

AlsaPlayer

0.99.80

Generated by Doxygen 1.7.5.1

Mon Mar 12 2012 02:17:57

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	_input_object Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Field Documentation	5
3.1.2.1	flags	6
3.1.2.2	frame_size	6
3.1.2.3	local_data	6
3.1.2.4	nr_channels	6
3.1.2.5	nr_frames	6
3.1.2.6	nr_tracks	6
3.1.2.7	object_mutex	6
3.1.2.8	path	6
3.1.2.9	ready	6
3.2	_input_plugin Struct Reference	7
3.2.1	Field Documentation	7
3.2.1.1	author	7
3.2.1.2	can_handle	8

3.2.1.3	channels	8
3.2.1.4	close	8
3.2.1.5	flags	8
3.2.1.6	frame_seek	8
3.2.1.7	frame_size	8
3.2.1.8	frame_to_sec	8
3.2.1.9	handle	8
3.2.1.10	init	8
3.2.1.11	name	8
3.2.1.12	nr_frames	8
3.2.1.13	nr_tracks	8
3.2.1.14	open	8
3.2.1.15	play_frame	8
3.2.1.16	plugin_handle	8
3.2.1.17	sample_rate	8
3.2.1.18	shutdown	8
3.2.1.19	stream_info	8
3.2.1.20	track_seek	9
3.2.1.21	version	9
3.3	_interface_plugin Struct Reference	9
3.3.1	Field Documentation	9
3.3.1.1	author	9
3.3.1.2	close	9
3.3.1.3	handle	9
3.3.1.4	init	9
3.3.1.5	name	9
3.3.1.6	running	9
3.3.1.7	start	9
3.3.1.8	stop	10
3.3.1.9	version	10
3.4	_output_plugin Struct Reference	10

3.4.1	Field Documentation	10
3.4.1.1	author	10
3.4.1.2	close	10
3.4.1.3	get_latency	10
3.4.1.4	get_queue_count	11
3.4.1.5	init	11
3.4.1.6	name	11
3.4.1.7	open	11
3.4.1.8	set_buffer	11
3.4.1.9	set_sample_rate	11
3.4.1.10	start_callbacks	12
3.4.1.11	version	12
3.4.1.12	write	12
3.5	_scope_plugin Struct Reference	12
3.5.1	Detailed Description	13
3.5.2	Field Documentation	13
3.5.2.1	author	13
3.5.2.2	handle	13
3.5.2.3	init	13
3.5.2.4	name	13
3.5.2.5	running	13
3.5.2.6	set_data	13
3.5.2.7	set_fft	13
3.5.2.8	shutdown	14
3.5.2.9	start	14
3.5.2.10	stop	14
3.5.2.11	version	14
4	File Documentation	15
4.1	input_plugin.h File Reference	15
4.1.1	Define Documentation	16

4.1.1.1	INPUT_PLUGIN_BASE_VERSION	16
4.1.1.2	INPUT_PLUGIN_VERSION	16
4.1.1.3	P_BUFFERING	16
4.1.1.4	P_FILEBASED	16
4.1.1.5	P_PERFECTSEEK	17
4.1.1.6	P_REENTRANT	17
4.1.1.7	P_SEEK	17
4.1.1.8	P_STREAMBASED	17
4.1.2	Typedef Documentation	17
4.1.2.1	input_can_handle_type	17
4.1.2.2	input_channels_type	17
4.1.2.3	input_close_type	17
4.1.2.4	input_flags_type	18
4.1.2.5	input_frame_seek_type	18
4.1.2.6	input_frame_size_type	18
4.1.2.7	input_frame_to_sec_type	18
4.1.2.8	input_init_type	18
4.1.2.9	input_nr_frames_type	18
4.1.2.10	input_nr_tracks_type	19
4.1.2.11	input_object	19
4.1.2.12	input_open_type	19
4.1.2.13	input_play_frame_type	19
4.1.2.14	input_plugin	19
4.1.2.15	input_plugin_handle_type	19
4.1.2.16	input_plugin_info_type	19
4.1.2.17	input_sample_rate_type	20
4.1.2.18	input_shutdown_type	20
4.1.2.19	input_stream_info_type	20
4.1.2.20	input_track_seek_type	20
4.1.2.21	input_version_type	20
4.2	interface_plugin.h File Reference	20

4.2.1	Define Documentation	21
4.2.1.1	INTERFACE_PLUGIN_BASE_VERSION	21
4.2.1.2	INTERFACE_PLUGIN_VERSION	21
4.2.2	Typedef Documentation	21
4.2.2.1	interface_close_type	21
4.2.2.2	interface_init_type	21
4.2.2.3	interface_plugin	21
4.2.2.4	interface_plugin_info_type	21
4.2.2.5	interface_running_type	21
4.2.2.6	interface_start_type	21
4.2.2.7	interface_stop_type	21
4.2.2.8	interface_version_type	21
4.3	output_plugin.h File Reference	22
4.3.1	Define Documentation	22
4.3.1.1	OUTPUT_PLUGIN_BASE_VERSION	22
4.3.1.2	OUTPUT_PLUGIN_VERSION	22
4.3.2	Typedef Documentation	22
4.3.2.1	output_close_type	22
4.3.2.2	output_get_latency_type	22
4.3.2.3	output_get_queue_count_type	22
4.3.2.4	output_init_type	23
4.3.2.5	output_open_type	23
4.3.2.6	output_plugin	23
4.3.2.7	output_plugin_info_type	23
4.3.2.8	output_set_buffer_type	23
4.3.2.9	output_set_sample_rate_type	23
4.3.2.10	output_start_callbacks_type	23
4.3.2.11	output_version_type	23
4.3.2.12	output_write_type	23
4.4	scope_plugin.h File Reference	23
4.4.1	Define Documentation	24

4.4.1.1	SCOPE_BG_BLUE	24
4.4.1.2	SCOPE_BG_GREEN	24
4.4.1.3	SCOPE_BG_RED	24
4.4.1.4	SCOPE_NICE	24
4.4.1.5	SCOPE_PLUGIN_BASE_VERSION	24
4.4.1.6	SCOPE_PLUGIN_VERSION	24
4.4.1.7	SCOPE_SLEEP	25
4.4.2	Typedef Documentation	25
4.4.2.1	scope_init_type	25
4.4.2.2	scope_plugin	25
4.4.2.3	scope_plugin_info_type	25
4.4.2.4	scope_running_type	25
4.4.2.5	scope_set_data_type	25
4.4.2.6	scope_set_fft_type	26
4.4.2.7	scope_shutdown_type	26
4.4.2.8	scope_start_type	26
4.4.2.9	scope_stop_type	26
4.4.2.10	scope_version_type	26

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

_input_object	5
_input_plugin	7
_interface_plugin	9
_output_plugin	10
_scope_plugin	12

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

input_plugin.h	15
interface_plugin.h	20
output_plugin.h	22
scope_plugin.h	23

Chapter 3

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`
- char * `path`
- 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::flags`

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

3.1.2.2 `int _input_object::frame_size`

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

3.1.2.3 `void* _input_object::local_data`

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

3.1.2.4 `int _input_object::nr_channels`

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

3.1.2.5 `int _input_object::nr_frames`

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

3.1.2.6 `int _input_object::nr_tracks`

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

3.1.2.7 `pthread_mutex_t _input_object::object_mutex`

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

3.1.2.8 `char* _input_object::path`

Path of the currently played file

3.1.2.9 `int _input_object::ready`

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

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 `char* _input_plugin::author`

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

3.2.1.2 `input_can_handle_type _input_plugin::can_handle`

3.2.1.3 `input_channels_type _input_plugin::channels`

3.2.1.4 `input_close_type _input_plugin::close`

3.2.1.5 `input_flags_type _input_plugin::flags`

Fixed flags for the plugin (P_*)

3.2.1.6 `input_frame_seek_type _input_plugin::frame_seek`

3.2.1.7 `input_frame_size_type _input_plugin::frame_size`

3.2.1.8 `input_frame_to_sec_type _input_plugin::frame_to_sec`

3.2.1.9 `void* _input_plugin::handle`

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

3.2.1.10 `input_init_type _input_plugin::init`

3.2.1.11 `char* _input_plugin::name`

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

3.2.1.12 `input_nr_frames_type _input_plugin::nr_frames`

3.2.1.13 `input_nr_tracks_type _input_plugin::nr_tracks`

3.2.1.14 `input_open_type _input_plugin::open`

3.2.1.15 `input_play_frame_type _input_plugin::play_frame`

3.2.1.16 `input_plugin_handle_type _input_plugin::plugin_handle`

3.2.1.17 `input_sample_rate_type _input_plugin::sample_rate`

3.2.1.18 `input_shutdown_type _input_plugin::shutdown`

3.2.1.19 `input_stream_info_type _input_plugin::stream_info`

3.2.1.20 `input_track_seek_type _input_plugin::track_seek`

3.2.1.21 `input_version_type _input_plugin::version`

Must be set to `INPUT_PLUGIN_VERSION`

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 `char* _interface_plugin::author`

3.3.1.2 `interface_close_type _interface_plugin::close`

3.3.1.3 `void* _interface_plugin::handle`

3.3.1.4 `interface_init_type _interface_plugin::init`

3.3.1.5 `char* _interface_plugin::name`

3.3.1.6 `interface_running_type _interface_plugin::running`

3.3.1.7 `interface_start_type _interface_plugin::start`

3.3.1.8 `interface_stop_type _interface_plugin::stop`

3.3.1.9 `interface_version_type _interface_plugin::version`

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 `char* _output_plugin::author`

Author of the plugin

3.4.1.2 `output_close_type _output_plugin::close`

Close the output plugin

3.4.1.3 `output_get_latency_type _output_plugin::get_latency`

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

3.4.1.4 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.

3.4.1.5 output_init_type _output_plugin::init

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

3.4.1.6 char* _output_plugin::name

Name of output plugin

3.4.1.7 output_open_type _output_plugin::open

Parameters

<i>path</i>	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.

3.4.1.8 output_set_buffer_type _output_plugin::set_buffer

Parameters

<i>frag_size</i>	Fragment size to use (in bytes)
<i>frag_count</i>	Fragment count to use (in bytes)
<i>channels</i>	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.

3.4.1.9 output_set_sample_rate_type _output_plugin::set_sample_rate

Parameters

<i>rate</i>	Sample rate to use
-------------	--------------------

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

3.4.1.10 `output_start_callbacks_type _output_plugin::start_callbacks`

Parameters

<i>data</i>	pointer to bufs structure in <code>AlsaNode</code>
-------------	--

This function is used for callback based plugins like JACK

3.4.1.11 `output_version_type _output_plugin::version`

Version of output plugin. Must be `OUTPUT_PLUGIN_VERSION`

3.4.1.12 `output_write_type _output_plugin::write`

Parameters

<i>data</i>	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.

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 `char* _scope_plugin::author`

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

3.5.2.2 `void* _scope_plugin::handle`

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

3.5.2.3 `scope_init_type _scope_plugin::init`

Should point to the implementation of your `init()` function. Required by the HOST.

3.5.2.4 `char* _scope_plugin::name`

Point to a character array with the name of the scope

3.5.2.5 `scope_running_type _scope_plugin::running`

Should point to the implementation of your `running()` function. Required by the HOST.

3.5.2.6 `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`.

3.5.2.7 `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.

3.5.2.8 `scope_shutdown_type _scope_plugin::shutdown`

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

3.5.2.9 `scope_start_type _scope_plugin::start`

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

3.5.2.10 `scope_stop_type _scope_plugin::stop`

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

3.5.2.11 `scope_version_type _scope_plugin::version`

Set to `SCOPE_PLUGIN_VERSION`

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

- [scope_plugin.h](#)

Chapter 4

File Documentation

4.1 input_plugin.h File Reference

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

Data Structures

- struct [_input_object](#)
- 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 struct [_input_object](#) [input_object](#)
- 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 struct [_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

<i>path</i>	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

<i>obj</i>	input object
------------	--------------

Returns number of channels in the stream

4.1.2.3 `typedef void(* input_close_type)(input_object *obj)`

Parameters

<i>obj</i>	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

<i>obj</i>	input object
<i>frame</i>	Seek to a specific frame number

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

Parameters

<i>obj</i>	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

<i>obj</i>	input object
<i>frame</i>	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

<i>obj</i>	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

<i>obj</i>	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

<i>obj</i>	input object
<i>path</i>	path of stream to open

Open stream

4.1.2.13 `typedef int(* input_play_frame_type)(input_object *obj, char *buffer)`

Parameters

<i>obj</i>	input object
<i>buffer</i>	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

<i>obj</i>	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

<i>obj</i>	input object
<i>info</i>	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

4.2 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 struct `_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 struct [_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 struct [_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

<i>buffer</i>	pointer to buffer data
<i>count</i>	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

<i>buffer</i>	buffer with FFT values
<i>samples</i>	number of FFT values per channel
<i>channels</i>	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.