MHAParser::vstring_t:



Public Attributes

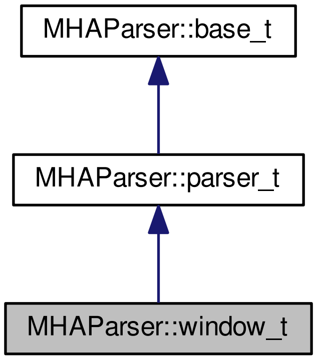
- std::vector< std::string > **data**
  Data field.

Additional Inherited Members

4.116 MHAParser::window_t Class Reference

MHA configuration interface for a window function generator.

Inheritance diagram for MHAParser::window_t:



Public Member Functions

- **window_t** (const std::string &help="Window type configuration.")
  Constructor to create parser class.
- **MHAWindow::base_t get_window** (unsigned int len) const
  Create a window instance, use default parameters.
- **MHAWindow::base_t get_window** (unsigned int len, float xmin) const
  Create a window instance.
- **MHAWindow::base_t get_window** (unsigned int len, float xmin, float xmax) const
  Create a window instance.
- **MHAWindow::base_t get_window** (unsigned int len, float xmin, float xmax, bool minincluded) const
  Create a window instance.
- **MHAWindow::base_t get_window** (unsigned int len, float xmin, float xmax, bool minincluded, bool maxincluded) const
  Create a window instance.
- MHAParser::window_t::wtype_t **get_type** () const
  Return currently selected window type.

Additional Inherited Members

4.116.1 Detailed Description

This class implements a configuration interface (sub-parser) for window type selection and user-defined window type. It provides member functions to generate an instance of **MHA←Window::base_t** (p. 278) based on the values provided by the configuration interface.

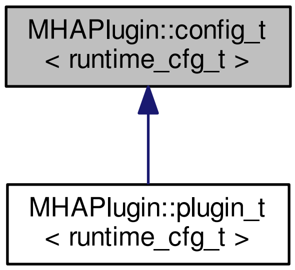
The configuration interface is derived from **MHAParser::parser_t** (p. 220) and can thus be inserted into the configuration tree using the **insert_item()** (p. 222) method of the parent parser.

If one of the pre-defined window types is used, then the window is generated using the **MHA↩ Window::fun_t** (p. 280) class constructor; for the user-defined type the values from the "user" variable are copied.

4.117 MHAPlugin::config_t< runtime_cfg_t > Class Template Reference

Template class for thread safe configuration.

Inheritance diagram for MHAPlugin::config_t< runtime_cfg_t >:



Protected Member Functions

- runtime_cfg_t ∗ **poll_config** ()

    Receive the latest run time configuration.
- runtime_cfg_t ∗ **last_config** ()

    Receive the latest run time configuration.
- void **push_config** (runtime_cfg_t ∗ncfg)

    Push a new run time configuration into the configuration fifo.

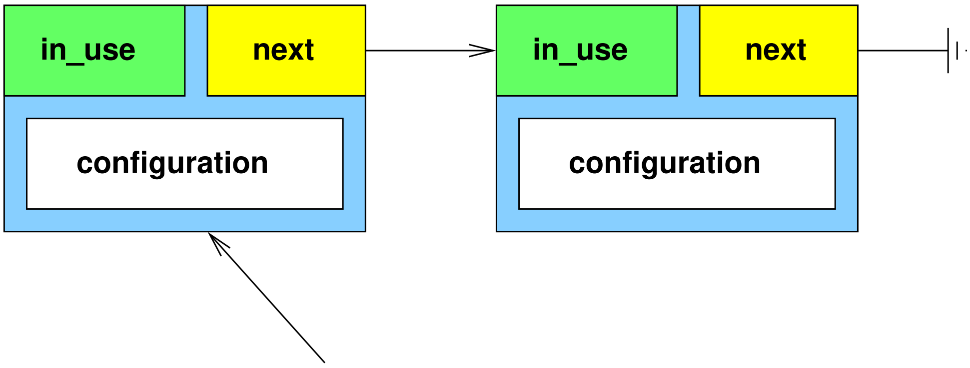
4.117.1 Detailed Description

template<class runtime_cfg_t>
class MHAPlugin::config_t< runtime_cfg_t >

This template class provides a mechanism for the handling of thread safe configuration which is required for run time configuration changes of the openMHA plugins.

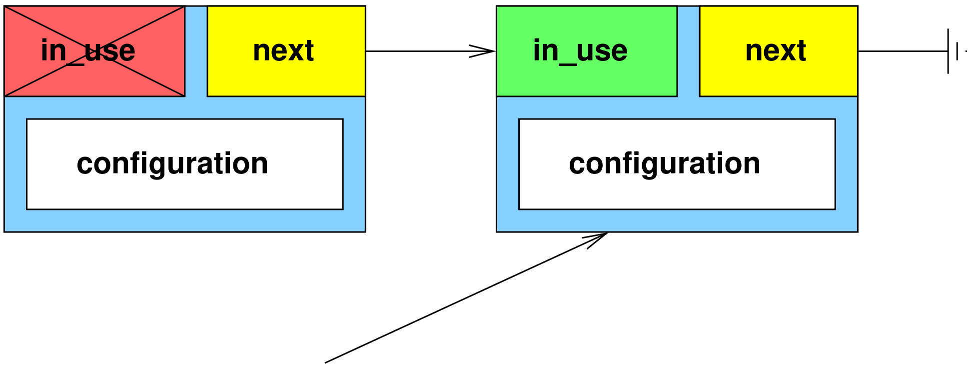
The template parameter runtime_cfg_t is the run time configuration class of the openMHA plugin. The constructor of that class should transform the **MHAParser** (p. 76) variables into derived runtime configuration. The constructor should fail if the configuration is invalid by any reason.

A new runtime configuration is provided by the function **push_config()** (p. 238). In the processing thread, the actual configuration can be received by a call of **poll_config()** (p. 237).



current configuration

Figure 5 Schematic drawing of runtime configuration update: configuration updated, but not used yet.



current configuration

Figure 6 Schematic drawing of runtime configuration update: configuration in use.

4.117.2 Member Function Documentation

4.117.2.1 template<class runtime_cfg_t > runtime_cfg_t ∗ MHAPlugin::config_t< runtime_cfg_t >::poll_config ( ) [protected]

This function stores the latest run time configuration into the protected class member variable 'cfg'. If no configuration exists, then an exception will be thrown. If no changes occured, then the value of 'cfg' will be untouched. This function should be called before any access to the 'cfg' variable, typically once in each signal processing call.

This function should be only called from the *processing* thread.

Exceptions

MHA_Error *(p. 132)*	if the resulting runtime configuration is NULL. This usually means that no push_config has occured.

4.117.2.2 template< class runtime_cfg_t > runtime_cfg_t * MHAPlugin::config_t< runtime_cfg_t >::last_config ( ) [protected]

This function stores the latest run time configuration into the protected class member variable 'cfg'. If no configuration exists, then an exception will be thrown. If no changes occured, then the value of 'cfg' will be untouched. This function may be called instead of poll_config.

The difference between poll_config and last_config is that poll_config marks previous configurations as ready for deletion, while this function does not. Therefore, memory usage of all runtime configurations will accumulate if only this function is called, but it enables safe access to previous runtime configurations.

Also, last_config does not raise an Exception when the latest run time configuration is NULL.

4.117.2.3 template< class runtime_cfg_t> void MHAPlugin::config_t< runtime_cfg_t >::push_config (runtime_cfg_t * *ncfg*) [protected]

This function adds a new run time configuration. The next time **poll_config** (p. 237) is called, this configuration will be available. Configurations which are not in use or are outdated will be removed.

This function should be only called from the *configuration* thread.

Parameters

ncfg	pointer on a new configuration

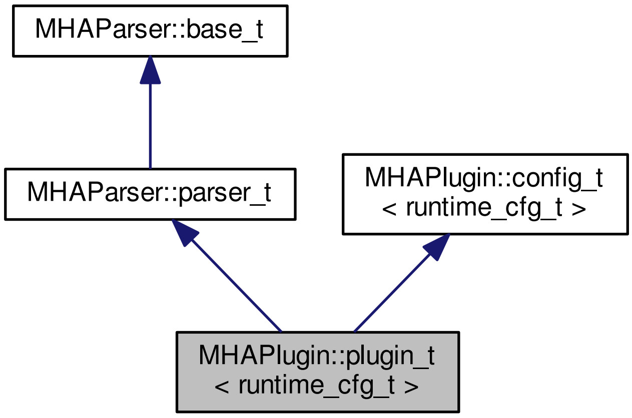
Warning

The runtime configuration passed to this function will be removed by the internal garbage collector. Do not free manually.

4.118 MHAPlugin::plugin_t< runtime_cfg_t > Class Template Reference

The template class for C++ openMHA plugins.

Inheritance diagram for MHAPlugin::plugin_t< runtime_cfg_t >:



Public Member Functions

- **plugin_t** (const std::string &, const **algo_comm_t** &)
  Constructor of plugin template.
- bool **is_prepared** () const
  Flag, if the prepare method is successfully called (or currently evaluated)
- **mhaconfig_t input_cfg** () const
  Current input channel configuration.
- **mhaconfig_t output_cfg** () const
  Current output channel configuration.

Protected Attributes

- **mhaconfig_t tftype**
  Member for storage of plugin interface configuration.
- **algo_comm_t ac**
  AC handle of the chain.

Additional Inherited Members

4.118.1 Detailed Description

template<class runtime_cfg_t>
class MHAPlugin::plugin_t< runtime_cfg_t >

Template Parameters

runtime_↩ cfg_t	run-time configuration.

This template class provides thread safe configuration handling and standard methods to be compatible to the C++ openMHA plugin wrapper macro **MHAPLUGIN_CALLBACKS** (p. 9).

The template parameter runtime_cfg_t should be the runtime configuration of the plugin.

See **MHAPlugin::config_t** (p. 236) for details on the thread safe communication update mechanism.

4.118.2 Constructor & Destructor Documentation

4.118.2.1 template< class runtime_cfg_t > MHAPlugin::plugin_t< runtime_cfg_t >::plugin_t (const std::string & *help*, const algo_comm_t & *iac*)

Parameters

help	Help comment to provide some general information about the plugin.
iac	AC space handle (will be stored into the member variable ac).

4.118.3 Member Data Documentation

4.118.3.1 template< class runtime_cfg_t > mhaconfig_t MHAPlugin::plugin_t< runtime_cfg_t >::tftype [protected]

This member is defined for convenience of the developer. Typically, the actual contents of **mhaconfig_t** (p. 155) are stored in this member in the prepare() method.

Note

This member is likely to be removed in later versions, use **input_cfg()** (p. 239) and **output_cfg()** (p. 239) instead.

4.118.3.2 template< class runtime_cfg_t > algo_comm_t MHAPlugin::plugin_t< runtime_cfg_t >::ac [protected]

This variable is initialized in the constructor and can be used by derived plugins to access the AC space. Its contents should not be modified.

4.119 mhaserver_t Class Reference

MHA Framework listening on TCP port for commands.

Inherits fw_t.

Public Member Functions

- **mhaserver_t** (const std::string &ao, const std::string &af, const std::string &lf)
- virtual std::string **received_group** (const std::string &line)

  A line of text was received from network client.
- virtual void **acceptor_started** (int status)

  Notification: "TCP port is open".
- virtual void **set_announce_port** (unsigned short announce_port)

  If set to nonzero, the spawning process has asked to be notified of the TCP port used by this process.
- void **logstring** (const std::string &)

  Log a message to log file.
- int **run** (unsigned short port, const std::string &_interface)

  Accept network connections and act on commands.

4.119.1 Constructor & Destructor Documentation

4.119.1.1 mhaserver_t::mhaserver_t (const std::string & *ao*, const std::string & *af*, const std::string & *lf*)

Parameters

ao	Acknowledgement string at end of successful command responses
af	Achknoledgement string at end of failed command responses
lf	File system path of file to use as log file. MHA appends.

4.119.2 Member Function Documentation

4.119.2.1 void mhaserver_t::set_announce_port (unsigned short *announce_port*) [virtual]

4.119.2.2 int mhaserver_t::run (unsigned short *port*, const std::string & *_interface*)

Calls **acceptor_started()** (p. 241) when the TCP port is opened. Calls received_group for every line received.

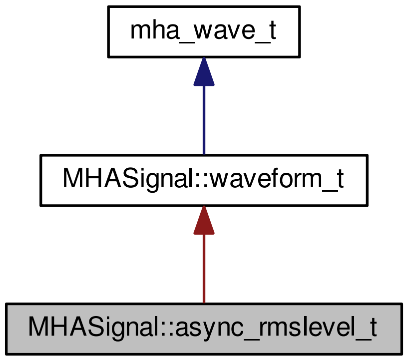
Returns

exit code that can be used as process exit code

4.120 MHASignal::async_rmslevel_t Class Reference

Class for asynchronous level metering.

Inheritance diagram for MHASignal::async_rmslevel_t:



Public Member Functions

- **async_rmslevel_t** (unsigned int frames, unsigned int **channels**)
  Constructor for level metering class.
- std::vector< float > **rmslevel** () const
  Read-only function for querying the current RMS level.
- std::vector< float > **peaklevel** () const
  Read-only function for querying the current peak level.
- void **process** (**mha_wave_t** ∗s)
  Function to store a chunk of audio in the level meter.

Additional Inherited Members

4.120.1 Detailed Description

4.120.2 Constructor & Destructor Documentation

4.120.2.1 MHASignal::async_rmslevel_t::async_rmslevel_t (unsigned int *frames,* unsigned int *channels*)

Allocate memory for metering. The RMS integration time corresponds to the number of frames in the buffer.

Parameters

frames	Number of frames to integrate.
channels	Number of channels used for level-metering.

4.120.3 Member Function Documentation

4.120.3.1 std::vector< float > MHASignal::async_rmslevel_t::rmslevel ( ) const

Returns

Vector of floats, one value for each channel, containing the RMS level in dB (SPL if calibrated properly).

4.120.3.2 std::vector< float > MHASignal::async_rmslevel_t::peaklevel ( ) const

Returns

Vector of floats, one value for each channel, containing the peak level in dB (SPL if calibrated properly).

4.120.3.3 void MHASignal::async_rmslevel_t::process (mha_wave_t ∗ *s*)

Parameters

s	Audio chunk (same number of channels required as given in the constructor).

4.121 MHASignal::delay_t Class Reference

Class to realize a simple delay of waveform streams.

Public Member Functions

- **delay_t** (std::vector< int > delays, unsigned int channels)
  Constructor.
- **mha_wave_t** ∗ **process** (**mha_wave_t** ∗s)
  Processing method.

4.121.1 Constructor & Destructor Documentation

4.121.1.1 MHASignal::delay_t::delay_t (std::vector< int > *delays,* unsigned int *channels*)

Parameters

delays	Vector of delays, one entry for each channel.
channels	Number of channels expected.

4.121.2 Member Function Documentation

4.121.2.1 mha_wave_t ∗ MHASignal::delay_t::process (mha_wave_t ∗ *s*)

Parameters

s	Input waveform fragment, with number of channels provided in constructor.

Returns

Output waveform fragment.

4.122 MHASignal::delay_wave_t Class Reference

Delayline containing wave fragments.

4.122.1 Detailed Description

The delayline contains waveform fragments. The delay can be configured in integer fragments (sample delay or sub-sample delay is not possible).

4.123 MHASignal::doublebuffer_t Class Reference

Double-buffering class.

Public Member Functions

- **doublebuffer_t** (unsigned int nchannels_in, unsigned int nchannels_out, unsigned int outer_fragsize, unsigned int inner_fragsize)

  Constructor of double buffer.
- **mha_wave_t** ∗ **outer_process** (**mha_wave_t** ∗s)

  Method to pass audio fragments into the inner layer.

Protected Member Functions

- virtual **mha_wave_t** ∗ **inner_process** (**mha_wave_t** ∗s)=0

  Method to realize inner processing callback.

4.123.1 Detailed Description

This class has two layers: The outer layer, with an outer fragment size, and an inner layer, with its own fragment size. Data is passed into the inner layer through the doublebuffer_t↩ ::outr_process() callback. The pure virtual method **doublebuffer_t::inner_process()** (p. 245) is called whenever enough data is available.

4.123.2 Constructor & Destructor Documentation

4.123.2.1 MHASignal::doublebuffer_t::doublebuffer_t (unsigned int *nchannels_in,* unsigned int *nchannels_out,* unsigned int *outer_fragsize,* unsigned int *inner_fragsize*)

Parameters

nchannels_in	Number of channels at the input (both layers).
nchannels_out	Number of channels at the output (both layers).
outer_fragsize	Fragment size of the outer layer (e.g., hardware fragment size)
inner_fragsize	Fragment size of the inner layer (e.g., software fragment size)

4.123.3 Member Function Documentation

4.123.3.1 mha_wave_t ∗ MHASignal::doublebuffer_t::outer_process (mha_wave_t ∗ *s*)

Parameters

s	Pointer to input waveform fragment.

Returns

Pointer to output waveform fragment.

4.123.3.2 virtual mha_wave_t∗ MHASignal::doublebuffer_t::inner_process (mha_wave_t ∗ *s*) [protected], [pure virtual]

To be overwritten by derived classes.

Parameters

s	Pointer to input waveform fragment.

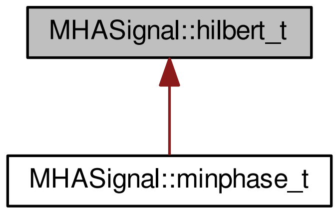
Returns

Pointer to output waveform fragment.

4.124 MHASignal::hilbert_t Class Reference

Hilbert transformation of a waveform segment.

Inheritance diagram for MHASignal::hilbert_t:



Public Member Functions

- **hilbert_t** (unsigned int len)
- void **operator()** (const **mha_wave_t** ∗, **mha_wave_t** ∗)

  Apply Hilbert transformation on a waveform segment.

4.124.1 Detailed Description

Returns the imaginary part of the inverse Fourier transformation of the Fourier transformed input signal with negative frequencies set to zero.

4.124.2 Constructor & Destructor Documentation

4.124.2.1 MHASignal::hilbert_t::hilbert_t (unsigned int *len*)

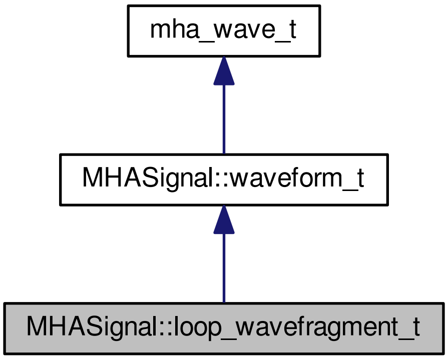
Parameters

len	Length of waveform segment

4.125 MHASignal::loop_wavefragment_t Class Reference

Copy a fixed waveform fragment to a series of waveform fragments of other size.

Inheritance diagram for MHASignal::loop_wavefragment_t:



Public Types

Public Member Functions

- **loop_wavefragment_t** (const **mha_wave_t** &src, bool loop, **level_mode_t** level_mode, std::vector< int > **channels**, unsigned int startpos=0)

  Constructor to create an instance of ***loop_wavefragment_t*** *(p. 247) based on an existing waveform block.*
- void **playback** (**mha_wave_t** ∗s, **playback_mode_t** pmode, **mha_wave_t** ∗level_pa, const std::vector< int > &**channels**)

  Add source waveform block to an output block.
- void **playback** (**mha_wave_t** ∗s, **playback_mode_t** pmode, **mha_wave_t** ∗level_pa)

  Add source waveform block to an output block.
- void **playback** (**mha_wave_t** ∗s, **playback_mode_t** pmode)

  Add source waveform block to an output block.

Additional Inherited Members

4.125.1 Detailed Description

This class is designed to continously play back a waveform to an output stream, with variable output block size.

4.125.2 Member Enumeration Documentation

4.125.2.1 enum **MHASignal::loop_wavefragment_t::level_mode_t**

Enumerator

relative The nominal level is applied as a gain to the source signal.

peak The nominal level is the peak level of source signal in Pascal.

rms The nominal level is the RMS level of the source signal in Pascal.

4.125.2.2 enum **MHASignal::loop_wavefragment_t::playback_mode_t**

Enumerator

add Add source signal to output stream.

replace Replace output stream by source signal.

input Do nothing, keep output stream (source position is unchanged).

mute Mute output stream (source position is unchanged).

4.125.3 Constructor & Destructor Documentation

4.125.3.1 MHASignal::loop_wavefragment_t::loop_wavefragment_t (const **mha_wave_t** & *src,* bool *loop,* **level_mode_t** *level_mode,* std::vector< int > *channels,* unsigned int *startpos* = 0)

Parameters

src	Waveform block to copy data from.
loop	Flag whether the block should be looped or played once.
level_mode	Configuration of playback level (see **MHASignal::loop_wavefragment_t::level_mode_t** (p. 248) for details)
channels	Mapping of input to output channels.
startpos	Starting position

4.125.4 Member Function Documentation

4.125.4.1 void MHASignal::loop_wavefragment_t::playback (**mha_wave_t** ∗ *s,* **playback_mode_t** *pmode,* **mha_wave_t** ∗ *level_pa,* const std::vector< int > & *channels*)

Parameters

s	Output block (streamed signal).
pmode	Playback mode (add, replace, input, mute).
level_pa	Linear output level/gain (depending on level_mode parameter in constructor); one value for each sample in output block.

Parameters

channels	Output channels

4.125.4.2 void MHASignal::loop_wavefragment_t::playback (mha_wave_t * *s*, playback_mode_t *pmode*, mha_wave_t * *level_pa*)

Parameters

s	Output block (streamed signal).
pmode	Playback mode (add, replace, input, mute).
level_pa	Linear output level/gain (depending on level_mode parameter in constructor); one value for each sample in output block.

4.125.4.3 void MHASignal::loop_wavefragment_t::playback (mha_wave_t * *s*, playback_mode_t *pmode*)

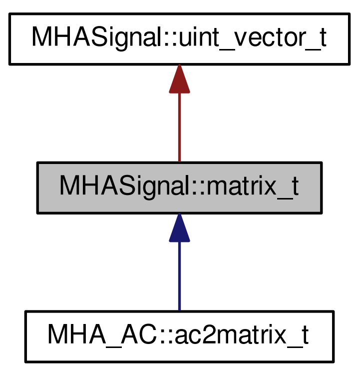
Parameters

s	Output block (streamed signal).
pmode	Playback mode (add, replace, input, mute).

4.126 MHASignal::matrix_t Class Reference

n-dimensional matrix with real or complex floating point values.

Inheritance diagram for MHASignal::matrix_t:



Public Member Functions

- **matrix_t** (unsigned int nrows, unsigned int ncols, bool b_is_complex=true)
  Create a two-dimensional matrix.
- **matrix_t** (const **mha_spec_t** &spec)
  Create a two-dimensional matrix from a spectrum, copy values.
- **matrix_t** (const **MHASignal::uint_vector_t** &**size**, bool b_is_complex=true)
  Create n-dimensional matrix, descriped by size argument.
- **matrix_t** (const uint8_t ∗buf, unsigned int len)
  Construct from memory area.
- **MHASignal::matrix_t** & **operator=** (const **comm_var_t** &v)
  Fill matrix with data of an AC variable object.
- **comm_var_t get_comm_var** ()
  Return a AC communication variable pointing to the data of the current matrix.
- unsigned int **dimension** () const
  Return the dimension of the matrix.
- unsigned int **size** (unsigned int k) const
  Return the size of the matrix.
- unsigned int **get_nelements** () const
  Return total number of elements.
- bool **is_same_size** (const **MHASignal::matrix_t** &)
  Test if matrix has same size as other.
- bool **iscomplex** () const
  Return information about complexity.
- **mha_real_t** & **real** (const **MHASignal::uint_vector_t** &index)
  Access real part of an element in a n-dimensional matrix.
- **mha_real_t** & **imag** (const **MHASignal::uint_vector_t** &index)
  Access imaginary part of an element in a n-dimensional matrix.
- **mha_complex_t** & **operator()** (const **MHASignal::uint_vector_t** &index)
  Access complex value of an element in a n-dimensional matrix.
- const **mha_real_t** & **real** (const **MHASignal::uint_vector_t** &index) const
  Access real part of an element in a n-dimensional matrix.
- const **mha_real_t** & **imag** (const **MHASignal::uint_vector_t** &index) const
  Access imaginary part of an element in a n-dimensional matrix.
- const **mha_complex_t** & **operator()** (const **MHASignal::uint_vector_t** &index) const
  Access complex value of an element in a n-dimensional matrix.
- **mha_real_t** & **real** (unsigned int row, unsigned int col)
  Access real part of an element in a two-dimensional matrix.
- **mha_real_t** & **imag** (unsigned int row, unsigned int col)
  Access imaginary part of an element in a two-dimensional matrix.
- **mha_complex_t** & **operator()** (unsigned int row, unsigned int col)
  Access complex value of an element in a two-dimensional matrix.
- const **mha_real_t** & **real** (unsigned int row, unsigned int col) const
  Access real part of an element in a two-dimensional matrix.
- const **mha_real_t** & **imag** (unsigned int row, unsigned int col) const

Access imaginary part of an element in a two-dimensional matrix.

- const **mha_complex_t** & **operator()** (unsigned int row, unsigned int col) const

  Access complex value of an element in a two-dimensional matrix.
- unsigned int **numbytes** () const

  Return number of bytes needed to store into memory.
- unsigned int **write** (uint8_t *buf, unsigned int len) const

  Copy to memory area.
- const **mha_real_t** * **get_rdata** () const

  Return pointer of real data.
- const **mha_complex_t** * **get_cdata** () const

  Return pointer of complex data.

Additional Inherited Members

4.126.1 Detailed Description

Warning

The member functions **imag()** (p. 253) and operator() should only be called if the matrix is defined to hold complex values.

4.126.2 Constructor & Destructor Documentation

4.126.2.1 MHASignal::matrix_t::matrix_t (unsigned int *nrows*, unsigned int *ncols*, bool *b_is_complex* = `true`)

Parameters

nrows	Number of rows
ncols	Number of columns
b_is_complex	Add space for complex values

4.126.2.2 MHASignal::matrix_t::matrix_t (const mha_spec_t & *spec*)

Parameters

spec	Source spectrum structure

4.126.2.3 MHASignal::matrix_t::matrix_t (const MHASignal::uint_vector_t & *size*, bool *b_is_complex* = `true`)

Parameters

size	Size vector
b_is_complex	Add space for complex values

4.126.2.4 MHASignal::matrix_t::matrix_t (const uint8_t * *buf,* unsigned int *len*)

Warning

This constructor is not real time safe

4.126.3 Member Function Documentation

4.126.3.1 MHASignal::matrix_t & MHASignal::matrix_t::operator= (const comm_var_t & *v*)

Parameters

v	Source AC variable (**comm_var_t** (p. 95))

Note

The type and dimension of the AC variable must match the type and dimension of the matrix.

4.126.3.2 comm_var_t MHASignal::matrix_t::get_comm_var ( )

Returns

AC variable object (**comm_var_t** (p. 95)), valid for the life time of the matrix.

4.126.3.3 unsigned int MHASignal::matrix_t::dimension ( ) const [inline]

Returns

Dimension of the matrix

4.126.3.4 unsigned int MHASignal::matrix_t::size (unsigned int *k*) const [inline]

Parameters

k	Dimension

Returns

Size of the matrix in dimension k

4.126.3.5 mha_real_t& MHASignal::matrix_t::real (const MHASignal::uint_vector_t & *index*) [inline]

Parameters

index	Index vector

4.126.3.6 mha_real_t& MHASignal::matrix_t::imag (const MHASignal::uint_vector_t & *index*) [inline]

Parameters

index	Index vector

4.126.3.7 mha_complex_t& MHASignal::matrix_t::operator() (const MHASignal::uint_vector_t & *index*) [inline]

Parameters

index	Index vector

4.126.3.8 const mha_real_t& MHASignal::matrix_t::real (const MHASignal::uint_vector_t & *index*) const [inline]

Parameters

index	Index vector

4.126.3.9 const mha_real_t& MHASignal::matrix_t::imag (const MHASignal::uint_vector_t & *index*) const [inline]

Parameters

index	Index vector

4.126.3.10 const mha_complex_t& MHASignal::matrix_t::operator() (const MHASignal::uint_vector_t & *index*) const [inline]

Parameters

index	Index vector

4.126.3.11 mha_real_t& MHASignal::matrix_t::real (unsigned int *row*, unsigned int *col*) [inline]

Parameters

row	Row number of element
col	Column number of element

4.126.3.12 mha_real_t& MHASignal::matrix_t::imag (unsigned int *row*, unsigned int *col*) [inline]

Parameters

row	Row number of element
col	Column number of element

4.126.3.13 mha_complex_t& MHASignal::matrix_t::operator() (unsigned int *row*, unsigned int *col*) [inline]

Parameters

row	Row number of element
col	Column number of element

4.126.3.14 const mha_real_t& MHASignal::matrix_t::real (unsigned int *row*, unsigned int *col*) const [inline]

Parameters

row	Row number of element
col	Column number of element

4.126.3.15 const mha_real_t& MHASignal::matrix_t::imag (unsigned int *row*, unsigned int *col*) const [inline]

Parameters

row	Row number of element
col	Column number of element

4.126.3.16 const **mha_complex_t**& MHASignal::matrix_t::operator() (unsigned int *row*, unsigned int *col*) const [inline]

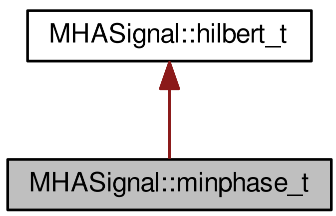
Parameters

row	Row number of element
col	Column number of element

4.127 MHASignal::minphase_t Class Reference

Minimal phase function.

Inheritance diagram for MHASignal::minphase_t:



Public Member Functions

- **minphase_t** (unsigned int fftlen, unsigned int ch)
  Constructor.
- void **operator()** (**mha_spec_t** ∗s)
  Transform input spectrum to a minimal-phase spectrum, discarding the original phase.

Additional Inherited Members

4.127.1 Detailed Description

The output spectrum $Y(f)$ is
$$Y(f) = |X(f)|e^{i\mathcal{H}\{\log|X(f)|\}},$$
with the input spectrum $X(f)$ and the Hilbert transformation $\mathcal{H}\{\cdots\}$.

4.127.2 Constructor & Destructor Documentation

4.127.2.1 MHASignal::minphase_t::minphase_t (unsigned int *fftlen*, unsigned int *ch*)

Parameters

fftlen	FFT length
ch	Number of channels

4.127.3 Member Function Documentation

4.127.3.1 void MHASignal::minphase_t::operator() (mha_spec_t * *s*)

Parameters

s	Spectrum to operate on.

4.128 MHASignal::quantizer_t Class Reference

Simple simulation of fixpoint quantization.

Public Member Functions

- **quantizer_t** (unsigned int num_bits)
  Constructor.
- void **operator()** (**mha_wave_t** &s)
  Quantization of a waveform fragment.

4.128.1 Detailed Description

4.128.2 Constructor & Destructor Documentation

4.128.2.1 MHASignal::quantizer_t::quantizer_t (unsigned int *num_bits*)

Parameters

num_bits	Number of bits to simulate, or zero for limiting to [-1,1] only.

4.128.3 Member Function Documentation

4.128.3.1 void MHASignal::quantizer_t::operator() (mha_wave_t & *s*)

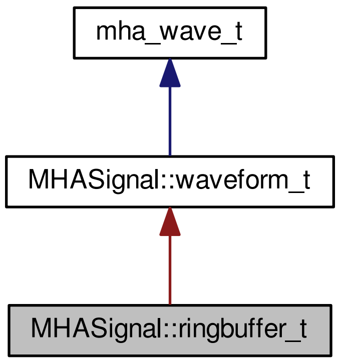
Parameters

s	Waveform fragment to be quantized.

4.129 MHASignal::ringbuffer_t Class Reference

A ringbuffer class for time domain audio signal, which makes no assumptions with respect to fragment size.

Inheritance diagram for MHASignal::ringbuffer_t:



Public Member Functions

- **ringbuffer_t** (unsigned frames, unsigned **channels**, unsigned prefilled_frames)
  Creates new ringbuffer.
- unsigned **contained_frames** () const
  number of currently contained frames
- **mha_real_t** & **value** (unsigned frame, unsigned channel)
  Access to value stored in ringbuffer.
- void **discard** (unsigned frames)
  *Discards the oldest frames Makes room for new **write** (p. 259), alters base frame index for **value** (p. 258).*
- void **write** (**mha_wave_t** &signal)
  Copies the contents of the signal into the ringbuffer if there is space.

Private Attributes

- unsigned **next_read_frame_index**
  identifies place with oldest frame in ringbuffer
- unsigned **next_write_frame_index**
  identifies place to store next frame in ringbuffer

Additional Inherited Members

4.129.1 Detailed Description

4.129.2 Constructor & Destructor Documentation

4.129.2.1 ringbuffer_t::ringbuffer_t (unsigned *frames*, unsigned *channels*, unsigned *prefilled_frames*)

Parameters

frames	Size of ringbuffer. Maximum frames can be stored in ringbuffer.
channels	Number of audio channels.
prefilled_frames	Number of frames to be prefilled with zero values

Exceptions

MHA_Error *(p. 132)*	if prefilled_frames > frames

4.129.3 Member Function Documentation

4.129.3.1 mha_real_t& MHASignal::ringbuffer_t::value (unsigned *frame*, unsigned *channel*) [inline]

Parameters

frame	frame index, 0 corresponds to oldest frame stored.
channel	audio channel

Returns

reference to contained sample value

Exceptions

MHA_Error *(p. 132)*	if channel or frame out of bounds.

4.129.3.2 void MHASignal::ringbuffer_t::discard (unsigned *frames*) [inline]

Parameters

frames	how many frames to discard.

Exceptions

MHA_Error *(p. 132)*	if frames > **contained_frames** (p. 257)

4.129.3.3 void MHASignal::ringbuffer_t::write (mha_wave_t & *signal*) [inline]

Parameters

signal	New signal to be appended to the signal already present in the ringbuffer

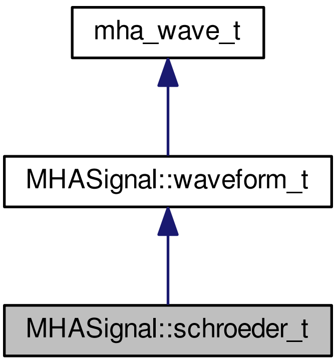
Exceptions

MHA_Error *(p. 132)*	if there is not enough space or if the channel count mismatches.

4.130 MHASignal::schroeder_t Class Reference

Schroeder tone complex class.

Inheritance diagram for MHASignal::schroeder_t:



Public Types

- typedef float(∗ **groupdelay_t**) (float f, float fmin, float fmax)

  Function type for group delay definition.

Public Member Functions

- **schroeder_t** (unsigned int len, unsigned int **channels**=1, **schroeder_t::sign_t** sign=**up**, **mha_real_t** speed=1)

  Constructor.
- **schroeder_t** (unsigned int len, unsigned int **channels**=1, **schroeder_t::groupdelay_t** freqfun=MHASignal::schroeder_t::identity, float fmin=0, float fmax=1, float eps=1e-10)

  Construct create Schroeder tone complex from a given frequency function.

Additional Inherited Members

4.130.1 Detailed Description

The Schroeder tone complex is a sweep defined in the sampled spectrum:

$$\Phi(f) = \sigma 2\pi\tau(2f/f_s)^{2\alpha}, \quad S(f) = e^{i\Phi(f)}$$

f is the sampled frequency in Hz, σ is the sign of the sweep (-1 for up sweep, +1 for down sweep), τ is the sweep duration in samples, f_s is the sampling rate in Hz and α is the relative sweep speed.

4.130.2 Member Typedef Documentation

4.130.2.1 typedef float(∗ MHASignal::schroeder_t::groupdelay_t) (float f, float fmin, float fmax)

Parameters

f	Frequency relative to Nyquist frequency.
fmin	Minimum frequency relative to Nyquist frequency.
fmax	Maximum frequency relative to Nyquist frequency.

4.130.3 Member Enumeration Documentation

4.130.3.1 enum MHASignal::schroeder_t::sign_t

Enumerator

up Sweep from zero to Nyquist frequency ($\sigma = -1$)

down Sweep from Nyquist frequency to zero ($\sigma = +1$)

4.130.4 Constructor & Destructor Documentation

4.130.4.1 MHASignal::schroeder_t::schroeder_t (unsigned int *len*, unsigned int *channels* = 1, schroeder_t::sign_t *sign* = up, mha_real_t *speed* = 1)

Parameters of the Schroeder tone complex are configured in the constructor.

Parameters

len	Length τ of the Schroeder tone complex in samples
channels	Number of channels
sign	Sign σ of Schroeder sweep
speed	Relative speed α (curvature of phase function)

4.130.4.2 MHASignal::schroeder_t::schroeder_t (unsigned int *len*, unsigned int *channels* = 1, schroeder_t::groupdelay_t *freqfun* = MHASignal::schroeder_t::identity, float *fmin* = 0, float *fmax* = 1, float *eps* = 1e-10)

The frequency function $g(f)$ defines the sweep speed and sign (based on the group delay). It must be defined in the interval [0,1) and should return values in the interval [0,1].

$$\Phi(f) = -4\pi\tau \int_0^\tau g(f)\,\mathrm{d}f, \quad S(f) = e^{i\Phi(f)}$$

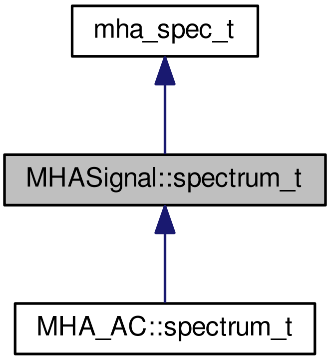
Parameters

len	Length τ of the Schroeder tone complex in samples.
channels	Number of channels.
freqfun	Frequency function $g(f)$.
fmin	Start frequency (relative to Nyquist frequency).
fmax	End frequency (relative to Nyquist frequency).
eps	Stability constant for frequency ranges not covered by Schroeder tone complex.

4.131 MHASignal::spectrum_t Class Reference

a signal processing class for spectral data (based on **mha_spec_t** (p. 141))

Inheritance diagram for MHASignal::spectrum_t:



Public Member Functions

- **spectrum_t** (const unsigned int &frames, const unsigned int &**channels**)

  constructor of spectrum class
- **spectrum_t** (const **mha_spec_t** &)

  Copy constructor.
- **spectrum_t** (const **MHASignal::spectrum_t** &)

  Copy constructor.
- **mha_complex_t** & **operator()** (unsigned int f, unsigned int ch)

  Access to element.
- **mha_complex_t** & **operator[ ]** (unsigned int k)

  Access to a single element, direct index into data buffer.
- **mha_complex_t** & **value** (unsigned int f, unsigned int ch)

  Access to element.
- void **copy** (const **mha_spec_t** &)

  copy all elements from a spectrum
- void **copy_channel** (const **mha_spec_t** &s, unsigned sch, unsigned dch)

  Copy one channel of a given spectrum signal to a target channel.
- void **export_to** (**mha_spec_t** &)

  copy elements to spectrum structure
- void **scale** (const unsigned int &, const unsigned int &, const unsigned int &, const **mha↩_real_t** &)

  scale section [a,b) in channel "ch" by "val"
- void **scale_channel** (const unsigned int &, const **mha_real_t** &)

  scale all elements in one channel

Additional Inherited Members

4.131.1 Constructor & Destructor Documentation

4.131.1.1 spectrum_t::spectrum_t (const unsigned int & *frames*, const unsigned int & *channels*)

Allocates buffers and initializes memory to zeros.

Parameters

frames	number of frames (fft bins) in one channel. Number of Frames is usually fftlen / 2 + 1
channels	number of channels

4.131.2 Member Function Documentation

4.131.2.1 mha_complex_t& MHASignal::spectrum_t::operator() (unsigned int *f*, unsigned int *ch*) [inline]

Parameters

f	Bin number
ch	Channel number

Returns

Reference to element

4.131.2.2 mha_complex_t& MHASignal::spectrum_t::operator[ ] (unsigned int *k*) [inline]

Parameters

k	Buffer index

Returns

Reference to element

4.131.2.3 mha_complex_t& MHASignal::spectrum_t::value (unsigned int *f*, unsigned int *ch*) [inline]

Parameters

f	Bin number
ch	Channel number

Returns

Reference to element

4.131.2.4 void spectrum_t::copy (const mha_spec_t & *src*)

Parameters

src	input spectrum

4.131.2.5 void spectrum_t::copy_channel (const mha_spec_t & *s,* unsigned *sch,* unsigned *dch*)

Parameters

s	Input spectrum signal
sch	Channel index in source signal
dch	Channel index in destination (this) signal

4.131.2.6 void spectrum_t::export_to (mha_spec_t & *dest*)

Parameters

dest	destination spectrum structure

4.131.2.7 void spectrum_t::scale (const unsigned int & *a,* const unsigned int & *b,* const unsigned int & *ch,* const mha_real_t & *val*)

Parameters

a	starting frame
b	end frame (excluded)
ch	channel number
val	scale factor

4.131.2.8 void spectrum_t::scale_channel (const unsigned int & *ch,* const mha_real_t & *src*)

Parameters

ch	channel number
src	scale factor

4.132 MHASignal::subsample_delay_t Class Reference

implements subsample delay in spectral domain.

Public Member Functions

- **subsample_delay_t** (const std::vector< float > &subsample_delay, unsigned fftlen)
  Constructor computes complex phase factors to apply to achieve subsample delay.
- void **process** (**mha_spec_t** ∗s)
  Apply the phase_gains to s to achieve the subsample delay.
- void **process** (**mha_spec_t** ∗s, unsigned idx)
  Apply the pase gains to channel idx in s to achieve the subsample delay in channel idx.

Public Attributes

- **spectrum_t phase_gains**
  The complex factors to apply to achieve the necessary phase shift.

Private Attributes

- unsigned **last_complex_bin**
  index of the last complex fft bin for the used fft length.

4.132.1 Detailed Description

When transformed back to the time domain, the signal is delayed by the configured fraction of a sample. This operation must not be used in a smoothgains bracket.

4.132.2 Constructor & Destructor Documentation

4.132.2.1 MHASignal::subsample_delay_t::subsample_delay_t (const std::vector< float > & *subsample_delay,* unsigned *fftlen*)

Parameters

subsample_delay	The subsample delay to apply. -0.5 <= subsample_delay <= 0.5
fftlen	FFT length

Exceptions

MHA_Error *(*p. *132)*	if the parameters are out of range

4.132.3 Member Function Documentation

4.132.3.1 void MHASignal::subsample_delay_t::process (mha_spec_t * *s*, unsigned *idx*)

Parameters

s	signal
idx	channel index, 0-based

Exceptions

MHA_Error *(*p. *132)*	if idx >= s->num_channels

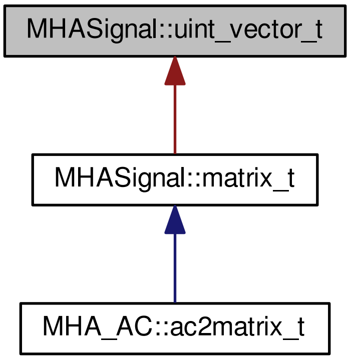
4.132.4 Member Data Documentation

4.132.4.1 unsigned MHASignal::subsample_delay_t::last_complex_bin [private]

4.133 MHASignal::uint_vector_t Class Reference

Vector of unsigned values, used for size and index description of n-dimensional matrixes.

Inheritance diagram for MHASignal::uint_vector_t:



Public Member Functions

- **uint_vector_t** (unsigned int len)
  Constructor, initializes all elements to zero.
- **uint_vector_t** (const uint8_t ∗buf, unsigned int len)
  Construct from memory area.
- bool **operator==** (const **uint_vector_t** &) const
  Check for equality.
- **uint_vector_t** & **operator=** (const **uint_vector_t** &)
  Assign from other ***uint_vector_t*** *(p. 266).*
- unsigned int **get_length** () const
  Return the length of the vector.
- const uint32_t & **operator[ ]** (unsigned int k) const
  Read-only access to elements.
- uint32_t & **operator[ ]** (unsigned int k)
  Access to elements.
- unsigned int **numbytes** () const
  Return number of bytes needed to store into memory.
- unsigned int **write** (uint8_t ∗buf, unsigned int len) const
  Copy to memory area.
- const uint32_t ∗ **getdata** () const
  Return pointer to the data field.

4.133.1 Constructor & Destructor Documentation

4.133.1.1 MHASignal::uint_vector_t::uint_vector_t (unsigned int *len*)

Parameters

len	Length of vector.

4.133.1.2 MHASignal::uint_vector_t::uint_vector_t (const uint8_t ∗ *buf*, unsigned int *len*)

Warning

This constructor is not real time safe

4.133.2 Member Function Documentation

4.133.2.1 uint_vector_t & MHASignal::uint_vector_t::operator= (const uint_vector_t & *src*)

Warning

This assignment will fail if the lengths mismatch.

4.133.2.2 unsigned int MHASignal::uint_vector_t::numbytes ( ) const

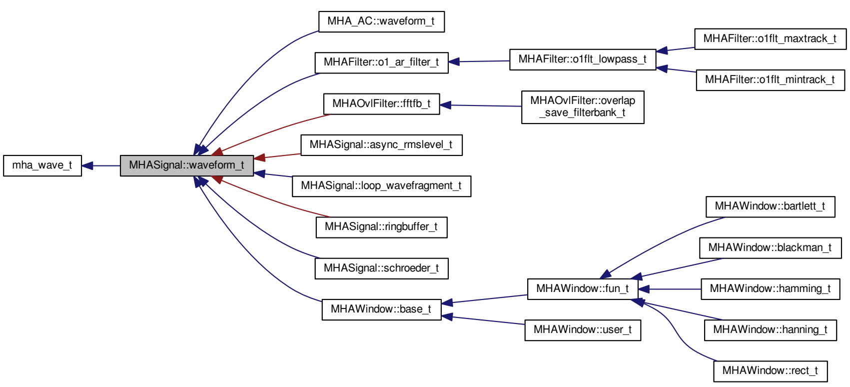
4.133.2.3 unsigned int MHASignal::uint_vector_t::write (uint8_t ∗ *buf*, unsigned int *len*) const

4.133.2.4 const uint32_t∗ MHASignal::uint_vector_t::getdata ( ) const [inline]

4.134 MHASignal::waveform_t Class Reference

signal processing class for waveform data (based on **mha_wave_t** (p. 154))

Inheritance diagram for MHASignal::waveform_t:



Public Member Functions

- **waveform_t** (const unsigned int &frames, const unsigned int &**channels**)
  *constructor of **waveform_t** (p. 268)*
- **waveform_t** (const **mhaconfig_t** &cf)
  Constructor to create a waveform from plugin configuration.
- **waveform_t** (const **mha_wave_t** &src)
  *Copy contructor for **mha_wave_t** (p. 154) source.*
- **waveform_t** (const **MHASignal::waveform_t** &src)
  Copy contructor.
- **waveform_t** (const std::vector< **mha_real_t** > &src)
  Copy contructor for std::vector<mha_real_t> source.
- **mha_real_t** & **value** (unsigned int t, unsigned int ch)
  Element accessor.
- **mha_real_t** & **operator()** (unsigned int t, unsigned int ch)
  Element accessor.
- const **mha_real_t** & **value** (unsigned int t, unsigned int ch) const
  Constant element accessor.
- const **mha_real_t** & **operator()** (unsigned int t, unsigned int ch) const
  Constant element accessor.
- **mha_real_t sum** (const unsigned int &a, const unsigned int &b)
  sum of all elements between [a,b) in all channels
- **mha_real_t sum** (const unsigned int &a, const unsigned int &b, const unsigned int &ch)
  sum of all elements between [a,b) in channel ch
- **mha_real_t sum** ()
  sum of all elements
- **mha_real_t sumsqr** ()
  sum of square of all elements
- **mha_real_t sum_channel** (const unsigned int &)
  return sum of all elements in one channel
- void **assign** (const unsigned int &k, const unsigned int &ch, const **mha_real_t** &val)
  set frame "k" in channel "ch" to value "val"
- void **assign** (const **mha_real_t** &)
  set all elements to value
- void **assign_frame** (const unsigned int &k, const **mha_real_t** &val)
  assign value "val" to frame k in all channels
- void **assign_channel** (const unsigned int &c, const **mha_real_t** &val)
  assign value "val" to channel ch in all frames
- void **copy** (const **mha_wave_t** &)
  copy data from source into current waveform
- void **copy_channel** (const **mha_wave_t** &, unsigned int, unsigned int)
  Copy one channel of a given waveform signal to a target channel.
- void **copy_from_at** (unsigned int, unsigned int, const **mha_wave_t** &, unsigned int)
  Copy part of the source signal into part of this waveform object.
- void **export_to** (**mha_wave_t** &)

*copy data into allocated **mha_wave_t** (p. 154) structure*

- void **limit** (const **mha_real_t** &**min**, const **mha_real_t** &**max**)

    limit target to range [min,max]

- void **power** (const **waveform_t** &)

    transform waveform signal (in Pa) to squared signal (in W/m^2)

- void **powspec** (const **mha_spec_t** &)

    get the power spectrum (in W/m^2) from a complex spectrum

- void **scale** (const unsigned int &a, const unsigned int &b, const unsigned int &ch, const **mha_real_t** &val)

    scale section [a,b) in channel "ch" by "val"

- void **scale** (const unsigned int &k, const unsigned int &ch, const **mha_real_t** &val)

    scale one element

- void **scale_channel** (const unsigned int &, const **mha_real_t** &)

    scale one channel of target with a scalar

Additional Inherited Members

4.134.1 Constructor & Destructor Documentation

4.134.1.1 waveform_t::waveform_t (const unsigned int & *frames*, const unsigned int & *channels*)

Allocates buffer memory and initializes values to zero.

Parameters

frames	number of frames in each channel
channels	number of channels

4.134.1.2 waveform_t::waveform_t (const **mhaconfig_t** & *cf*) [explicit]

Parameters

cf	Plugin configuration

4.134.1.3 waveform_t::waveform_t (const std::vector< **mha_real_t** > & *src*)

A waveform structure with a single channel is created, the length is equal to the number of elements in the source vector.

4.134.2 Member Function Documentation

4.134.2.1 **mha_real_t**& MHASignal::waveform_t::value (unsigned int *t*, unsigned int *ch*) [inline]

Parameters

t	Frame number
ch	Channel number

Returns

Reference to element

4.134.2.2 mha_real_t& MHASignal::waveform_t::operator() (unsigned int *t*, unsigned int *ch*) [inline]

Parameters

t	Frame number
ch	Channel number

Returns

Reference to element

4.134.2.3 const mha_real_t& MHASignal::waveform_t::value (unsigned int *t*, unsigned int *ch*) const [inline]

Parameters

t	Frame number
ch	Channel number

Returns

Reference to element

4.134.2.4 const mha_real_t& MHASignal::waveform_t::operator() (unsigned int *t*, unsigned int *ch*) const [inline]

Parameters

t	Frame number
ch	Channel number

Returns

Reference to element

4.134.2.5 mha_real_t waveform_t::sum (const unsigned int & *a*, const unsigned int & *b*)

Parameters

a	starting frame
b	end frame (excluded)

Returns

sum

4.134.2.6 mha_real_t waveform_t::sum (const unsigned int & *a*, const unsigned int & *b*, const unsigned int & *ch*)

Parameters

a	starting frame
b	end frame (exluded)
ch	channel number

Returns

sum

4.134.2.7 mha_real_t waveform_t::sum ( )

Returns

sum of all elements

4.134.2.8 mha_real_t waveform_t::sumsqr ( )

Returns

sum of square of all elements

4.134.2.9 mha_real_t waveform_t::sum_channel (const unsigned int & *ch*)

Parameters

ch	channel number

Returns

sum

4.134.2.10 void waveform_t::assign (const unsigned int & *k*, const unsigned int & *ch*, const mha_real_t & *val*)

Parameters

k	frame number
ch	channel number
val	new value

4.134.2.11 void waveform_t::assign (const mha_real_t & *val*)

Parameters

val	new value

4.134.2.12 void waveform_t::assign_frame (const unsigned int & *k*, const mha_real_t & *val*)

Parameters

k	frame number
val	new value

4.134.2.13 void waveform_t::assign_channel (const unsigned int & *ch*, const mha_real_t & *val*)

Parameters

ch	channel number
val	new value

4.134.2.14 void waveform_t::copy (const mha_wave_t & *src*)

Parameters

src	input data (need to be same size as target)

4.134.2.15 void waveform_t::copy_channel (const mha_wave_t & *src*, unsigned int *src_channel*, unsigned int *dest_channel*)

Parameters

src	Input waveform signal
src_channel	Channel in source signal
dest_channel	Channel number in destination signal

4.134.2.16 void waveform_t::copy_from_at (unsigned int *to_pos,* unsigned int *len,* const mha_wave_t & *src,* unsigned int *from_pos*)

Source and target have to have the same number of channels.

Parameters

to_pos	Offset in target
len	Number of frames copied
src	Source
from_pos	Offset in source

4.134.2.17 void waveform_t::export_to (mha_wave_t & *dest*)

Parameters

dest	destination structure

4.134.2.18 void waveform_t::limit (const mha_real_t & *min,* const mha_real_t & *max*)

Parameters

min	lower limit
max	upper limit

4.134.2.19 void waveform_t::power (const waveform_t & *src*)

Parameters

src	linear waveform signal (in Pa)

4.134.2.20 void waveform_t::powspec (const mha_spec_t & *src*)

Parameters

src	complex spectrum (normalized to Pa)

4.134.2.21 void waveform_t::scale (const unsigned int & *a*, const unsigned int & *b*, const unsigned int & *ch*, const **mha_real_t** & *val*)

Parameters

a	starting frame
b	end frame (excluded)
ch	channel number
val	scale factor

4.134.2.22 void waveform_t::scale (const unsigned int & *k*, const unsigned int & *ch*, const **mha_real_t** & *val*)

Parameters

k	frame number
ch	channel number
val	scale factor

4.134.2.23 void waveform_t::scale_channel (const unsigned int & *ch*, const **mha_real_t** & *src*)

Parameters

ch	channel number
src	factor

4.135 MHATableLookup::xy_table_t Class Reference

Class for interpolation with non-equidistant x values.

Inherits MHATableLookup::table_t.

Inherited by MHAOvlFilter::barkscale::bark2hz_t, and MHAOvlFilter::barkscale::hz2bark_t.

Public Member Functions

- **mha_real_t lookup** (**mha_real_t** x)

  Return the y-value at the position of the nearest x value below input.
- **mha_real_t interp** (**mha_real_t** x)

  Linear interpolation function.
- void **add_entry** (**mha_real_t** x, **mha_real_t** y)

  Add a single x-y pair entry.
- void **add_entry** (**mha_real_t** ∗pVX, **mha_real_t** ∗pVY, unsigned int len)

Add multiple entries at once.

- void **clear** ()

  Clear the table and transformation functions.
- void **set_xfun** (float(∗pXFun)(float))

  Set transformation function for x values.
- void **set_yfun** (float(∗pYFun)(float))

  Set transformation function for y values during insertion.
- void **set_xyfun** (float(∗pYFun)(float, float))

  Set transformation function for y values during insertion, based on x and y values.

4.135.1 Detailed Description

Linear interpolation of the x-y table is performed. A transformation of x and y-values is possible; if a transformation function is provided for the x-values, the same function is applied to the argument of **xy_table_t::interp()** (p. 276) and **xy_table_t::lookup()** (p. 276). The transformation of y values is applied only during insertion into the table. Two functions for y-transformation can be provided: a simple transformation which depends only on the y values, or a transformation which takes both (non-transformed) x and y value as an argument. The two-argument transformation is applied before the one-argument transformation.

4.135.2 Member Function Documentation

4.135.2.1 mha_real_t xy_table_t::lookup (mha_real_t *x*)

Parameters

x	Input value

Returns

y value at nearest x value below input.

4.135.2.2 mha_real_t xy_table_t::interp (mha_real_t *x*)

Parameters

x	x value

Returns

interpolated y value

4.135.2.3 void xy_table_t::add_entry (mha_real_t *x*, mha_real_t *y*)

Parameters

x	x value
y	corresponding y value

4.135.2.4 void xy_table_t::add_entry (mha_real_t * *pVX*, mha_real_t * *pVY*, unsigned int *uLength*)

Parameters

pVX	array of x values
pVY	array of y values
uLength	Length of x and y arrays

4.135.2.5 void xy_table_t::set_xfun (float(*)(float) *fun*)

Parameters

fun	Transformation function.

4.135.2.6 void xy_table_t::set_yfun (float(*)(float) *fun*)

Parameters

fun	Transformation function.

4.135.2.7 void xy_table_t::set_xyfun (float(*)(float, float) *fun*)

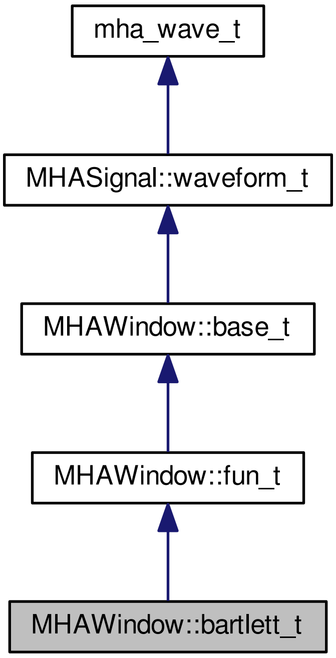
Parameters

fun	Transformation function.

4.136 MHAWindow::bartlett_t Class Reference

Bartlett window.

Inheritance diagram for MHAWindow::bartlett_t:

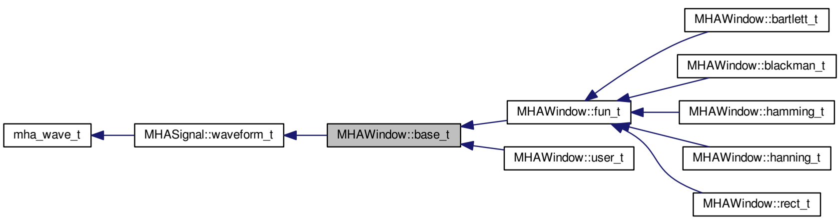


Additional Inherited Members

4.137 MHAWindow::base_t Class Reference

Common base for window types.

Inheritance diagram for MHAWindow::base_t:



Public Member Functions

- **base_t** (unsigned int len)
  Constructor.
- **base_t** (const **MHAWindow::base_t** &src)
  Copy constructor.
- void **operator()** (**mha_wave_t** &) const
  Apply window to waveform segment (reference)
- void **operator()** (**mha_wave_t** ∗) const
  Apply window to waveform segment (pointer)
- void **ramp_begin** (**mha_wave_t** &) const
  Apply a ramp at the begining.
- void **ramp_end** (**mha_wave_t** &) const
  Apply a ramp at the end.

Additional Inherited Members

4.137.1 Constructor & Destructor Documentation

4.137.1.1 MHAWindow::base_t::base_t (unsigned int *len*)

Parameters

len	Window length in samples.

4.137.1.2 MHAWindow::base_t::base_t (const MHAWindow::base_t & *src*)

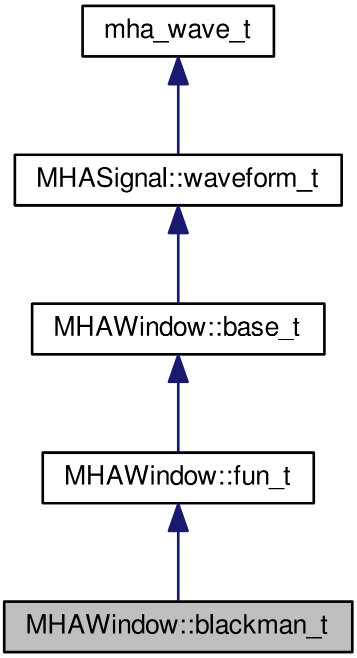
Parameters

src	Source to be copied

4.138 MHAWindow::blackman_t Class Reference

Blackman window.

Inheritance diagram for MHAWindow::blackman_t:

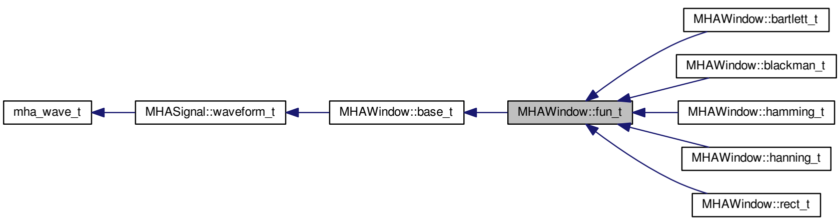


Additional Inherited Members

4.139 MHAWindow::fun_t Class Reference

Generic window based on a generator function.

Inheritance diagram for MHAWindow::fun_t:



Public Member Functions

- **fun_t** (unsigned int n, float(∗fun)(float), float xmin=-1, float xmax=1, bool min_↩ included=true, bool max_included=false)

    Constructor.

Additional Inherited Members

4.139.1 Detailed Description

The generator function should return a valid window function in the interval [-1,1[.

4.139.2 Constructor & Destructor Documentation

4.139.2.1 MHAWindow::fun_t::fun_t (unsigned int *n*, float(*)(float) *fun*, float *xmin* = `-1`, float *xmax* = `1`, bool *min_included* = `true`, bool *max_included* = `false`)

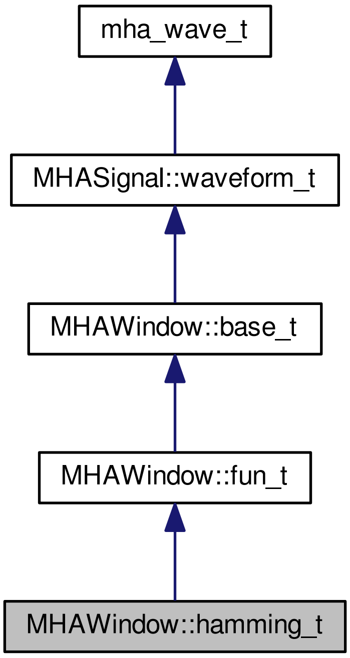
Parameters

n	Window length
fun	Generator function, i.e. **MHAWindow::hanning()** (p. 304)
xmin	Start value of window, i.e. -1 for full window or 0 for fade-out ramp.
xmax	Last value of window, i.e. 1 for full window
min_included	Flag if minimum value is included
max_included	Flag if maximum value is included

4.140 MHAWindow::hamming_t Class Reference

Hamming window.

Inheritance diagram for MHAWindow::hamming_t:

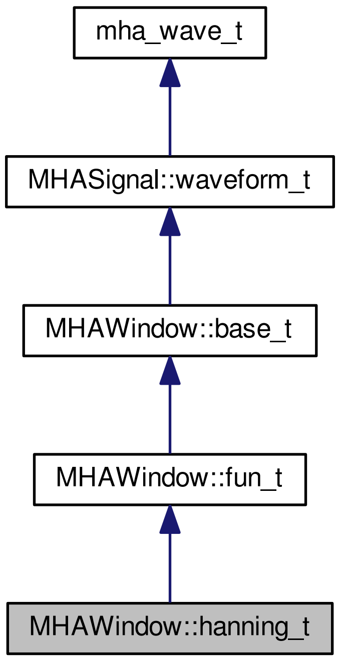


Additional Inherited Members

4.141 MHAWindow::hanning_t Class Reference

von-Hann window

Inheritance diagram for MHAWindow::hanning_t:

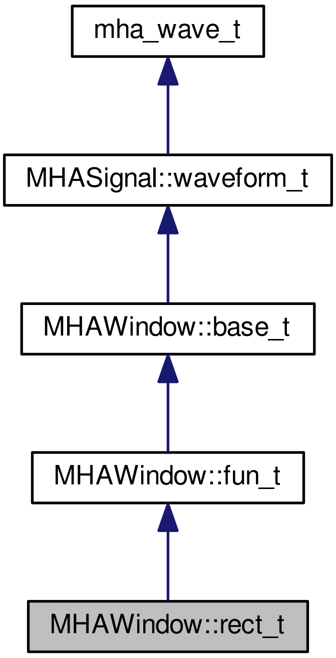


Additional Inherited Members

4.142 MHAWindow::rect_t Class Reference

Rectangular window.

Inheritance diagram for MHAWindow::rect_t:

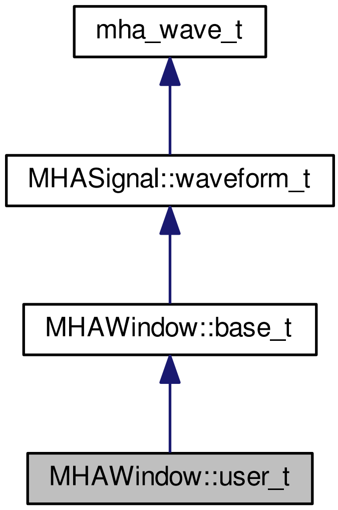


Additional Inherited Members

4.143 MHAWindow::user_t Class Reference

User defined window.

Inheritance diagram for MHAWindow::user_t:



Public Member Functions

- **user_t** (const std::vector< **mha_real_t** > &wnd)
  Constructor.

Additional Inherited Members

4.143.1 Constructor & Destructor Documentation

4.143.1.1 MHAWindow::user_t::user_t (const std::vector< mha_real_t > & *wnd*)

Parameters

wnd	User defined window

4.144 PluginLoader::fourway_processor_t Class Reference

This abstract class defines the interface for classes that implement all types of signal domain processing supported by the MHA: wave2wave, spec2spec, wave2spec, and spec2wave.

Inherited by PluginLoader::mhapluginloader_t.

Public Member Functions

- virtual void **process** (**mha_wave_t** ∗s_in, **mha_wave_t** ∗∗s_out)=0
  Pure waveform processing.
- virtual void **process** (**mha_spec_t** ∗s_in, **mha_spec_t** ∗∗s_out)=0
  Pure spectrum processing.
- virtual void **process** (**mha_wave_t** ∗s_in, **mha_spec_t** ∗∗s_out)=0
  Signal processing with domain transformation from waveform to spectrum.
- virtual void **process** (**mha_spec_t** ∗s_in, **mha_wave_t** ∗∗s_out)=0
  Signal processing with domain transformation from spectrum to waveform.
- virtual void **prepare** (**mhaconfig_t** &settings)=0
  Prepares the processor for signal processing.
- virtual void **release** ()=0
  Resources allocated for signal processing in ***fourway_processor_t::prepare*** *(p. 287) are released here in* ***fourway_processor_t::release*** *(p. 287).*
- virtual std::string **parse** (const std::string &query)=0
  Parser interface.
- virtual ∼**fourway_processor_t** ()
  Classes with virtual methods need virtual destructor.

4.144.1 Detailed Description

For supporting different output domains for the same input domain, the processing methods are overloaded with respect to input domain and output domain.

4.144.2 Constructor & Destructor Documentation

4.144.2.1 virtual PluginLoader::fourway_processor_t::∼fourway_processor_t ( ) [inline], [virtual]

This destructor is empty.

4.144.3 Member Function Documentation

4.144.3.1 virtual void PluginLoader::fourway_processor_t::process (mha_wave_t ∗ *s_in*, mha_wave_t ∗∗ *s_out*) [pure virtual]

Parameters

s_in	input waveform signal
s_out	output waveform signal

4.144.3.2 virtual void PluginLoader::fourway_processor_t::process (mha_spec_t * *s_in*, mha_spec_t ** *s_out*) [pure virtual]

Parameters

s_in	input spectrum signal
s_out	output spectrum signal

4.144.3.3 virtual void PluginLoader::fourway_processor_t::process (mha_wave_t * *s_in*, mha_spec_t ** *s_out*) [pure virtual]

Parameters

s_in	input waveform signal
s_out	output spectrum signal

4.144.3.4 virtual void PluginLoader::fourway_processor_t::process (mha_spec_t * *s_in*, mha_wave_t ** *s_out*) [pure virtual]

Parameters

s_in	input spectrum signal
s_out	output waveform signal

4.144.3.5 virtual void PluginLoader::fourway_processor_t::prepare (mhaconfig_t & *settings*) [pure virtual]

Parameters

settings	domain and dimensions of the signal. The contents of settings may be modified by the prepare implementation. Upon calling **fourway_processor_t::prepare** (p. 287), settings reflects domain and dimensions of the input signal. When **fourway_processor_t::prepare** (p. 287) returns, settings reflects domain and dimensions of the output signal.

4.144.3.6 virtual void PluginLoader::fourway_processor_t::release ( ) [pure virtual]

5 File Documentation

5.1 mha.h File Reference

common types for MHA kernel, MHA framework applications and external plugins

Classes

- struct **mha_complex_t**
  Type for complex floating point values.
- struct **mha_direction_t**
  Channel source direction structure.
- struct **mha_channel_info_t**
  Channel information structure.
- struct **mha_wave_t**
  Waveform signal structure.
- struct **mha_spec_t**
  Spectrum signal structure.
- struct **mha_audio_descriptor_t**
  *Description of an audio fragment (planned as a replacement of **mhaconfig_t** (p. 155)).*
- struct **mha_audio_t**
  *An audio fragment in the openMHA (planned as a replacement of **mha_wave_t** (p. 154) and **mha_spec_t** (p. 141)).*
- struct **mhaconfig_t**
  MHA prepare configuration structure.
- struct **comm_var_t**
  Algorithm communication variable structure.
- struct **algo_comm_t**
  A reference handle for algorithm communication variables.

Macros

- #define **MHA_CALLBACK_TEST**(x)
  Test macro to compare function type definition and declaration.
- #define **MHA_VERSION_MAJOR** 4
  Major version number of MHA.
- #define **MHA_VERSION_MINOR** 5
  Minor version number of MHA.
- #define **MHA_VERSION_RELEASE** 0
  Release number of MHA.
- #define **MHA_VERSION_BUILD** 0
  Build number of MHA (currently unused)
- #define **MHA_STRUCT_SIZEMATCH** (unsigned int)((sizeof(**mha_real_t**)==4)+2∗(sizeof(**mha_complex_t**)==8)+4∗(sizeof(**mha_wave_t**)==8+2∗sizeof(void∗))+8∗(sizeof(**mha_spec_t**)==8+2∗sizeof(void∗))+16∗(sizeof(**mhaconfig_t**)==24))
  Test number for structure sizes.
- #define **MHA_VERSION** (unsigned int)((**MHA_STRUCT_SIZEMATCH** | (**MHA_VERSION_RELEASE** << 8) | (**MHA_VERSION_MINOR** << 16) | (**MHA_VERSION_MAJOR** << 24)))
  Full version number of MHA kernel.
- #define **MHA_VERSION_STRING** MHA_XSTRF(**MHA_VERSION_MAJOR**) "." MHA_XSTRF(**MHA_VERSION_MINOR**)
  Version string of MHA kernel (major.minor)

Typedefs

- typedef float **mha_real_t**

  openMHA type for real numbers
- typedef void ∗ **mha_fft_t**

  Handle for an FFT object.

5.2 mha_algo_comm.h File Reference

Header file for Algorithm Communication.

Classes

- class **MHA_AC::spectrum_t**

  Insert a ***MHASignal::spectrum_t*** *(*p. *261) class into the AC space.*
- class **MHA_AC::waveform_t**

  Insert a ***MHASignal::waveform_t*** *(*p. *268) class into the AC space.*
- class **MHA_AC::int_t**

  Insert a integer variable into the AC space.
- class **MHA_AC::float_t**

  Insert a float point variable into the AC space.
- class **MHA_AC::double_t**

  Insert a double precision floating point variable into the AC space.
- class **MHA_AC::ac2matrix_t**

  Copy AC variable to a matrix.
- class **MHA_AC::acspace2matrix_t**

  Copy all or a subset of all numeric AC variables into an array of matrixes.

Namespaces

- **MHA_AC**

  Functions and classes for Algorithm Communication (AC) support.

Functions

- **mha_spec_t MHA_AC::get_var_spectrum** (**algo_comm_t** ac, const std::string &name)

  Convert an AC variable into a spectrum.
- **mha_wave_t MHA_AC::get_var_waveform** (**algo_comm_t** ac, const std::string &name)

  Convert an AC variable into a waveform.
- int **MHA_AC::get_var_int** (**algo_comm_t** ac, const std::string &name)

  Return value of an integer scalar AC variable.
- float **MHA_AC::get_var_float** (**algo_comm_t** ac, const std::string &name)

  Return value of an floating point scalar AC variable.
- std::vector< float > **MHA_AC::get_var_vfloat** (**algo_comm_t** ac, const std::string &name)

  Return value of an floating point vector AC variable as standard vector of floats.

5.3 mha_defs.h File Reference

Preprocessor definitions common to all MHA components.

Macros

- #define **M_PI** 3.14159265358979323846
  Define pi if it is not defined yet.
- #define **MIN**(a, b) (((a)<(b))?(a):(b))
  Macro for minimum function.
- #define **MAX**(a, b) (((a)>(b))?(a):(b))
  Macro for maximum function.

5.3.1 Detailed Description

This file contains all preprocessor and type definitions which are common to all Master Hearing Aid components.

5.4 mha_error.cpp File Reference

Implementation of openMHA error handling.

Functions

- static unsigned **digits** (unsigned n)
  Compute number of digits in an unsigned integer.
- void **mha_debug** (const char ∗fmt,...)
  Print an info message (stderr on Linux, OutputDebugString in Windows).

5.4.1 Detailed Description

This file forms a seperate library.

5.5 mha_filter.hh File Reference

Header file for IIR filter classes.

Classes

- class **MHAFilter::filter_t**
  Generic IIR filter class.
- class **MHAFilter::diff_t**
  Differentiator class (non-normalized)
- class **MHAFilter::o1_ar_filter_t**
  First order attack-release lowpass filter.
- class **MHAFilter::o1flt_lowpass_t**
  First order low pass filter.
- class **MHAFilter::o1flt_maxtrack_t**
  First order maximum tracker.
- class **MHAFilter::o1flt_mintrack_t**
  First order minimum tracker.
- class **MHAFilter::iir_filter_t**
  IIR filter class wrapper for integration into parser structure.
- class **MHAFilter::adapt_filter_t**
  Adaptive filter.
- class **MHAFilter::fftfilter_t**
  FFT based FIR filter implementation.
- class **MHAFilter::fftfilterbank_t**
  FFT based FIR filterbank implementation.
- struct **MHAFilter::transfer_function_t**
  a structure containing a source channel number, a target channel number, and an impulse response.
- struct **MHAFilter::transfer_matrix_t**
  A sparse matrix of transfer function partitionss.
- class **MHAFilter::partitioned_convolution_t**
  A filter class for partitioned convolution.
- struct **MHAFilter::partitioned_convolution_t::index_t**
  Bookkeeping class.
- class **MHAFilter::smoothspec_t**
  Smooth spectral gains, create a windowed impulse response.
- class **MHAFilter::resampling_filter_t**
  Hann shaped low pass filter for resampling.
- class **MHAFilter::polyphase_resampling_t**
  A class that does polyphase resampling.
- class **MHAFilter::blockprocessing_polyphase_resampling_t**
  A class that does polyphase resampling and takes into account block processing.
- class **MHAFilter::iir_ord1_real_t**
  First order recursive filter.

Namespaces

- **MHAFilter**
  Namespace for IIR and FIR filter classes.

Functions

- void **MHAFilter::o1_lp_coeffs** (const **mha_real_t** tau, const **mha_real_t** fs, **mha_real_t** &c1, **mha_real_t** &c2)

  Set first order filter coefficients from time constant and sampling rate.
- void **MHAFilter::butter_stop_ord1** (double ∗A, double ∗B, double f1, double f2, double fs)

  Setup a first order butterworth band stop filter.
- **MHASignal::waveform_t** ∗ **MHAFilter::spec2fir** (const **mha_spec_t** ∗spec, const unsigned int fftlen, const **MHAWindow::base_t** &window, const bool minphase)

  Create a windowed impulse response/FIR filter coefficients from a spectrum.
- unsigned **MHAFilter::gcd** (unsigned a, unsigned b)

  greatest common divisor
- double **MHAFilter::sinc** (double x)

  sin(x)/x function, coping with x=0.
- std::pair< unsigned, unsigned > **MHAFilter::resampling_factors** (float source_sampling_rate, float target_sampling_rate, float factor=1.0f)

  Computes rational resampling factor from two sampling rates.

5.6 mha_parser.hh File Reference

Header file for the MHA-Parser script language.

Classes

- class **MHAParser::keyword_list_t**

  Keyword list class.
- class **MHAParser::base_t**

  Base class for all parser items.
- class **MHAParser::parser_t**

  Parser node class.
- class **MHAParser::monitor_t**

  Base class for monitors and variable nodes.
- class **MHAParser::variable_t**

  Base class for variable nodes.
- class **MHAParser::range_var_t**

  Base class for all variables with a numeric value range.
- class **MHAParser::kw_t**

  Variable with keyword list value.
- class **MHAParser::string_t**

  Variable with a string value.
- class **MHAParser::vstring_t**

  Vector variable with string values.
- class **MHAParser::bool_t**

Variable with a boolean value ("yes"/"no")

- class **MHAParser::int_t**

  Variable with integer value.

- class **MHAParser::float_t**

  Variable with float value.

- class **MHAParser::complex_t**

  Variable with complex value.

- class **MHAParser::vint_t**

  Variable with vector<int> value.

- class **MHAParser::vfloat_t**

  Vector variable with float value.

- class **MHAParser::vcomplex_t**

  Vector variable with complex value.

- class **MHAParser::mfloat_t**

  Matrix variable with float value.

- class **MHAParser::mcomplex_t**

  Matrix variable with complex value.

- class **MHAParser::int_mon_t**

  Monitor variable with int value.

- class **MHAParser::bool_mon_t**

  Monitor with string value.

- class **MHAParser::string_mon_t**

  Monitor with string value.

- class **MHAParser::vstring_mon_t**

  Vector of monitors with string value.

- class **MHAParser::vint_mon_t**

  Vector of ints monitor.

- class **MHAParser::vfloat_mon_t**

  Vector of floats monitor.

- class **MHAParser::mfloat_mon_t**

  Matrix of floats monitor.

- class **MHAParser::float_mon_t**

  Monitor with float value.

- class **MHAParser::complex_mon_t**

  Monitor with complex value.

- class **MHAParser::vcomplex_mon_t**

  Monitor with vector of complex values.

- class **MHAParser::mcomplex_mon_t**

  Matrix of complex numbers monitor.

- class **MHAParser::commit_t**< **receiver_t** >

  Parser variable with event-emission functionality.

Namespaces

- **MHAParser**
  Name space for the openMHA-Parser configuration language.
- **MHAParser::StrCnv**
  String converter namespace.

Macros

- #define **insert_member**(x) insert_item(#x,&x)
  Macro to insert a member variable into a parser.

Functions

- void **MHAParser::strreplace** (std::string &, const std::string &, const std::string &)
  string replace function
- void **MHAParser::StrCnv::str2val** (const std::string &, bool &)
  Convert from string.
- void **MHAParser::StrCnv::str2val** (const std::string &, float &)
  Convert from string.
- void **MHAParser::StrCnv::str2val** (const std::string &, **mha_complex_t** &)
  Convert from string.
- void **MHAParser::StrCnv::str2val** (const std::string &, int &)
  Convert from string.
- void **MHAParser::StrCnv::str2val** (const std::string &, keyword_list_t &)
  Convert from string.
- void **MHAParser::StrCnv::str2val** (const std::string &, std::string &)
  Convert from string.
- template< class arg_t >
  void **MHAParser::StrCnv::str2val** (const std::string &s, std::vector< arg_t > &val)
  Converter for vector types.
- template<>
  void **MHAParser::StrCnv::str2val< mha_real_t >** (const std::string &s, std::vector< **mha_real_t** > &v)
  Converter for vector<mha_real_t> with Matlab-style expansion.
- template< class arg_t >
  void **MHAParser::StrCnv::str2val** (const std::string &s, std::vector< std::vector< arg_t > > &val)
  Converter for matrix types.
- std::string **MHAParser::StrCnv::val2str** (const bool &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const float &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const **mha_complex_t** &)
  Convert to string.

- std::string **MHAParser::StrCnv::val2str** (const int &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const keyword_list_t &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::string &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< float > &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< **mha_complex_t** > &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< int > &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< std::string > &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< std::vector< float > > &)
  Convert to string.
- std::string **MHAParser::StrCnv::val2str** (const std::vector< std::vector< **mha_complex_t** > > &)
  Convert to string.

5.6.1 Macro Definition Documentation

5.6.1.1 #define insert_member(*x*) insert_item(#x,&x)

Parameters

x	Member variable to be inserted. Name of member variable will be used as configuration name.

See also **MHAParser::parser_t::insert_item()** (p. 222).

5.7 mha_plugin.hh File Reference

Header file for MHA C++ plugin class templates.

Classes

- class **MHAPlugin::config_t**< **runtime_cfg_t** >
  Template class for thread safe configuration.
- class **MHAPlugin::plugin_t**< **runtime_cfg_t** >
  The template class for C++ openMHA plugins.

Namespaces

- **MHAPlugin**

  Namespace for openMHA plugin class templates and thread-safe runtime configurations.

Macros

- #define **MHAPLUGIN_CALLBACKS_PREFIX**(prefix, classname, indom, outdom)

  C++ wrapper macro for the plugin interface.
- #define **MHAPLUGIN_DOCUMENTATION_PREFIX**(prefix, cat, doc)

  Wrapper macro for the plugin documentation interface.
- #define **MHAPLUGIN_CALLBACKS**(plugname, classname, indom, outdom) **MHAPLUGIN_CALLBACKS_PREFIX**(MHA_STATIC_ ## plugname ## _,classname,indom,outdom)

  C++ wrapper macro for the plugin interface.
- #define **MHAPLUGIN_DOCUMENTATION**(plugname, cat, doc) **MHAPLUGIN_DOCUMENTATION_PREFIX**(MHA_STATIC_ ## plugname ## _,cat,doc)

  Wrapper macro for the plugin documentation interface.

Functions

- **__attribute__** ((unused)) static const char ∗mha_git_commit_hash

  store git commit hash in every binary plgin to support reproducible research

5.7.1 Detailed Description

This file defines useful macros and template classes for the development of MHA plugins. A set of macros wraps a C++ interface around the ANSI-C plugin interface. The plugin_t template class defines a corresponding C++ class with all required members. This class can make use of thread safe configurations (config_t).

5.8 mha_signal.hh File Reference

Header file for audio signal handling and processing classes.

Classes

- class **MHASignal::spectrum_t**

  a signal processing class for spectral data (based on ***mha_spec_t*** *(p. 141))*
- class **MHASignal::waveform_t**

  signal processing class for waveform data (based on ***mha_wave_t*** *(p. 154))*
- class **MHASignal::doublebuffer_t**

  Double-buffering class.
- class **MHASignal::ringbuffer_t**

  A ringbuffer class for time domain audio signal, which makes no assumptions with respect to fragment size.
- class **MHASignal::hilbert_t**

  Hilbert transformation of a waveform segment.
- class **MHASignal::minphase_t**

  Minimal phase function.
- class **MHAWindow::base_t**

  Common base for window types.
- class **MHAWindow::fun_t**

  Generic window based on a generator function.
- class **MHAWindow::rect_t**

  Rectangular window.
- class **MHAWindow::bartlett_t**

  Bartlett window.
- class **MHAWindow::hanning_t**

  von-Hann window
- class **MHAWindow::hamming_t**

  Hamming window.
- class **MHAWindow::blackman_t**

  Blackman window.
- class **MHAWindow::user_t**

  User defined window.
- class **MHASignal::delay_wave_t**

  Delayline containing wave fragments.
- class **MHASignal::async_rmslevel_t**

  Class for asynchronous level metering.
- class **MHASignal::uint_vector_t**

  Vector of unsigned values, used for size and index description of n-dimensional matrixes.
- class **MHASignal::matrix_t**

  n-dimensional matrix with real or complex floating point values.
- class **MHASignal::schroeder_t**

  Schroeder tone complex class.
- class **MHASignal::quantizer_t**

  Simple simulation of fixpoint quantization.
- class **MHASignal::loop_wavefragment_t**

  Copy a fixed waveform fragment to a series of waveform fragments of other size.

- class **MHASignal::delay_t**

  Class to realize a simple delay of waveform streams.
- class **MHASignal::subsample_delay_t**

  implements subsample delay in spectral domain.

Namespaces

- **MHASignal**

  Namespace for audio signal handling and processing classes.
- **MHAWindow**

  Collection of Window types.

Functions

- void **MHASignal::for_each** (**mha_wave_t** ∗s, **mha_real_t**(∗fun)(**mha_real_t**))

  *Apply a function to each element of a **mha_wave_t** (p. 154).*
- **mha_real_t MHASignal::lin2db** (**mha_real_t** x)

  Conversion from linear scale to dB (no SPL reference)
- **mha_real_t MHASignal::db2lin** (**mha_real_t** x)

  Conversion from dB scale to linear (no SPL reference)
- **mha_real_t MHASignal::pa2dbspl** (**mha_real_t** x)

  Conversion from linear Pascal scale to dB SPL.
- **mha_real_t MHASignal::pa22dbspl** (**mha_real_t** x, **mha_real_t** eps=1e-20f)

  Conversion from squared Pascal scale to dB SPL.
- **mha_real_t MHASignal::dbspl2pa** (**mha_real_t** x)

  Conversion from dB SPL to linear Pascal scale.
- **mha_real_t MHASignal::smp2sec** (**mha_real_t** n, **mha_real_t** srate)

  conversion from samples to seconds
- **mha_real_t MHASignal::sec2smp** (**mha_real_t** sec, **mha_real_t** srate)

  conversion from seconds to samples
- **mha_real_t MHASignal::bin2freq** (**mha_real_t** bin, unsigned fftlen, **mha_real_t** srate)

  conversion from fft bin index to frequency
- **mha_real_t MHASignal::freq2bin** (**mha_real_t** freq, unsigned fftlen, **mha_real_t** srate)

  conversion from frequency to fft bin index
- **mha_real_t MHASignal::smp2rad** (**mha_real_t** samples, unsigned bin, unsigned fftlen)

  conversion from delay in samples to phase shift
- **mha_real_t MHASignal::rad2smp** (**mha_real_t** phase_shift, unsigned bin, unsigned fftlen)

  conversion from phase shift to delay in samples
- template< class elem_type >
  std::vector< elem_type > **MHASignal::dupvec** (std::vector< elem_type > vec, unsigned n)

  Duplicate last vector element to match desired size.

- template<class elem_type >
  std::vector< elem_type > **MHASignal::dupvec_chk** (std::vector< elem_type > vec, unsigned n)

  Duplicate last vector element to match desired size, check for dimension.
- bool **equal_dim** (const **mha_wave_t** &a, const **mha_wave_t** &b)

  Test for equal dimension of waveform structures.
- bool **equal_dim** (const **mha_wave_t** &a, const **mhaconfig_t** &b)

  Test for match of waveform dimension with mhaconfig structure.
- bool **equal_dim** (const **mha_spec_t** &a, const **mha_spec_t** &b)

  Test for equal dimension of spectrum structures.
- bool **equal_dim** (const **mha_spec_t** &a, const **mhaconfig_t** &b)

  Test for match of spectrum dimension with mhaconfig structure.
- bool **equal_dim** (const **mha_wave_t** &a, const **mha_spec_t** &b)

  Test for equal dimension of waveform/spectrum structures.
- bool **equal_dim** (const **mha_spec_t** &a, const **mha_wave_t** &b)

  Test for equal dimension of waveform/spectrum structures.
- void **integrate** (**mha_wave_t** &s)

  Numeric integration of a signal vector (real values)
- void **integrate** (**mha_spec_t** &s)

  Numeric integration of a signal vector (complex values)
- unsigned int **size** (const **mha_wave_t** &s)

  Return size of a waveform structure.
- unsigned int **size** (const **mha_spec_t** &s)

  Return size of a spectrum structure.
- unsigned int **size** (const **mha_wave_t** ∗s)

  Return size of a waveform structure.
- unsigned int **size** (const **mha_spec_t** ∗s)

  Return size of a spectrum structure.
- void **clear** (**mha_wave_t** &s)

  Set all values of waveform to zero.
- void **clear** (**mha_wave_t** ∗s)

  Set all values of waveform to zero.
- void **clear** (**mha_spec_t** &s)

  Set all values of spectrum to zero.
- void **clear** (**mha_spec_t** ∗s)

  Set all values of spectrum to zero.
- void **assign** (**mha_wave_t** self, **mha_real_t** val)

  Set all values of waveform 'self' to 'val'.
- void **assign** (**mha_wave_t** self, const **mha_wave_t** &val)

  Set all values of waveform 'self' to 'val'.
- void **assign** (**mha_spec_t** self, const **mha_spec_t** &val)

  Set all values of spectrum 'self' to 'val'.
- void **timeshift** (**mha_wave_t** &self, int shift)

  Time shift of waveform chunk.
- **mha_wave_t range** (**mha_wave_t** s, unsigned int k0, unsigned int len)

Return a time interval from a waveform chunk.

- **mha_spec_t channels** (**mha_spec_t** s, unsigned int ch_start, unsigned int nch)

  Return a channel interval from a spectrum.
- **mha_real_t** & **value** (**mha_wave_t** ∗s, unsigned int fr, unsigned int ch)

  Access an element of a waveform structure.
- const **mha_real_t** & **value** (const **mha_wave_t** ∗s, unsigned int fr, unsigned int ch)

  Constant access to an element of a waveform structure.
- **mha_complex_t** & **value** (**mha_spec_t** ∗s, unsigned int fr, unsigned int ch)

  Access to an element of a spectrum.
- const **mha_complex_t** & **value** (const **mha_spec_t** ∗s, unsigned int fr, unsigned int ch)

  Constant access to an element of a spectrum.
- **mha_real_t** & **value** (**mha_wave_t** &s, unsigned int fr, unsigned int ch)

  Access to an element of a waveform structure.
- const **mha_real_t** & **value** (const **mha_wave_t** &s, unsigned int fr, unsigned int ch)

  Constant access to an element of a waveform structure.
- **mha_complex_t** & **value** (**mha_spec_t** &s, unsigned int fr, unsigned int ch)

  Access to an element of a spectrum.
- const **mha_complex_t** & **value** (const **mha_spec_t** &s, unsigned int fr, unsigned int ch)

  Constant access to an element of a spectrum.
- std::vector< float > **std_vector_float** (const **mha_wave_t** &)

  *Converts a **mha_wave_t** (p. 154) structure into a std::vector<float> (interleaved order).*
- std::vector< std::vector< float > > **std_vector_vector_float** (const **mha_wave_t** &)

  *Converts a **mha_wave_t** (p. 154) structure into a std::vector< std::vector<float> > (outer vector represents channels).*
- std::vector< std::vector< **mha_complex_t** > > **std_vector_vector_complex** (const **mha_spec_t** &)

  *Converts a **mha_spec_t** (p. 141) structure into a std::vector< std::vector<mha_complex_t> > (outer vector represents channels).*
- **mha_wave_t** & **operator+=** (**mha_wave_t** &, const **mha_real_t** &)

  Addition operator.
- **mha_wave_t** & **operator+=** (**mha_wave_t** &, const **mha_wave_t** &)

  Addition operator.
- **mha_wave_t** & **operator-=** (**mha_wave_t** &, const **mha_wave_t** &)

  Subtraction operator.
- **mha_spec_t** & **operator-=** (**mha_spec_t** &, const **mha_spec_t** &)

  Subtraction operator.
- **mha_wave_t** & **operator∗=** (**mha_wave_t** &, const **mha_real_t** &)

  Element-wise multiplication operator.
- **mha_wave_t** & **operator∗=** (**mha_wave_t** &, const **mha_wave_t** &)

  Element-wise multiplication operator.
- **mha_spec_t** & **operator∗=** (**mha_spec_t** &, const **mha_real_t** &)

  Element-wise multiplication operator.
- **mha_spec_t** & **operator∗=** (**mha_spec_t** &, const **mha_wave_t** &)

  Element-wise multiplication operator.
- **mha_spec_t** & **operator∗=** (**mha_spec_t** &, const **mha_spec_t** &)

Element-wise multiplication operator.

- **mha_spec_t** & **operator/=** (**mha_spec_t** &, const **mha_spec_t** &)

  Element-wise division operator.
- **mha_wave_t** & **operator/=** (**mha_wave_t** &, const **mha_wave_t** &)

  Element-wise division operator.
- **mha_spec_t** & **operator+=** (**mha_spec_t** &, const **mha_spec_t** &)

  Addition operator.
- **mha_spec_t** & **operator+=** (**mha_spec_t** &, const **mha_real_t** &)

  Addition operator.
- **mha_spec_t** & **safe_div** (**mha_spec_t** &self, const **mha_spec_t** &v, **mha_real_t** eps)

  In-Place division with lower limit on divisor.
- **mha_wave_t** & **operator^=** (**mha_wave_t** &self, const **mha_real_t** &arg)

  Exponent operator.
- void **MHASignal::copy_channel** (**mha_spec_t** &self, const **mha_spec_t** &src, unsigned sch, unsigned dch)

  Copy one channel of a source signal.
- void **MHASignal::copy_channel** (**mha_wave_t** &self, const **mha_wave_t** &src, unsigned src_channel, unsigned dest_channel)

  Copy one channel of a source signal.
- **mha_real_t MHASignal::rmslevel** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen)

  Return RMS level of a spectrum channel.
- **mha_real_t MHASignal::colored_intensity** (const **mha_spec_t** &s, unsigned int channel, unsigned int fftlen, **mha_real_t** sqfreq_response[ ])

  Colored spectrum intensity.
- **mha_real_t MHASignal::maxabs** (const **mha_spec_t** &s, unsigned int channel)

  Find maximal absolute value.
- **mha_real_t MHASignal::rmslevel** (const **mha_wave_t** &s, unsigned int channel)

  Return RMS level of a waveform channel.
- **mha_real_t MHASignal::maxabs** (const **mha_wave_t** &s, unsigned int channel)

  Find maximal absolute value.
- **mha_real_t MHASignal::maxabs** (const **mha_wave_t** &s)

  Find maximal absolute value.
- **mha_real_t MHASignal::max** (const **mha_wave_t** &s)

  Find maximal value.
- **mha_real_t MHASignal::min** (const **mha_wave_t** &s)

  Find minimal value.
- **mha_real_t MHASignal::sumsqr_channel** (const **mha_wave_t** &s, unsigned int channel)

  Calculate sum of squared values in one channel.
- **mha_real_t MHASignal::sumsqr_frame** (const **mha_wave_t** &s, unsigned int frame)

  Calculate sum over all channels of squared values.
- void **MHASignal::limit** (**mha_wave_t** &s, const **mha_real_t** &min, const **mha_real_t** &max)

  Limit the singal in the waveform buffer to the range [min, max].
- **mha_complex_t** & **set** (**mha_complex_t** &self, **mha_real_t** real, **mha_real_t** imag=0)

Assign real and imaginary parts to a ***mha_complex_t*** *(p. 123) variable.*

- **mha_complex_t mha_complex** (**mha_real_t** real, **mha_real_t** imag=0)

  Create a new ***mha_complex_t*** *(p. 123) with specified real and imaginary parts.*

- **mha_complex_t** & **set** (**mha_complex_t** &self, const std::complex< **mha_real_t** > &**stdcomplex**)

  Assign a ***mha_complex_t*** *(p. 123) variable from a std::complex.*

- std::complex< **mha_real_t** > **stdcomplex** (const **mha_complex_t** &self)

  Create a std::complex from ***mha_complex_t*** *(p. 123).*

- **mha_complex_t** & **expi** (**mha_complex_t** &self, **mha_real_t angle**)

  replaces the value of the given ***mha_complex_t*** *(p. 123) with exp(i*b).*

- double **angle** (const **mha_complex_t** &self)

  Computes the angle of a complex number in the complex plane.

- **mha_complex_t** & **operator+=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Addition of two complex numbers, overwriting the first.

- **mha_complex_t operator+** (const **mha_complex_t** &self, const **mha_complex_t** &other)

  Addition of two complex numbers, result is a temporary object.

- **mha_complex_t** & **operator+=** (**mha_complex_t** &self, **mha_real_t** other_real)

  Addition of a complex and a real number, overwriting the complex.

- **mha_complex_t operator+** (const **mha_complex_t** &self, **mha_real_t** other_real)

  Addition of a complex and a real number, result is a temporary object.

- **mha_complex_t** & **operator-=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Subtraction of two complex numbers, overwriting the first.

- **mha_complex_t operator-** (const **mha_complex_t** &self, const **mha_complex_t** &other)

  Subtraction of two complex numbers, result is a temporary object.

- **mha_complex_t** & **operator-=** (**mha_complex_t** &self, **mha_real_t** other_real)

  Subtraction of a complex and a real number, overwriting the complex.

- **mha_complex_t operator-** (const **mha_complex_t** &self, **mha_real_t** other_real)

  Subtraction of a complex and a real number, result is a temporary object.

- **mha_complex_t** & **operator*=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Multiplication of two complex numbers, overwriting the first.

- **mha_complex_t operator*** (const **mha_complex_t** &self, const **mha_complex_t** &other)

  Multiplication of two complex numbers, result is a temporary object.

- **mha_complex_t** & **operator*=** (**mha_complex_t** &self, **mha_real_t** other_real)

  Multiplication of a complex and a real number, overwriting the complex.

- **mha_complex_t** & **expi** (**mha_complex_t** &self, **mha_real_t angle**, **mha_real_t** factor)

  replaces (!) the value of the given ***mha_complex_t*** *(p. 123) with a * exp(i*b)*

- **mha_complex_t operator*** (const **mha_complex_t** &self, **mha_real_t** other_real)

  Multiplication of a complex and a real number, result is a temporary object.

- **mha_real_t abs2** (const **mha_complex_t** &self)

  Compute the square of the absolute value of a complex value.

- **mha_real_t abs** (const **mha_complex_t** &self)

  Compute the absolute value of a complex value.

- **mha_complex_t** & **operator/=** (**mha_complex_t** &self, **mha_real_t** other_real)

  Division of a complex and a real number, overwriting the complex.
- **mha_complex_t** **operator/** (const **mha_complex_t** &self, **mha_real_t** other_real)

  Division of a complex and a real number, result is a temporary object.
- **mha_complex_t** & **safe_div** (**mha_complex_t** &self, const **mha_complex_t** &other, **mha_real_t** eps, **mha_real_t** eps2)

  Safe division of two complex numbers, overwriting the first.
- **mha_complex_t** & **operator/=** (**mha_complex_t** &self, const **mha_complex_t** &other)

  Division of two complex numbers, overwriting the first.
- **mha_complex_t** **operator/** (const **mha_complex_t** &self, const **mha_complex_↩ t** &other)

  Division of two complex numbers, result is a temporary object.
- **mha_complex_t** **operator-** (const **mha_complex_t** &self)

  Unary minus on a complex results in a negative temporary object.
- bool **operator==** (const **mha_complex_t** &x, const **mha_complex_t** &y)

  Compare two complex numbers for equality.
- bool **operator!=** (const **mha_complex_t** &x, const **mha_complex_t** &y)

  Compare two complex numbers for inequality.
- void **conjugate** (**mha_complex_t** &self)

  *Replace (!) the value of this **mha_complex_t** (p. 123) with its conjugate.*
- void **conjugate** (**mha_spec_t** &self)

  *Replace (!) the value of this **mha_spec_t** (p. 141) with its conjugate.*
- **mha_complex_t** **_conjugate** (const **mha_complex_t** &self)

  Compute the cojugate of this complex value.
- void **reciprocal** (**mha_complex_t** &self)

  Replace the value of this complex with its reciprocal.
- **mha_complex_t** **_reciprocal** (const **mha_complex_t** &self)

  compute the reciprocal of this complex value.
- void **normalize** (**mha_complex_t** &self)

  Divide a complex by its absolute value, thereby normalizing it (projecting onto the unit circle).
- void **normalize** (**mha_complex_t** &self, **mha_real_t** margin)

  Divide a complex by its absolute value, thereby normalizing it (projecting onto the unit circle), with a safety margin.
- bool **almost** (const **mha_complex_t** &self, const **mha_complex_t** &other, **mha_real_t** times_epsilon=1e2)

  Compare two complex numbers for equality except for a small relative error.
- bool **operator**< (const **mha_complex_t** &x, const **mha_complex_t** &y)

  Compares the absolute values of two complex numbers.
- std::ostream & **operator**<< (std::ostream &o, const **mha_complex_t** &c)

  *ostream operator for **mha_complex_t** (p. 123)*
- std::istream & **operator**>> (std::istream &i, **mha_complex_t** &c)

  *preliminary istream operator for **mha_complex_t** (p. 123) without error checking*
- **mha_fft_t** **mha_fft_new** (unsigned int n)

  Create a new FFT handle.
- void **mha_fft_free** (**mha_fft_t** h)

Destroy an FFT handle.
- void **mha_fft_wave2spec** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_t** ∗out)

  Tranform waveform segment into spectrum.
- void **mha_fft_wave2spec** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_t** ∗out, bool swaps)

  Tranform waveform segment into spectrum.
- void **mha_fft_spec2wave** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_t** ∗out)

  Tranform spectrum into waveform segment.
- void **mha_fft_spec2wave** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_t** ∗out, unsigned int offset)

  Tranform spectrum into waveform segment.
- void **mha_fft_forward** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)

  Complex to complex FFT (forward).
- void **mha_fft_backward** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)

  Complex to complex FFT (backward).
- void **mha_fft_forward_scale** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)

  Complex to complex FFT (forward).
- void **mha_fft_backward_scale** (**mha_fft_t** h, **mha_spec_t** ∗sIn, **mha_spec_t** ∗sOut)

  Complex to complex FFT (backward).
- void **mha_fft_wave2spec_scale** (**mha_fft_t** h, const **mha_wave_t** ∗in, **mha_spec_↩ t** ∗out)

  Tranform waveform segment into spectrum.
- void **mha_fft_spec2wave_scale** (**mha_fft_t** h, const **mha_spec_t** ∗in, **mha_wave_↩ t** ∗out)

  Tranform spectrum into waveform segment.
- float **MHAWindow::rect** (float)

  Rectangular window function.
- float **MHAWindow::bartlett** (float)

  Bartlett window function.
- float **MHAWindow::hanning** (float)

  Hanning window function.
- float **MHAWindow::hamming** (float)

  Hamming window function.
- float **MHAWindow::blackman** (float)

  Blackman window function.
- template< class elem_type >
  elem_type **MHASignal::kth_smallest** (elem_type array[ ], unsigned n, unsigned k)

  Fast search for the kth smallest element of an array.
- template< class elem_type >
  elem_type **MHASignal::median** (elem_type array[ ], unsigned n)

  Fast median search.
- template< class elem_type >
  elem_type **MHASignal::mean** (const std::vector< elem_type > &data, elem_type start↩ _val)

  Calculate average of elements in a vector.

- template< class elem_type >
  std::vector< elem_type > **MHASignal::quantile** (std::vector< elem_type > data, const std::vector< elem_type > &p)

  Calculate quantile of elements in a vector.
- void **MHASignal::saveas_mat4** (const **mha_spec_t** &data, const std::string &varname, FILE *fh)

  Save a openMHA spectrum as a variable in a Matlab4 file.
- void **MHASignal::saveas_mat4** (const **mha_wave_t** &data, const std::string &varname, FILE *fh)

  Save a openMHA waveform as a variable in a Matlab4 file.
- void **MHASignal::saveas_mat4** (const std::vector< **mha_real_t** > &data, const std↩ ::string &varname, FILE *fh)

  Save a float vector as a variable in a Matlab4 file.
- void **MHASignal::copy_permuted** (**mha_wave_t** *dest, const **mha_wave_t** *src)

  Copy contents of a waveform to a permuted waveform.

Variables

- unsigned long int **MHASignal::signal_counter** = 0

  Signal counter to produce signal ID strings.

5.8.1 Detailed Description

The classes for waveform, spectrum and filterbank signals defined in this file are "intelligent" versions of the basic waveform, spectrum and filterbank structures used in the C function calls.

5.9 mha_tablelookup.hh File Reference

Header file for table lookup classes.

Classes

- class **MHATableLookup::xy_table_t**

  Class for interpolation with non-equidistant x values.

Namespaces

- **MHATableLookup**

  Namespace for table lookup classes.

Index