

# AlsaPlayer Reference Manual

0.99.77

Generated by Doxygen 1.5.1

Sat Mar 17 03:37:10 2007



# Contents

<b>1</b>	<b>AlsaPlayer Data Structure Index</b>	<b>1</b>
1.1	AlsaPlayer Data Structures . . . . .	1
<b>2</b>	<b>AlsaPlayer File Index</b>	<b>3</b>
2.1	AlsaPlayer File List . . . . .	3
<b>3</b>	<b>AlsaPlayer Data Structure Documentation</b>	<b>5</b>
3.1	_input_object Struct Reference . . . . .	5
3.2	_input_plugin Struct Reference . . . . .	7
3.3	_interface_plugin Struct Reference . . . . .	9
3.4	_output_plugin Struct Reference . . . . .	10
3.5	_scope_plugin Struct Reference . . . . .	13
3.6	_stream_info Struct Reference . . . . .	15
<b>4</b>	<b>AlsaPlayer File Documentation</b>	<b>19</b>
4.1	input_plugin.h File Reference . . . . .	19
4.2	interface_plugin.h File Reference . . . . .	25
4.3	output_plugin.h File Reference . . . . .	27
4.4	scope_plugin.h File Reference . . . . .	29



# Chapter 1

# AlsaPlayer Data Structure Index

## 1.1 AlsaPlayer Data Structures

Here are the data structures with brief descriptions:

<a href="#">_input_object</a>	5
<a href="#">_input_plugin</a>	7
<a href="#">_interface_plugin</a>	9
<a href="#">_output_plugin</a>	10
<a href="#">_scope_plugin</a>	13
<a href="#">_stream_info</a>	15



# Chapter 2

## AlsaPlayer File Index

### 2.1 AlsaPlayer File List

Here is a list of all files with brief descriptions:

<a href="#">input_plugin.h</a>	19
<a href="#">interface_plugin.h</a>	25
<a href="#">output_plugin.h</a>	27
<a href="#">scope_plugin.h</a>	29





## Chapter 3

# AlsaPlayer Data Structure Documentation

### 3.1 `_input_object` Struct Reference

```
#include <input_plugin.h>
```

#### Data Fields

- int `ready`
- int `flags`
- int `nr_frames`
- int `nr_tracks`
- int `nr_channels`
- int `frame_size`
- void \* `local_data`
- pthread\_mutex\_t `object_mutex`

#### 3.1.1 Detailed Description

This is a structure that keeps frequently used parameters of an input instance. It also contains a pointer to any `local_data` that might be allocated by the plugin itself.

### 3.1.2 Field Documentation

#### 3.1.2.1 `int _input_object::ready`

Flag that should be set to 1 if your plugin is ready to accept `play_frame()` callback

#### 3.1.2.2 `int _input_object::flags`

Stream specific flags that should be set in the `open()` call. Read the description of the `P_*` definitions for details.

#### 3.1.2.3 `int _input_object::nr_frames`

The total number of frames in the stream. Should be set in the `open()` call.

#### 3.1.2.4 `int _input_object::nr_tracks`

The number of tracks, if any, in the stream. Should be set in the `open()` call.

#### 3.1.2.5 `int _input_object::nr_channels`

The number of PCM channels in the stream. Should always be 2 at this time.

#### 3.1.2.6 `int _input_object::frame_size`

The frame size in bytes. `play_frame()` will be called with this value.

#### 3.1.2.7 `void* _input_object::local_data`

If your plugin needs extra space for its own variables assign the allocated data structure to this pointer

#### 3.1.2.8 `pthread_mutex_t _input_object::object_mutex`

The object mutex. Used to lock and unlock the data structures. Initialized and called from the HOST.

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

- [input\\_plugin.h](#)

## 3.2 `_input_plugin` Struct Reference

```
#include <input_plugin.h>
```

### Data Fields

- `input_version_type` version
- `input_flags_type` flags
- `char * name`
- `char * author`
- `void * handle`
- `input_init_type` init
- `input_shutdown_type` shutdown
- `input_plugin_handle_type` plugin\_handle
- `input_can_handle_type` can\_handle
- `input_open_type` open
- `input_close_type` close
- `input_play_frame_type` play\_frame
- `input_frame_seek_type` frame\_seek
- `input_frame_size_type` frame\_size
- `input_nr_frames_type` nr\_frames
- `input_frame_to_sec_type` frame\_to\_sec
- `input_sample_rate_type` sample\_rate
- `input_channels_type` channels
- `input_stream_info_type` stream\_info
- `input_nr_tracks_type` nr\_tracks
- `input_track_seek_type` track\_seek

### 3.2.1 Field Documentation

#### 3.2.1.1 `input_version_type _input_plugin::version`

Must be set to `INPUT_PLUGIN_VERSION`

#### 3.2.1.2 `input_flags_type _input_plugin::flags`

Fixed flags for the plugin (`P_*`)

#### 3.2.1.3 `char* _input_plugin::name`

Should point to a character array containing the name of this plugin

**3.2.1.4 [char\\* \\_input\\_plugin::author](#)**

Should point to a character array containing the name of the author(s) of this plugin.

**3.2.1.5 [void\\* \\_input\\_plugin::handle](#)**

dlopen() handle of this plugin. Filled in by the HOST.

**3.2.1.6 [input\\_init\\_type \\_input\\_plugin::init](#)****3.2.1.7 [input\\_shutdown\\_type \\_input\\_plugin::shutdown](#)****3.2.1.8 [input\\_plugin\\_handle\\_type \\_input\\_plugin::plugin\\_handle](#)****3.2.1.9 [input\\_can\\_handle\\_type \\_input\\_plugin::can\\_handle](#)****3.2.1.10 [input\\_open\\_type \\_input\\_plugin::open](#)****3.2.1.11 [input\\_close\\_type \\_input\\_plugin::close](#)****3.2.1.12 [input\\_play\\_frame\\_type \\_input\\_plugin::play\\_frame](#)****3.2.1.13 [input\\_frame\\_seek\\_type \\_input\\_plugin::frame\\_seek](#)****3.2.1.14 [input\\_frame\\_size\\_type \\_input\\_plugin::frame\\_size](#)****3.2.1.15 [input\\_nr\\_frames\\_type \\_input\\_plugin::nr\\_frames](#)****3.2.1.16 [input\\_frame\\_to\\_sec\\_type \\_input\\_plugin::frame\\_to\\_sec](#)****3.2.1.17 [input\\_sample\\_rate\\_type \\_input\\_plugin::sample\\_rate](#)****3.2.1.18 [input\\_channels\\_type \\_input\\_plugin::channels](#)****3.2.1.19 [input\\_stream\\_info\\_type \\_input\\_plugin::stream\\_info](#)****3.2.1.20 [input\\_nr\\_tracks\\_type \\_input\\_plugin::nr\\_tracks](#)****3.2.1.21 [input\\_track\\_seek\\_type \\_input\\_plugin::track\\_seek](#)**

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

- [input\\_plugin.h](#)

## 3.3 \_interface\_plugin Struct Reference

```
#include <interface_plugin.h>
```

### Data Fields

- [interface\\_version\\_type version](#)
- char \* [name](#)
- char \* [author](#)
- void \* [handle](#)
- [interface\\_init\\_type init](#)
- [interface\\_start\\_type start](#)
- [interface\\_running\\_type running](#)
- [interface\\_stop\\_type stop](#)
- [interface\\_close\\_type close](#)

### 3.3.1 Field Documentation

**3.3.1.1** [interface\\_version\\_type \\_interface\\_plugin::version](#)

**3.3.1.2** [char\\* \\_interface\\_plugin::name](#)

**3.3.1.3** [char\\* \\_interface\\_plugin::author](#)

**3.3.1.4** [void\\* \\_interface\\_plugin::handle](#)

**3.3.1.5** [interface\\_init\\_type \\_interface\\_plugin::init](#)

**3.3.1.6** [interface\\_start\\_type \\_interface\\_plugin::start](#)

**3.3.1.7** [interface\\_running\\_type \\_interface\\_plugin::running](#)

**3.3.1.8** [interface\\_stop\\_type \\_interface\\_plugin::stop](#)

**3.3.1.9** [interface\\_close\\_type \\_interface\\_plugin::close](#)

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

- [interface\\_plugin.h](#)

## 3.4 `_output_plugin` Struct Reference

```
#include <output_plugin.h>
```

### Data Fields

- `output_version_type` version
- `char * name`
- `char * author`
- `output_init_type` init
- `output_open_type` open
- `output_close_type` close
- `output_write_type` write
- `output_start_callbacks_type` start\_callbacks
- `output_set_buffer_type` set\_buffer
- `output_set_sample_rate_type` set\_sample\_rate
- `output_get_queue_count_type` get\_queue\_count
- `output_get_latency_type` get\_latency

### 3.4.1 Field Documentation

#### 3.4.1.1 `output_version_type _output_plugin::version`

Version of output plugin. Must be OUTPUT\_PLUGIN\_VERSION

#### 3.4.1.2 `char* _output_plugin::name`

Name of output plugin

#### 3.4.1.3 `char* _output_plugin::author`

Author of the plugin

#### 3.4.1.4 `output_init_type _output_plugin::init`

Initialize output plugin. Called before the plugin is opened for use

#### 3.4.1.5 [output\\_open\\_type \\_output\\_plugin::open](#)

**Parameters:**

*path* The path or device designation that should be used

Opens the output plugin. A value of 1 should be returned on success, 0 on failure.

#### 3.4.1.6 [output\\_close\\_type \\_output\\_plugin::close](#)

Close the output plugin

#### 3.4.1.7 [output\\_write\\_type \\_output\\_plugin::write](#)

**Parameters:**

*data* Buffer that contains the data Number of bytes that should be read from the buffer

Write out data to the output device. This is a byte count and will typically be the same size as a fragment. A value of 1 should be returned on success, 0 on failure.

#### 3.4.1.8 [output\\_start\\_callbacks\\_type \\_output\\_plugin::start\\_callbacks](#)

**Parameters:**

*data* pointer to bufs structure in AlsaNode

This function is used for callback based plugins like JACK

#### 3.4.1.9 [output\\_set\\_buffer\\_type \\_output\\_plugin::set\\_buffer](#)

**Parameters:**

*frag\_size* Fragment size to use (in bytes)

*frag\_count* Fragment count to use (in bytes)

*channels* Number of channels to use

Set up the output device with the given parameters. Some output devices do not accept such configurations in which case they should just be ignored, but still expect frag\_size data chunks in the write function. A value of 1 should be returned on success, 0 on failure.

#### 3.4.1.10 [output\\_set\\_sample\\_rate\\_type \\_output\\_plugin::set\\_sample\\_rate](#)

**Parameters:**

*rate* Sample rate to use

Set the sample rate of the output device. A value of 1 should be returned on success, 0 on failure.

#### 3.4.1.11 [output\\_get\\_queue\\_count\\_type \\_output\\_plugin::get\\_queue\\_count](#)

Returns the number of bytes pending in the hardware buffer of output device. This function is optional.

#### 3.4.1.12 [output\\_get\\_latency\\_type \\_output\\_plugin::get\\_latency](#)

Returns the latency of the output device in bytes. This function is optional.

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

- [output\\_plugin.h](#)



## 3.5 `_scope_plugin` Struct Reference

```
#include <scope_plugin.h>
```

### Data Fields

- `scope_version_type` `version`
- `char *` `name`
- `char *` `author`
- `void *` `handle`
- `scope_init_type` `init`
- `scope_start_type` `start`
- `scope_running_type` `running`
- `scope_stop_type` `stop`
- `scope_shutdown_type` `shutdown`
- `scope_set_data_type` `set_data`
- `scope_set_fft_type` `set_fft`

### 3.5.1 Detailed Description

You should declare a `scope_plugin` variable and populate it with pointers of the specific functions implemented by your scope

### 3.5.2 Field Documentation

#### 3.5.2.1 `scope_version_type _scope_plugin::version`

Set to `SCOPE_PLUGIN_VERSION`

#### 3.5.2.2 `char* _scope_plugin::name`

Point to a character array with the name of the scope

#### 3.5.2.3 `char* _scope_plugin::author`

Point to a character array with the name of the author(s) of the scope

#### 3.5.2.4 `void* _scope_plugin::handle`

Pointer to a `dlopen()` handle. This is filled in by the HOST. Set to `NULL`.

**3.5.2.5 [scope\\_init\\_type \\_scope\\_plugin::init](#)**

Should point to the implementation of your [init\(\)](#) function. Required by the HOST.

**3.5.2.6 [scope\\_start\\_type \\_scope\\_plugin::start](#)**

Should point to the implementation of your [start\(\)](#) function. Required by the HOST.

**3.5.2.7 [scope\\_running\\_type \\_scope\\_plugin::running](#)**

Should point to the implementation of your [running\(\)](#) function. Required by the HOST.

**3.5.2.8 [scope\\_stop\\_type \\_scope\\_plugin::stop](#)**

Should point to the implementation of your [stop\(\)](#) function. Required by the HOST.

**3.5.2.9 [scope\\_shutdown\\_type \\_scope\\_plugin::shutdown](#)**

Should point to the implementation of your [shutdown\(\)](#) function. Required by the HOST.

**3.5.2.10 [scope\\_set\\_data\\_type \\_scope\\_plugin::set\\_data](#)**

Should point to the function that collects PCM data. If you don't want PCM data set to NULL.

**3.5.2.11 [scope\\_set\\_fft\\_type \\_scope\\_plugin::set\\_fft](#)**

Should point to the function that collects FFT data. If you don't want FFT data set to NULL. NB. [set\\_data](#) and [set\\_fft](#) can't both be NULL, at least one must be set.

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

- [scope\\_plugin.h](#)

## 3.6 `_stream_info` Struct Reference

```
#include <input_plugin.h>
```

### Data Fields

- char `stream_type` [128]
- char `artist` [128]
- char `title` [128]
- char `album` [128]
- char `genre` [128]
- char `year` [10]
- char `track` [10]
- char `comment` [128]
- char `status` [32]
- char `path` [1024]
- int `channels`
- int `tracks`
- int `current_track`
- int `sample_rate`
- int `bitrate`

### 3.6.1 Detailed Description

This structure is used to pass information about a stream/song from the plugin to the host.

### 3.6.2 Field Documentation

#### 3.6.2.1 char `_stream_info::stream_type`[128]

Should describe what type of stream this is (MP3, OGG, etc). May also contain format data and things like sample rate. Text

#### 3.6.2.2 char `_stream_info::artist`[128]

Author of the stream. Usually the name of the Artist or Band

#### 3.6.2.3 char `_stream_info::title`[128]

The song title.

**3.6.2.4 char [\\_stream\\_info::album](#)[128]**

The album name.

**3.6.2.5 char [\\_stream\\_info::genre](#)[128]**

The genre of this song

**3.6.2.6 char [\\_stream\\_info::year](#)[10]**

The year of this song

**3.6.2.7 char [\\_stream\\_info::track](#)[10]**

The track number of this song

**3.6.2.8 char [\\_stream\\_info::comment](#)[128]**

The comment of this song

**3.6.2.9 char [\\_stream\\_info::status](#)[32]**

The status of the plugin. Can have something like "Seeking..." or perhaps "Buffering" depending on what the plugin instance is doing.

**3.6.2.10 char [\\_stream\\_info::path](#)[1024]**

The path of the stream

**3.6.2.11 int [\\_stream\\_info::channels](#)**

The number of channels

**3.6.2.12 int [\\_stream\\_info::tracks](#)**

The number of tracks

**3.6.2.13 int [\\_stream\\_info::current\\_track](#)**

The current track;

**3.6.2.14** `int _stream_info::sample_rate`

The sampling rate

**3.6.2.15** `int _stream_info::bitrate`

The bitrate

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

- [input\\_plugin.h](#)



## Chapter 4

# AlsaPlayer File Documentation

### 4.1 input\_plugin.h File Reference

```
#include <pthread.h>
```

#### Data Structures

- struct [\\_input\\_object](#)
- struct [\\_stream\\_info](#)
- struct [\\_input\\_plugin](#)

#### Defines

- #define [P\\_SEEK](#) 1
- #define [P\\_PERFECTSEEK](#) 2
- #define [P\\_REENTRANT](#) 4
- #define [P\\_FILEBASED](#) 8
- #define [P\\_STREAMBASED](#) 16
- #define [P\\_BUFFERING](#) 32
- #define [INPUT\\_PLUGIN\\_BASE\\_VERSION](#) 0x1000
- #define [INPUT\\_PLUGIN\\_VERSION](#) (INPUT\_PLUGIN\_BASE\_VERSION + 16)

#### Typedefs

- typedef [\\_input\\_object](#) [input\\_object](#)
- typedef [\\_stream\\_info](#) [stream\\_info](#)

- typedef int [input\\_version\\_type](#)
- typedef int [input\\_flags\\_type](#)
- typedef int(\*) [input\\_init\\_type](#) (void)
- typedef void(\*) [input\\_shutdown\\_type](#) (void)
- typedef void \* [input\\_plugin\\_handle\\_type](#)
- typedef float(\*) [input\\_can\\_handle\\_type](#) (const char \*path)
- typedef int(\*) [input\\_open\\_type](#) (input\_object \*obj, const char \*path)
- typedef void(\*) [input\\_close\\_type](#) (input\_object \*obj)
- typedef int(\*) [input\\_play\\_frame\\_type](#) (input\_object \*obj, char \*buffer)
- typedef int(\*) [input\\_frame\\_seek\\_type](#) (input\_object \*obj, int frame)
- typedef int(\*) [input\\_frame\\_size\\_type](#) (input\_object \*obj)
- typedef int(\*) [input\\_nr\\_frames\\_type](#) (input\_object \*obj)
- typedef long(\*) [input\\_frame\\_to\\_sec\\_type](#) (input\_object \*obj, int frame)
- typedef int(\*) [input\\_sample\\_rate\\_type](#) (input\_object \*obj)
- typedef int(\*) [input\\_channels\\_type](#) (input\_object \*obj)
- typedef int(\*) [input\\_stream\\_info\\_type](#) (input\_object \*obj, stream\_info \*info)
- typedef int(\*) [input\\_nr\\_tracks\\_type](#) (input\_object \*obj)
- typedef int(\*) [input\\_track\\_seek\\_type](#) (input\_object \*obj, int track)
- typedef [\\_input\\_plugin](#) [input\\_plugin](#)
- typedef [input\\_plugin](#) \*(\*) [input\\_plugin\\_info\\_type](#) (void)

### 4.1.1 Define Documentation

#### 4.1.1.1 #define INPUT\_PLUGIN\_BASE\_VERSION 0x1000

The base version number of the scope plugin. Set at 0x1000.

#### 4.1.1.2 #define INPUT\_PLUGIN\_VERSION (INPUT\_PLUGIN\_BASE\_VERSION + 16)

The version of the input plugin API. This should be incremented whenever structural changes are made to the API. This value should only be changed by the maintainers.

#### 4.1.1.3 #define P\_BUFFERING 32

Set minimal buffer

#### 4.1.1.4 #define P\_FILEBASED 8

Set this flag if the stream is file based (local disk file)



#### 4.1.1.5 `#define P_PERFECTSEEK 2`

Set this flag if your plugin is able to do sample accurate seeking in the stream. This is required for reverse speed playback.

#### 4.1.1.6 `#define P_REENTRANT 4`

Set this flag if your plugin is reentrant.

#### 4.1.1.7 `#define P_SEEK 1`

Set this flag if your plugin is able to seek in the stream

#### 4.1.1.8 `#define P_STREAMBASED 16`

Set this if the stream is a real stream e.g. HTTP or UDP based

### 4.1.2 Typedef Documentation

#### 4.1.2.1 `typedef float(*) input\_can\_handle\_type(const char *path)`

**Parameters:**

*path* Path to stream

Returns a rating between 0.0 and 1.0 for how well this plugin can handle the given path  
1.0 = Excellent 0.0 = Huh?

#### 4.1.2.2 `typedef int(*) input\_channels\_type(input_object *obj)`

**Parameters:**

*obj* input object

Returns number of channels in the stream

#### 4.1.2.3 `typedef void(*) input\_close\_type(input_object *obj)`

**Parameters:**

*obj* input object

Close stream

#### 4.1.2.4 `typedef int input_flags_type`

Capability flags for this plugin

#### 4.1.2.5 `typedef int(*) input_frame_seek_type(input_object *obj, int frame)`

**Parameters:**

*obj* input object

*frame* Seek to a specific frame number

#### 4.1.2.6 `typedef int(*) input_frame_size_type(input_object *obj)`

**Parameters:**

*obj* input object

Returns the frame size in bytes

#### 4.1.2.7 `typedef long(*) input_frame_to_sec_type(input_object *obj, int frame)`

**Parameters:**

*obj* input object

*frame* frame number

Returns the offset from the start time in centiseconds (100th of a second) for the frame given.

#### 4.1.2.8 `typedef int(*) input_init_type(void)`

Init plugin

#### 4.1.2.9 `typedef int(*) input_nr_frames_type(input_object *obj)`

**Parameters:**

*obj* input object

Returns the total number of frames in the stream

#### 4.1.2.10 typedef int(\*) [input\\_nr\\_tracks\\_type](#)([input\\_object](#) \*obj)

**Parameters:**

*obj* input object

Return number of tracks. Optional

#### 4.1.2.11 typedef struct [\\_input\\_object](#) [input\\_object](#)

This is a structure that keeps frequently used parameters of an input instance. It also contains a pointer to any local\_data that might be allocated by the plugin itself.

#### 4.1.2.12 typedef int(\*) [input\\_open\\_type](#)([input\\_object](#) \*obj, const char \*path)

**Parameters:**

*obj* input object

*path* path of stream to open

Open stream

#### 4.1.2.13 typedef int(\*) [input\\_play\\_frame\\_type](#)([input\\_object](#) \*obj, char \*buffer)

**Parameters:**

*obj* input object

*buffer* buffer where we should write the frame to

Play/decode a single frame. This function should write exactly one frame to the buffer. If there is not enough PCM data to fill the frame it should be padded with zeros (silence).

#### 4.1.2.14 typedef struct [\\_input\\_plugin](#) [input\\_plugin](#)

#### 4.1.2.15 typedef void\* [input\\_plugin\\_handle\\_type](#)

Handle for plugin. Filled in by the host

#### 4.1.2.16 typedef [input\\_plugin](#)\*(\*) [input\\_plugin\\_info\\_type](#)(void)

Every input plugin should have an [input\\_plugin\\_info\(\)](#) function that returns a pointer to an [input\\_plugin](#) structure that is set up with pointers to your implementations. If

your plugin is compiled using C++ make sure you 'extern "C"' the `input_plugin_info()` function or else the HOST will not be able to load the plugin.

#### 4.1.2.17 `typedef int(*) input_sample_rate_type(input_object *obj)`

##### Parameters:

*obj* input object

Returns the sample rate of the stream

#### 4.1.2.18 `typedef void(*) input_shutdown_type(void)`

Prepare the plugin for removal

#### 4.1.2.19 `typedef int(*) input_stream_info_type(input_object *obj, stream_info *info)`

##### Parameters:

*obj* input object

*info* pointer to stream\_info structure

Return stream info of the current stream. You should not allocate space for the stream\_info structure. The HOST will take care of that.

#### 4.1.2.20 `typedef int(*) input_track_seek_type(input_object *obj, int track)`

#### 4.1.2.21 `typedef int input_version_type`

input plugin binary version. Must be set to INPUT\_PLUGIN\_VERSION

#### 4.1.2.22 `typedef struct _stream_info stream_info`

This structure is used to pass information about a stream/song from the plugin to the host.

## 4.2 interface\_plugin.h File Reference

```
#include "CorePlayer.h"
#include "Playlist.h"
```

### Data Structures

- struct [\\_interface\\_plugin](#)

### Defines

- #define [INTERFACE\\_PLUGIN\\_BASE\\_VERSION](#) 0x1000
- #define [INTERFACE\\_PLUGIN\\_VERSION](#) (INTERFACE\_PLUGIN\_BASE\_VERSION + 4)

### Typedefs

- typedef int [interface\\_version\\_type](#)
- typedef int(\*) [interface\\_init\\_type](#) ()
- typedef int(\*) [interface\\_start\\_type](#) (Playlist \*, int, char \*\*)
- typedef int(\*) [interface\\_running\\_type](#) ()
- typedef int(\*) [interface\\_stop\\_type](#) ()
- typedef void(\*) [interface\\_close\\_type](#) ()
- typedef [\\_interface\\_plugin](#) [interface\\_plugin](#)
- typedef [interface\\_plugin](#) \*(\*) [interface\\_plugin\\_info\\_type](#) ()

### 4.2.1 Define Documentation

4.2.1.1 `#define INTERFACE_PLUGIN_BASE_VERSION 0x1000`

4.2.1.2 `#define INTERFACE_PLUGIN_VERSION (INTERFACE_PLUGIN_ -  
BASE_VERSION + 4)`

### 4.2.2 Typedef Documentation

4.2.2.1 `typedef void(*) interface\_close\_type()`

4.2.2.2 `typedef int(*) interface\_init\_type()`

4.2.2.3 `typedef struct \_interface\_plugin interface\_plugin`

4.2.2.4 `typedef interface\_plugin*(*) interface\_plugin\_info\_type()`

4.2.2.5 `typedef int(*) interface\_running\_type()`

4.2.2.6 `typedef int(*) interface\_start\_type(Playlist *, int, char **)`

4.2.2.7 `typedef int(*) interface\_stop\_type()`

4.2.2.8 `typedef int interface\_version\_type`

## 4.3 output\_plugin.h File Reference

### Data Structures

- struct [\\_output\\_plugin](#)

### Defines

- #define [OUTPUT\\_PLUGIN\\_BASE\\_VERSION](#) 0x1000
- #define [OUTPUT\\_PLUGIN\\_VERSION](#) (OUTPUT\_PLUGIN\_BASE\_VERSION + 6)

### Typedefs

- typedef int [output\\_version\\_type](#)
- typedef int(\*) [output\\_init\\_type](#) (void)
- typedef int(\*) [output\\_open\\_type](#) (const char \*path)
- typedef void(\*) [output\\_close\\_type](#) (void)
- typedef int(\*) [output\\_write\\_type](#) (void \*data, int byte\_count)
- typedef int(\*) [output\\_start\\_callbacks\\_type](#) (void \*data)
- typedef int(\*) [output\\_set\\_buffer\\_type](#) (int \*frag\_size, int \*frag\_count, int \*channels)
- typedef unsigned int(\*) [output\\_set\\_sample\\_rate\\_type](#) (unsigned int rate)
- typedef int(\*) [output\\_get\\_queue\\_count\\_type](#) (void)
- typedef int(\*) [output\\_get\\_latency\\_type](#) (void)
- typedef [\\_output\\_plugin](#) [output\\_plugin](#)
- typedef [output\\_plugin](#) \*(\*) [output\\_plugin\\_info\\_type](#) (void)

### 4.3.1 Define Documentation

4.3.1.1 `#define OUTPUT_PLUGIN_BASE_VERSION 0x1000`

4.3.1.2 `#define OUTPUT_PLUGIN_VERSION (OUTPUT_PLUGIN_BASE_VERSION + 6)`

### 4.3.2 Typedef Documentation

4.3.2.1 `typedef void(*) output_close_type(void)`

4.3.2.2 `typedef int(*) output_get_latency_type(void)`

4.3.2.3 `typedef int(*) output_get_queue_count_type(void)`

4.3.2.4 `typedef int(*) output_init_type(void)`

4.3.2.5 `typedef int(*) output_open_type(const char *path)`

4.3.2.6 `typedef struct _output_plugin output_plugin`

4.3.2.7 `typedef output_plugin*(*) output_plugin_info_type(void)`

4.3.2.8 `typedef int(*) output_set_buffer_type(int *frag_size, int *frag_count, int *channels)`

4.3.2.9 `typedef unsigned int(*) output_set_sample_rate_type(unsigned int rate)`

4.3.2.10 `typedef int(*) output_start_callbacks_type(void *data)`

4.3.2.11 `typedef int output_version_type`

4.3.2.12 `typedef int(*) output_write_type(void *data, int byte_count)`



## 4.4 scope\_plugin.h File Reference

### Data Structures

- struct [\\_scope\\_plugin](#)

### Defines

- #define [SCOPE\\_PLUGIN\\_BASE\\_VERSION](#) 0x1000
- #define [SCOPE\\_PLUGIN\\_VERSION](#) (SCOPE\_PLUGIN\_BASE\_VERSION + 7)
- #define [SCOPE\\_NICE](#) 10
- #define [SCOPE\\_SLEEP](#) 20000
- #define [SCOPE\\_BG\\_RED](#) 0
- #define [SCOPE\\_BG\\_GREEN](#) 0
- #define [SCOPE\\_BG\\_BLUE](#) 0

### Typedefs

- typedef int [scope\\_version\\_type](#)
- typedef int(\*) [scope\\_init\\_type](#) (void \*arg)
- typedef void(\*) [scope\\_start\\_type](#) (void)
- typedef int(\*) [scope\\_running\\_type](#) (void)
- typedef void(\*) [scope\\_stop\\_type](#) (void)
- typedef void(\*) [scope\\_shutdown\\_type](#) (void)
- typedef void(\*) [scope\\_set\\_data\\_type](#) (void \*buffer, int count)
- typedef void(\*) [scope\\_set\\_fft\\_type](#) (void \*buffer, int samples, int channels)
- typedef [\\_scope\\_plugin](#) [scope\\_plugin](#)
- typedef [scope\\_plugin](#) (\*) [scope\\_plugin\\_info\\_type](#) (void)

#### 4.4.1 Define Documentation

##### 4.4.1.1 #define SCOPE\_BG\_BLUE 0

The value of the BLUE component of the default background color for scope windows. Value should be from 0-255

##### 4.4.1.2 #define SCOPE\_BG\_GREEN 0

The value of the GREEN component of the default background color for scope windows. Value should be from 0-255

#### 4.4.1.3 `#define SCOPE_BG_RED 0`

The value of the RED component of the default background color for scope windows. Value should be from 0-255

#### 4.4.1.4 `#define SCOPE_NICE 10`

The default nice level scope plugins should be set at. Most scope plugins are just eye candy and as such should not interfere with other processes on your system. They should only use CPU cycles that would otherwise be wasted. Setting the scopes to a nice level of 10 or higher pretty much insures this. If you don't like this policy you can lower the value. Keep in mind that negative values will only work if you run the HOST as root

#### 4.4.1.5 `#define SCOPE_PLUGIN_BASE_VERSION 0x1000`

The base version number of the scope plugin. Set at 0x1000

#### 4.4.1.6 `#define SCOPE_PLUGIN_VERSION (SCOPE_PLUGIN_BASE_VERSION + 7)`

The version of the scope plugin API. This should be increased whenever structural changes are made to the API. This value should only be changed by the maintainers.

#### 4.4.1.7 `#define SCOPE_SLEEP 20000`

The default sleep time in microseconds for scopes. After every render iteration a scope should sleep for this amount of time. You should use the `dosleep()` call i.e. `dosleep(SCOPE_SLEEP)`. A value of 20000 will let scopes run at  $100000/20000 = 50$  frames per second. If the scopes are consuming too much CPU consider raising this value.

### 4.4.2 Typedef Documentation

#### 4.4.2.1 `typedef int(*) scope_init_type(void *arg)`

The init function of a scope plugin. This function should initialize all data structures needed for the scope plugin. Return value should be 1 on success, 0 if initialization fails.

##### Parameters:

*Set* to NULL. This currently used for internal plugins only

#### 4.4.2.2 typedef struct [\\_scope\\_plugin](#) [scope\\_plugin](#)

You should declare a `scope_plugin` variable and populate it with pointers of the specific functions implemented by your scope

#### 4.4.2.3 typedef [scope\\_plugin](#)\*(\*) [scope\\_plugin\\_info\\_type](#)(void)

Every scope plugin should have a `scope_plugin_info()` function that returns a pointer to a `scope_plugin` structure that is filled with pointers to your function implementations.

#### 4.4.2.4 typedef int(\*) [scope\\_running\\_type](#)(void)

This function should tell the HOST if the scope is running i.e. on-screen and rendering. A value of 1 should be returned if this is the case, 0 if the scope is not active.

#### 4.4.2.5 typedef void(\*) [scope\\_set\\_data\\_type](#)(void \*buffer, int count)

##### Parameters:

*buffer* pointer to buffer data

*count* number of short (int16) samples in buffer

The `set_data` function should be defined if your scope wants to get it hands on PCM data. The format of the buffer is short (int16) interleaved stereo data. A count value of 1024 means there are 2048 short samples in the buffer. These samples are interleaved, so even sample positions are from the left channel, uneven sample positions from the right channel. The API will be changed to accommodate variable channels in the not too distant future.

#### 4.4.2.6 typedef void(\*) [scope\\_set\\_fft\\_type](#)(void \*buffer, int samples, int channels)

##### Parameters:

*buffer* buffer with FFT values

*samples* number of FFT values per channel

*channels* number of channels

This function should be defined if your scope wants to get FFT data. The HOST typically calculates 256 FFT values per channel (going from low frequency range to high). The value is between 0-256. The buffer format is NON-interleaved int (int32). So if `samples = 256` and `channels = 2` then there are  $2 * 256$  number of samples in the buffer. The first 256 are for channel 1, the other 256 for channel 2.

**4.4.2.7 typedef void(\*) [scope\\_shutdown\\_type](#)(void)**

The shutdown function is called just before the plugin is unloaded or just before the HOST decides to exit. All data structures allocated in the init routine should be freed here.

**4.4.2.8 typedef void(\*) [scope\\_start\\_type](#)(void)**

This function will be called when the HOST wants to activate the scope. It should pop up the scope window and start rendering the PCM or FFT data

**4.4.2.9 typedef void(\*) [scope\\_stop\\_type](#)(void)**

This function should stop and close the scope window if it was running. It should just return if the scope is not running.

**4.4.2.10 typedef int [scope\\_version\\_type](#)**

The API this scope was compiled against. It should always be set to SCOPE\_PLUGIN\_VERSION. Failing to set this will most likely result in a scope plugin that won't load.

# Index

- [\\_input\\_object](#), [5](#)
  - [flags](#), [6](#)
  - [frame\\_size](#), [6](#)
  - [local\\_data](#), [6](#)
  - [nr\\_channels](#), [6](#)
  - [nr\\_frames](#), [6](#)
  - [nr\\_tracks](#), [6](#)
  - [object\\_mutex](#), [6](#)
  - [ready](#), [6](#)
- [\\_input\\_plugin](#), [7](#)
  - [author](#), [7](#)
  - [can\\_handle](#), [8](#)
  - [channels](#), [8](#)
  - [close](#), [8](#)
  - [flags](#), [7](#)
  - [frame\\_seek](#), [8](#)
  - [frame\\_size](#), [8](#)
  - [frame\\_to\\_sec](#), [8](#)
  - [handle](#), [8](#)
  - [init](#), [8](#)
  - [name](#), [7](#)
  - [nr\\_frames](#), [8](#)
  - [nr\\_tracks](#), [8](#)
  - [open](#), [8](#)
  - [play\\_frame](#), [8](#)
  - [plugin\\_handle](#), [8](#)
  - [sample\\_rate](#), [8](#)
  - [shutdown](#), [8](#)
  - [stream\\_info](#), [8](#)
  - [track\\_seek](#), [8](#)
  - [version](#), [7](#)
- [\\_interface\\_plugin](#), [9](#)
  - [author](#), [9](#)
  - [close](#), [9](#)
  - [handle](#), [9](#)
  - [init](#), [9](#)
  - [name](#), [9](#)
  - [running](#), [9](#)
  - [start](#), [9](#)
  - [stop](#), [9](#)
  - [version](#), [9](#)
- [\\_output\\_plugin](#), [10](#)
  - [author](#), [10](#)
  - [close](#), [11](#)
  - [get\\_latency](#), [12](#)
  - [get\\_queue\\_count](#), [12](#)
  - [init](#), [10](#)
  - [name](#), [10](#)
  - [open](#), [10](#)
  - [set\\_buffer](#), [11](#)
  - [set\\_sample\\_rate](#), [11](#)
  - [start\\_callbacks](#), [11](#)
  - [version](#), [10](#)
  - [write](#), [11](#)
- [\\_scope\\_plugin](#), [13](#)
  - [author](#), [13](#)
  - [handle](#), [13](#)
  - [init](#), [13](#)
  - [name](#), [13](#)
  - [running](#), [14](#)
  - [set\\_data](#), [14](#)
  - [set\\_fft](#), [14](#)
  - [shutdown](#), [14](#)
  - [start](#), [14](#)
  - [stop](#), [14](#)
  - [version](#), [13](#)
- [\\_stream\\_info](#), [15](#)
  - [album](#), [15](#)
  - [artist](#), [15](#)
  - [bitrate](#), [17](#)
  - [channels](#), [16](#)
  - [comment](#), [16](#)
  - [current\\_track](#), [16](#)
  - [genre](#), [16](#)

- path, 16
- sample\_rate, 16
- status, 16
- stream\_type, 15
- title, 15
- track, 16
- tracks, 16
- year, 16
- album
  - \_stream\_info, 15
- artist
  - \_stream\_info, 15
- author
  - \_input\_plugin, 7
  - \_interface\_plugin, 9
  - \_output\_plugin, 10
  - \_scope\_plugin, 13
- bitrate
  - \_stream\_info, 17
- can\_handle
  - \_input\_plugin, 8
- channels
  - \_input\_plugin, 8
  - \_stream\_info, 16
- close
  - \_input\_plugin, 8
  - \_interface\_plugin, 9
  - \_output\_plugin, 11
- comment
  - \_stream\_info, 16
- current\_track
  - \_stream\_info, 16
- flags
  - \_input\_object, 6
  - \_input\_plugin, 7
- frame\_seek
  - \_input\_plugin, 8
- frame\_size
  - \_input\_object, 6
  - \_input\_plugin, 8
- frame\_to\_sec
  - \_input\_plugin, 8
- genre
  - \_stream\_info, 16
- get\_latency
  - \_output\_plugin, 12
- get\_queue\_count
  - \_output\_plugin, 12
- handle
  - \_input\_plugin, 8
  - \_interface\_plugin, 9
  - \_scope\_plugin, 13
- init
  - \_input\_plugin, 8
  - \_interface\_plugin, 9
  - \_output\_plugin, 10
  - \_scope\_plugin, 13
- input\_can\_handle\_type
  - input\_plugin.h, 21
- input\_channels\_type
  - input\_plugin.h, 21
- input\_close\_type
  - input\_plugin.h, 21
- input\_flags\_type
  - input\_plugin.h, 21
- input\_frame\_seek\_type
  - input\_plugin.h, 22
- input\_frame\_size\_type
  - input\_plugin.h, 22
- input\_frame\_to\_sec\_type
  - input\_plugin.h, 22
- input\_init\_type
  - input\_plugin.h, 22
- input\_nr\_frames\_type
  - input\_plugin.h, 22
- input\_nr\_tracks\_type
  - input\_plugin.h, 22
- input\_object
  - input\_plugin.h, 23
- input\_open\_type
  - input\_plugin.h, 23
- input\_play\_frame\_type
  - input\_plugin.h, 23
- input\_plugin
  - input\_plugin.h, 23
- input\_plugin.h, 19

- input\_can\_handle\_type, 21
- input\_channels\_type, 21
- input\_close\_type, 21
- input\_flags\_type, 21
- input\_frame\_seek\_type, 22
- input\_frame\_size\_type, 22
- input\_frame\_to\_sec\_type, 22
- input\_init\_type, 22
- input\_nr\_frames\_type, 22
- input\_nr\_tracks\_type, 22
- input\_object, 23
- input\_open\_type, 23
- input\_play\_frame\_type, 23
- input\_plugin, 23
- INPUT\_PLUGIN\_BASE\_-  
VERSION, 20
- input\_plugin\_handle\_type, 23
- input\_plugin\_info\_type, 23
- INPUT\_PLUGIN\_VERSION, 20
- input\_sample\_rate\_type, 24
- input\_shutdown\_type, 24
- input\_stream\_info\_type, 24
- input\_track\_seek\_type, 24
- input\_version\_type, 24
- P\_BUFFERING, 20
- P\_FILEBASED, 20
- P\_PERFECTSEEK, 20
- P\_REENTRANT, 21
- P\_SEEK, 21
- P\_STREAMBASED, 21
- stream\_info, 24
- INPUT\_PLUGIN\_BASE\_VERSION
  - input\_plugin.h, 20
- input\_plugin\_handle\_type
  - input\_plugin.h, 23
- input\_plugin\_info\_type
  - input\_plugin.h, 23
- INPUT\_PLUGIN\_VERSION
  - input\_plugin.h, 20
- input\_sample\_rate\_type
  - input\_plugin.h, 24
- input\_shutdown\_type
  - input\_plugin.h, 24
- input\_stream\_info\_type
  - input\_plugin.h, 24
- input\_track\_seek\_type
  - input\_plugin.h, 24
- input\_plugin.h, 24
- input\_version\_type
  - input\_plugin.h, 24
- interface\_close\_type
  - interface\_plugin.h, 26
- interface\_init\_type
  - interface\_plugin.h, 26
- interface\_plugin
  - interface\_plugin.h, 26
- interface\_plugin.h, 25
  - interface\_close\_type, 26
  - interface\_init\_type, 26
  - interface\_plugin, 26
  - INTERFACE\_PLUGIN\_BASE\_-  
VERSION, 26
  - interface\_plugin\_info\_type, 26
  - INTERFACE\_PLUGIN\_-  
VERSION, 26
  - interface\_running\_type, 26
  - interface\_start\_type, 26
  - interface\_stop\_type, 26
  - interface\_version\_type, 26
  - INTERFACE\_PLUGIN\_BASE\_-  
VERSION
    - interface\_plugin.h, 26
  - interface\_plugin\_info\_type
    - interface\_plugin.h, 26
  - INTERFACE\_PLUGIN\_VERSION
    - interface\_plugin.h, 26
  - interface\_running\_type
    - interface\_plugin.h, 26
  - interface\_start\_type
    - interface\_plugin.h, 26
  - interface\_stop\_type
    - interface\_plugin.h, 26
  - interface\_version\_type
    - interface\_plugin.h, 26
- local\_data
  - \_input\_object, 6
- name
  - \_input\_plugin, 7
  - \_interface\_plugin, 9
  - \_output\_plugin, 10
  - \_scope\_plugin, 13

- nr\_channels
  - \_input\_object, 6
- nr\_frames
  - \_input\_object, 6
  - \_input\_plugin, 8
- nr\_tracks
  - \_input\_object, 6
  - \_input\_plugin, 8
- object\_mutex
  - \_input\_object, 6
- open
  - \_input\_plugin, 8
  - \_output\_plugin, 10
- output\_close\_type
  - output\_plugin.h, 28
- output\_get\_latency\_type
  - output\_plugin.h, 28
- output\_get\_queue\_count\_type
  - output\_plugin.h, 28
- output\_init\_type
  - output\_plugin.h, 28
- output\_open\_type
  - output\_plugin.h, 28
- output\_plugin
  - output\_plugin.h, 28
- output\_plugin.h, 27
  - output\_close\_type, 28
  - output\_get\_latency\_type, 28
  - output\_get\_queue\_count\_type, 28
  - output\_init\_type, 28
  - output\_open\_type, 28
  - output\_plugin, 28
  - OUTPUT\_PLUGIN\_BASE\_VERSION, 28
  - output\_plugin\_info\_type, 28
  - OUTPUT\_PLUGIN\_VERSION, 28
  - output\_set\_buffer\_type, 28
  - output\_set\_sample\_rate\_type, 28
  - output\_start\_callbacks\_type, 28
  - output\_version\_type, 28
  - output\_write\_type, 28
- OUTPUT\_PLUGIN\_BASE\_VERSION
- output\_plugin.h, 28
- output\_plugin\_info\_type
  - output\_plugin.h, 28
- OUTPUT\_PLUGIN\_VERSION
  - output\_plugin.h, 28
- output\_set\_buffer\_type
  - output\_plugin.h, 28
- output\_set\_sample\_rate\_type
  - output\_plugin.h, 28
- output\_start\_callbacks\_type
  - output\_plugin.h, 28
- output\_version\_type
  - output\_plugin.h, 28
- output\_write\_type
  - output\_plugin.h, 28
- P\_BUFFERING
  - input\_plugin.h, 20
- P\_FILEBASED
  - input\_plugin.h, 20
- P\_PERFECTSEEK
  - input\_plugin.h, 20
- P\_REENTRANT
  - input\_plugin.h, 21
- P\_SEEK
  - input\_plugin.h, 21
- P\_STREAMBASED
  - input\_plugin.h, 21
- path
  - \_stream\_info, 16
- play\_frame
  - \_input\_plugin, 8
- plugin\_handle
  - \_input\_plugin, 8
- ready
  - \_input\_object, 6
- running
  - \_interface\_plugin, 9
  - \_scope\_plugin, 14
- sample\_rate
  - \_input\_plugin, 8
  - \_stream\_info, 16
- SCOPE\_BG\_BLUE
  - scope\_plugin.h, 29



SCOPE\_BG\_GREEN  
    scope\_plugin.h, 29  
SCOPE\_BG\_RED  
    scope\_plugin.h, 29  
scope\_init\_type  
    scope\_plugin.h, 30  
SCOPE\_NICE  
    scope\_plugin.h, 30  
scope\_plugin  
    scope\_plugin.h, 31  
scope\_plugin.h, 29  
    SCOPE\_BG\_BLUE, 29  
    SCOPE\_BG\_GREEN, 29  
    SCOPE\_BG\_RED, 29  
    scope\_init\_type, 30  
    SCOPE\_NICE, 30  
    scope\_plugin, 31  
    SCOPE\_PLUGIN\_BASE\_-  
        VERSION, 30  
    scope\_plugin\_info\_type, 31  
    SCOPE\_PLUGIN\_VERSION,  
        30  
    scope\_running\_type, 31  
    scope\_set\_data\_type, 31  
    scope\_set\_fft\_type, 31  
    scope\_shutdown\_type, 31  
    SCOPE\_SLEEP, 30  
    scope\_start\_type, 32  
    scope\_stop\_type, 32  
    scope\_version\_type, 32  
SCOPE\_PLUGIN\_BASE\_VERSION  
    scope\_plugin.h, 30  
scope\_plugin\_info\_type  
    scope\_plugin.h, 31  
SCOPE\_PLUGIN\_VERSION  
    scope\_plugin.h, 30  
scope\_running\_type  
    scope\_plugin.h, 31  
scope\_set\_data\_type  
    scope\_plugin.h, 31  
scope\_set\_fft\_type  
    scope\_plugin.h, 31  
scope\_shutdown\_type  
    scope\_plugin.h, 31  
SCOPE\_SLEEP  
    scope\_plugin.h, 30  
scope\_start\_type  
    scope\_plugin.h, 32  
scope\_stop\_type  
    scope\_plugin.h, 32  
scope\_version\_type  
    scope\_plugin.h, 32  
set\_buffer  
    \_output\_plugin, 11  
set\_data  
    \_scope\_plugin, 14  
set\_fft  
    \_scope\_plugin, 14  
set\_sample\_rate  
    \_output\_plugin, 11  
shutdown  
    \_input\_plugin, 8  
    \_scope\_plugin, 14  
start  
    \_interface\_plugin, 9  
    \_scope\_plugin, 14  
start\_callbacks  
    \_output\_plugin, 11  
status  
    \_stream\_info, 16  
stop  
    \_interface\_plugin, 9  
    \_scope\_plugin, 14  
stream\_info  
    \_input\_plugin, 8  
    input\_plugin.h, 24  
stream\_type  
    \_stream\_info, 15  
title  
    \_stream\_info, 15  
track  
    \_stream\_info, 16  
track\_seek  
    \_input\_plugin, 8  
tracks  
    \_stream\_info, 16  
version  
    \_input\_plugin, 7  
    \_interface\_plugin, 9  
    \_output\_plugin, 10

\_scope\_plugin, [13](#)

write

  \_output\_plugin, [11](#)

year

  \_stream\_info, [16](#)