JACK-AUDIO-CONNECTION-KIT

0.120.1

Generated by Doxygen 1.7.1

Mon Sep 13 2010 12:29:28

Contents

Chapter 1

JACK Audio Connection Kit

1.1 Introduction

JACK is a low-latency audio server, written for any operating system that is reasonably POSIX compliant. It currently exists for Linux, OS X, Solaris, FreeBSD and Windows. It can connect several client applications to an audio device, and allow them to share audio with each other. Clients can run as separate processes like normal applications, or within the JACK server as "plugins".

JACK was designed from the ground up for professional audio work, and its design focuses on two key areas: synchronous execution of all clients, and low latency operation.

See also

<http://jackaudio.org>

1.2 JACK Overview

Traditionally it has been hard if not impossible to write audio applications that can share data with each other. In addition, configuring and managing audio interface hardware has often been one of the most complex aspect of writing audio software.

JACK changes all this by providing an API that does several things:

1. provides a high level abstraction for programmers that removes the audio interface hardware from the picture and allows them to concentrate on the core functionality of their software.

2. allows applications to send and receive audio data to/from each other as well as the audio interface. There is no difference in how an application sends or receives data

regardless of whether it comes from/goes to another application or an audio interface.

For programmers with experience of several other audio APIs such as PortAudio, Apple's CoreAudio, Steinberg's VST and ASIO as well as many others, JACK presents a familiar model: your program provides a "callback" function that will be executed at the right time. Your callback can send and receive data as well as do other signal processing tasks. You are not responsible for managing audio interfaces or threading, and there is no "format negotiation": all audio data within JACK is represented as 32 bit floating point values.

For those with experiences rooted in the Unix world, JACK presents a somewhat unfamiliar API. Most Unix APIs are based on the read/write model spawned by the "everything is a file" abstraction that Unix is rightly famous for. The problem with this design is that it fails to take the realtime nature of audio interfaces into account, or more precisely, it fails to force application developers to pay sufficient attention to this aspect of their task. In addition, it becomes rather difficult to facilitate inter-application audio routing when different programs are not all running synchronously.

Using JACK within your program is very simple, and typically consists of just:

- calling jack_client_open() to connect to the JACK server.
- registering "ports" to enable data to be moved to and from your application.
- registering a "process callback" which will be called at the right time by the JACK server.
- telling JACK that your application is ready to start processing data.

There is a lot more that you can do with JACK's interfaces, but for many applications, this is all that is needed. The simple_client.c example demonstrates a complete (simple!) JACK application that just copies the signal arriving at its input port to its output port. Similarly, inprocess.c shows how to write an internal client "plugin" that runs within the JACK server process.

1.3 Reference

The JACK programming interfaces are described in several header files. We present them here broken into useful categories to make the API a little clearer:

- Creating & manipulating clients
- Setting Client Callbacks
- Creating and managing client threads
- Controlling & querying JACK server operation

- Creating & manipulating ports
- Looking up ports
- Handling time
- Transport and Timebase control
- Controlling error/information output
- The non-callback API
- Reading and writing MIDI data

The full API is described in:

- <jack/jack.h> is the main JACK interface.
- <jack/statistics.h> provides interfaces for monitoring the performance of a running JACK server.
- <jack/intclient.h> allows loading and unloading JACK internal clients.
- <jack/ringbuffer.h> defines a simple API for using lock-free ringbuffers. These are a good way to pass data between threads, when streaming realtime data to slower media, like audio file playback or recording.
- <jack/transport.h> defines a simple transport control mechanism for starting, stopping and repositioning clients. This is described in the JACK Transport Design document.
- <jack/types.h> defines the main JACK data types.
- <jack/thread.h> functions standardize thread creation for JACK and its clients.
- <jack/midiport.h> functions to handle reading and writing of MIDI data to a port

In addition, the tools directory provides numerous examples of simple JACK clients that nevertheless use the API to do something useful. It includes

- a metronome.
- a recording client that can capture any number of channels from any JACK sources and store them as an audio file.
- command line clients to control the transport mechanism, change the buffer size and more.
- commands to load and unload JACK internal clients.
- tools for checking the status of a running JACK system.

and many more.

1.4 Porting

JACK is designed to be portable to any system supporting the relevant POSIX and ANSI C standards. It currently runs under GNU/Linux, Mac OS X and Berkeley Unix on several different processor architectures. If you want to port JACK to another platform, please read the Porting JACK document.

1.5 License

Chapter 2

JACK Transport Design

The JACK Audio Connection Kit provides simple transport interfaces for starting, stopping and repositioning a set of clients. This document describes the overall design of these interfaces, their detailed specifications are in <jack/transport.h>

2.1 Requirements

- We need sample-level accuracy for transport control. This implies that the transport client logic has to be part of the realtime process chain.

- We don't want to add another context switch. So, the transport client logic has to run in the context of the client's process thread. To avoid making an extra pass through the process graph, no transport changes take effect until the following process cycle. That way, the transport info is stable throughout each cycle.

- We want to allow multiple clients to change the transport state. This is mostly a usability issue. Any client can start or stop playback, or seek to a new location. The user need not switch windows to accomplish these tasks.

- We want a way for clients with heavyweight state to sync up when the user presses "play", before the transport starts rolling.

- We want to provide for ongoing binary compatibility as the transport design evolves.

2.2 Overview

The former transport master role has been divided into two layers:

- Timebase Master - counting beats, frames, etc. on every cycle.
- Transport Control - start, stop and reposition the playback.

Existing transport clients continue to work in compatibility mode. But, old-style timebase masters will no longer control the transport.

2.3 Timebase Master

The timebase master continuously updates extended position information, counting beats, timecode, etc. Without this extended information, there is no need for this function. There is at most one master active at a time. If no client is registered as timebase master, frame numbers will be the only position information available.

The timebase master registers a callback that updates position information while the transport is rolling. Its output affects the following process cycle. This function is called immediately after the process callback in the same thread whenever the transport is rolling, or when any client has set a new position in the previous cycle. The first cycle after jack_set_timebase_callback() is also treated as a new position, or the first cycle after jack_activate() if the client had been inactive.

```
typedef int  (*JackTimebaseCallback)(jack_transport_state_t state,
                                     jack_nframes_t nframes,
                                     jack_position_t *pos,
                                     int new_pos,
                                     void *arg);
```

When a new client takes over, the former timebase callback is no longer called. Taking over the timebase may be done conditionally, in which case the takeover fails when there is a master already. The existing master can release it voluntarily, if desired.

```
int  jack_set_timebase_callback (jack_client_t *client,
                                 int conditional,
```

```
                                    JackTimebaseCallback timebase_callback,
                                    void *arg);

int  jack_release_timebase(jack_client_t *client);
```

If the timebase master releases the timebase or exits the JACK graph for any reason, the JACK engine takes over at the start of the next process cycle. The transport state does not change. If rolling, it continues to play, with frame numbers as the only available position information.

2.4 Transport Control

The JACK engine itself manages stopping and starting of the transport. Any client can make transport control requests at any time. These requests take effect no sooner than the next process cycle, sometimes later. The transport state is always valid, initially it is JackTransportStopped.

```
void jack_transport_start (jack_client_t *client);
void jack_transport_stop (jack_client_t *client);
```

The engine handles polling of slow-sync clients. When someone calls jack_transport_start(), the engine resets the poll bits and changes to a new state, JackTransportStarting. The *sync_callback* function for each slow-sync client will be invoked in the JACK process thread while the transport is starting. If it has not already done so, the client needs to initiate a seek to reach the starting position. The *sync_callback* returns false until the seek completes and the client is ready to play. When all slow-sync clients are ready, the state changes to JackTransportRolling.

```
typedef int  (*JackSyncCallback)(jack_transport_state_t state,
                                 jack_position_t *pos, void *arg);
```

This callback is a realtime function that runs in the JACK process thread.

```
int  jack_set_sync_callback (jack_client_t *client,
                             JackSyncCallback sync_callback, void *arg);
```

Clients that don't declare a *sync_callback* are assumed to be ready immediately, any time the transport wants to start. If a client no longer requires slow-sync processing, it can set its *sync_callback* to NULL.

```
int  jack_set_sync_timeout (jack_client_t *client,
                            jack_time_t usecs);
```

There must be a *timeout* to prevent unresponsive slow-sync clients from completely halting the transport mechanism. Two seconds is the default. When this *timeout* expires, the transport will start rolling, even if some slow-sync clients are still unready.

The *sync_callback* for these clients continues being invoked, giving them an opportunity to catch up.

```
int  jack_transport_reposition (jack_client_t *client,
                                jack_position_t *pos);
int  jack_transport_locate (jack_client_t *client,
                            jack_nframes_t frame);
```

These request a new transport position. They can be called at any time by any client. Even the timebase master must use them. If the request is valid, it goes into effect in two process cycles. If there are slow-sync clients and the transport is already rolling, it will enter the JackTransportStarting state and begin invoking their *sync_callbacks* until ready.

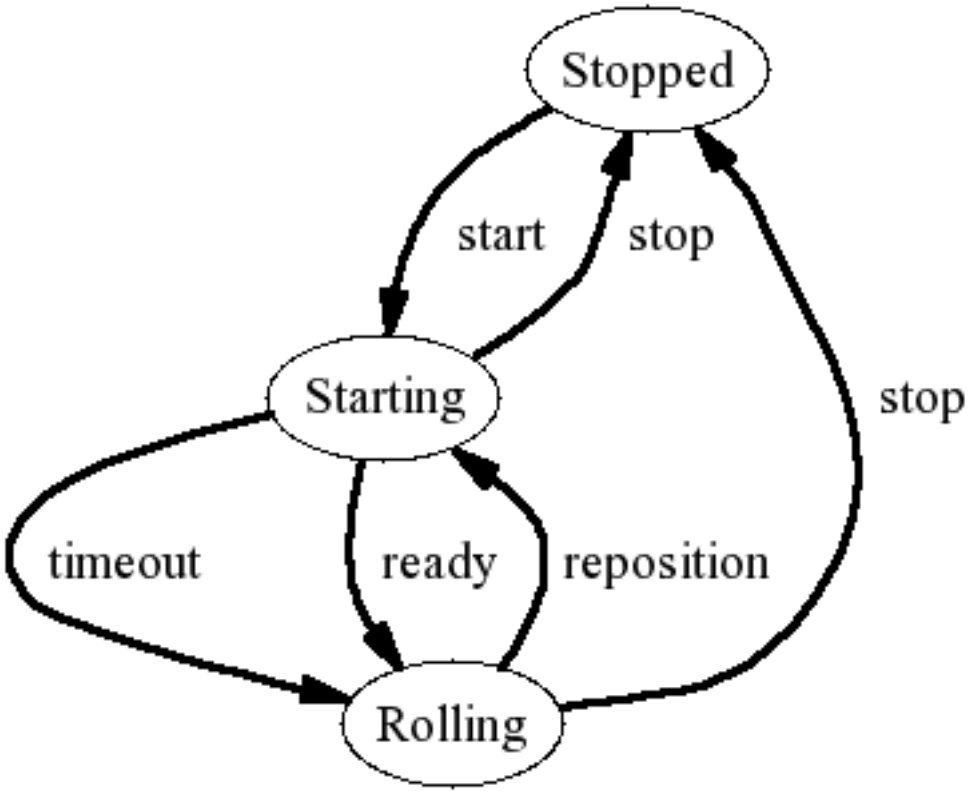

Figure 2.1: Transport State Transition Diagram

2.5 Transport Clients

Transport clients were formerly known as "transport slaves". We want to make it easy for almost every JACK client to be a transport client.

```
jack_transport_state_t jack_transport_query (jack_client_t *client,
                                             jack_position_t *pos);
```

This function can be called from any thread. If called from the process thread, *pos* corresponds to the first frame of the current cycle and the state returned is valid for the entire cycle.

2.6 Compatibility

During the transition period we will support the old-style interfaces in compatibility mode as deprecated interfaces. This compatibility is not 100%, there are limitations.

The main reasons for doing this are:

- facilitate testing with clients that already have transport support
- provide a clean migration path, so application developers are not discouraged from supporting the transport interface

These deprecated interfaces continue to work:

```
typedef struct jack_transport_info_t;

void jack_get_transport_info (jack_client_t *client,
                              jack_transport_info_t *tinfo);
```

Unfortunately, the old-style timebase master interface cannot coexist cleanly with such new features as jack_transport_locate() and slow-sync clients. So, these interfaces are only present as stubs:

```
void jack_set_transport_info (jack_client_t *client,
                              jack_transport_info_t *tinfo);
int  jack_engine_takeover_timebase (jack_client_t *);
```

For compatibility with future changes, it would be good to avoid structures entirely. Nevertheless, the jack_position_t structure provides a convenient way to collect timebase information in several formats that clearly all refer to a single moment. To minimize future binary compatibility problems, this structure has some padding at the end, making it possible to extend it without necessarily breaking compatibility. New fields can be allocated from the padding area, with access controlled by newly defined valid bits, all of which are currently forced to zero. That allows the structure size and offsets to remain constant.

2.7 Issues Not Addressed

This design currently does not address several issues. This means they will probably not be included in JACK release 1.0.

- variable speed
- reverse play
- looping

Chapter 3

Porting JACK

The JACK Audio Connection Kit is designed to be portable to any system supporting the relevant POSIX and C language standards. It currently works with GNU/Linux and Mac OS X on several different processor architectures. This document describes the steps needed to port JACK to another platform, and the methods used to provide portability.

3.1 Requirements

- Each platform should build directly from CVS or from a tarball using the GNU ./configure tools. Platform-specific toolsets can by used for development, but the GNU tools should at least work for basic distribution and configuration.

- For long-term maintainability we want to minimize the use of conditional compilation in source files.

- We should provide generic versions of all system-dependent headers, so platforms need only provide those they modify.

- In some cases, operating system-specific information must be able to override processor-specific data.

3.2 Overview

JACK relies on two types of platform-specific headers:

- Operating System Dependencies
- Processor Dependencies

OS-specific headers take precedence over CPU-specific headers.

The JACK `configure.host` script and its system-dependent header directories were adapted from the `libstdc++-v3` component of the GNU Compiler Collective, <http://gcc.gnu.org>.

3.3 C Language Dependencies

JACK is written to conform with C99, as defined in International Standard ISO/IEC 9899:1999. Because many existing compilers do not fully support this standard, some new features should be avoided for portablility reasons. For example, variables should not be declared in the middle of a compound statement, because many compilers still cannot handle that language extension.

3.4 Operating System Dependencies

JACK is written for a POSIX environment compliant with IEEE Std 1003.1-2001, ISO/IEC 9945:2003, including the POSIX Threads Extension (1003.1c-1995) and the Realtime and Realtime Threads feature groups. When some needed POSIX feature is missing on a platform, the preferred solution is to provide a substitute, as with the `fakepoll.c` implementation for Mac OS X.

Whenever possible, OS dependencies should be auto-detected by `configure`. Sometimes they can be isolated in OS-specific header files, found in subdirectories of `config/os` and referenced with a <sysdeps/xxx.h> name.

If conditional compilation must be used in mainline platform-independent code, avoid using the system name. Instead, `#define` a descriptive name in <config.h>, and test it like this:

```
\#ifdef JACK_USE_MACH_THREADS
        allocate_mach_serverport(engine, client);
        client->running = FALSE;
\#endif
```

Be sure to place any generic implementation alternative in the `#else` or use an `#ifndef`, so no other code needs to know your conditional labels.

3.5 Processor Dependencies

JACK uses some low-level machine operations for thread-safe updates to shared memory. A low-level implementation of <sysdeps/atomicity.h> is provided for every target processor architecture. There is also a generic implementation using POSIX spin locks, but that is not a good enough solution for serious use.

The GCC package provides versions that work on most modern hardware. We've tried to keep things as close to the original as possible, while removing a bunch of os-specific files that didn't seem relevant. A primary goal has been to avoid changing the CPU-dependent <sysdeps/atomicity.h> headers.

The relevant GCC documentation provides some helpful background, especially the atomicity.h discussion at <http://gcc.gnu.org/onlinedocs/porting/Thread-safety.html>.

3.6 Issues Not Addressed

- Cross-compilation has not been tested, or even thought through in much detail. The *host* is the system on which JACK will run. This may differ from the *build* system doing the compilation. These are selected using the standard ./configure options --host and --build. Usually, ./config.guess can print the appropriate canonical name for any system on which it runs.

- Platform-specific build tools like Apple's Project Builder are not well-supported.

Chapter 4

Deprecated List

Global jack_engine_takeover_timebase(jack_client_t *) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
This function still exists for compatibility with the earlier transport interface, but it does nothing. Instead, see transport.h and use jack_set_timebase_callback().

Global jack_get_transport_info(jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_DEPR
This is for compatibility with the earlier transport interface. Use jack_transport_-query(), instead.

Global jack_internal_client_close(const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
Please use jack_internal_client_load().

Global jack_internal_client_new(const char *client_name, const char *load_name, const char *load_init) JACK_OPTI
Please use jack_internal_client_load().

Global jack_port_tie(jack_port_t *src, jack_port_t *dst) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
This function will be removed from a future version of JACK. Do not use it. There is no replacement. It has turned out to serve essentially no purpose in real-life JACK clients.

Global jack_port_untie(jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
This function will be removed from a future version of JACK. Do not use it. There is no replacement. It has turned out to serve essentially no purpose in real-life JACK clients.

Global jack_set_transport_info(jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_W
This function still exists for compatibility with the earlier transport interface, but it does nothing. Instead, define a JackTimebaseCallback.

Class jack_transport_info_t This is for compatibility with the earlier transport interface. Use the jack_position_t struct, instead.

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 Creating & manipulating clients

Functions

- jack_client_t * jack_client_open (const char *client_name, jack_options_t options, jack_status_t *status,...) JACK_OPTIONAL_WEAK_EXPORT
- jack_client_t * jack_client_new (const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_client_close (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_name_size (void) JACK_OPTIONAL_WEAK_EXPORT
- char * jack_get_client_name (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_internal_client_new (const char *client_name, const char *load_name, const char *load_init) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- void jack_internal_client_close (const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_activate (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_deactivate (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- pthread_t jack_client_thread_id (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT

8.1.1 Detailed Description

Note: More documentation can be found in jack/types.h.

8.1.2 Function Documentation

8.1.2.1 int jack_activate (jack_client_t * *client*)

Tell the Jack server that the program is ready to start processing audio.

Returns

0 on success, otherwise a non-zero error code

Referenced by jack_initialize(), and main().

8.1.2.2 int jack_client_close (jack_client_t * *client*)

Disconnects an external client from a JACK server.

Returns

0 on success, otherwise a non-zero error code

Referenced by main().

8.1.2.3 int jack_client_name_size (void)

Returns

the maximum number of characters in a JACK client name including the final NULL character. This value is a constant.

8.1.2.4 jack_client_t* jack_client_new (const char * *client_name*)

THIS FUNCTION IS DEPRECATED AND SHOULD NOT BE USED IN NEW JACK CLIENTS

8.1.2.5 jack_client_t* jack_client_open (const char * *client_name*, jack_options_t *options*, jack_status_t * *status*, ...)

Open an external client session with a JACK server. This interface is more complex but more powerful than jack_client_new(). With it, clients may choose which of several servers to connect, and control whether and how to start the server automatically, if it was not already running. There is also an option for JACK to generate a unique client name, when necessary.

Parameters

client_name of at most jack_client_name_size() characters. The name scope is local to each server. Unless forbidden by the JackUseExactName option, the server will modify this name to create a unique variant, if needed.

options formed by OR-ing together JackOptions bits. Only the JackOpenOptions bits are allowed.

status (if non-NULL) an address for JACK to return information from the open operation. This status word is formed by OR-ing together the relevant JackStatus bits.

Optional parameters: depending on corresponding [*options* bits] additional parameters may follow *status* (in this order).

- [JackServerName] *(char *) server_name* selects from among several possible concurrent server instances. Server names are unique to each user. If unspecified, use "default" unless $JACK_DEFAULT_SERVER is defined in the process environment.

Returns

Opaque client handle if successful. If this is NULL, the open operation failed, **status* includes JackFailure and the caller is not a JACK client.

Referenced by main().

8.1.2.6 pthread_t jack_client_thread_id (jack_client_t *)

Returns

the pthread ID of the thread running the JACK client side code.

8.1.2.7 int jack_deactivate (jack_client_t * *client*)

Tell the Jack server to remove this *client* from the process graph. Also, disconnect all ports belonging to it, since inactive clients have no port connections.

Returns

0 on success, otherwise a non-zero error code

8.1.2.8 char* jack_get_client_name (jack_client_t * *client*)

Returns

pointer to actual client name. This is useful when JackUseExactName is not specified on open and JackNameNotUnique status was returned. In that case, the actual name will differ from the *client_name* requested.

Referenced by main().

8.1.2.9 void jack_internal_client_close (const char * *client_name*)

Remove an internal client from a JACK server.

Deprecated

Please use jack_internal_client_load().

8.1.2.10 int jack_internal_client_new (const char * *client_name*, const char * *load_name*, const char * *load_init*)

Load an internal client into the Jack server.

Internal clients run inside the JACK server process. They can use most of the same functions as external clients. Each internal client must declare jack_initialize() and jack_finish() entry points, called at load and unload times. See inprocess.c for an example of how to write an internal client.

Deprecated

Please use jack_internal_client_load().

Parameters

client_name of at most jack_client_name_size() characters.

load_name of a shared object file containing the code for the new client.

load_init an arbitary string passed to the jack_initialize() routine of the new client (may be NULL).

Returns

0 if successful.

8.2 The non-callback API

Functions

- jack_nframes_t jack_thread_wait (jack_client_t *, int status) JACK_-OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_cycle_wait (jack_client_t *client) JACK_OPTIONAL_-WEAK_EXPORT
- void jack_cycle_signal (jack_client_t *client, int status) JACK_OPTIONAL_-WEAK_EXPORT
- int jack_set_process_thread (jack_client_t *client, JackThreadCallback fun, void *arg) JACK_OPTIONAL_WEAK_EXPORT

8.2.1 Function Documentation

8.2.1.1 void jack_cycle_signal (jack_client_t * *client,* int *status*)

Signal next clients in the graph.

Parameters

client - pointer to a JACK client structure

status - if non-zero, calling thread should exit

8.2.1.2 jack_nframes_t jack_cycle_wait (jack_client_t * *client*)

Wait until this JACK client should process data.

Parameters

client - pointer to a JACK client structure

Returns

the number of frames of data to process

8.2.1.3 int jack_set_process_thread (jack_client_t * *client,* JackThreadCallback *fun,* void * *arg*)

Tell the Jack server to call *thread_callback* in the RT thread. Typical use are in conjunction with *jack_cycle_wait* and @ jack_cycle_signal functions. The code in the

supplied function must be suitable for real-time execution. That means that it cannot call functions that might block for a long time. This includes malloc, free, printf, pthread_mutex_lock, sleep, wait, poll, select, pthread_join, pthread_cond_wait, etc, etc.

Returns

0 on success, otherwise a non-zero error code.

8.2.1.4 jack_nframes_t jack_thread_wait (jack_client_t *, int *status*)

THIS FUNCTION IS DEPRECATED AND SHOULD NOT BE USED IN NEW JACK CLIENTS

It should be replace by use of @ jack_cycle_wait and @ jack_cycle_signal functions.

8.3 Setting Client Callbacks

Functions

- int jack_set_thread_init_callback (jack_client_t *client, JackThreadInitCallback thread_init_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- void jack_on_shutdown (jack_client_t *client, JackShutdownCallback function, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- void jack_on_info_shutdown (jack_client_t *client, JackInfoShutdownCallback function, void *arg) JACK_WEAK_EXPORT
- int jack_set_process_callback (jack_client_t *client, JackProcessCallback process_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_freewheel_callback (jack_client_t *client, JackFreewheelCallback freewheel_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_buffer_size_callback (jack_client_t *client, JackBufferSizeCallback bufsize_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sample_rate_callback (jack_client_t *client, JackSampleRateCallback srate_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_client_registration_callback (jack_client_t *, JackClientRegistrationCallback registration_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_port_registration_callback (jack_client_t *, JackPortRegistrationCallback registration_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_port_connect_callback (jack_client_t *, JackPortConnectCallback connect_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT

- int jack_set_graph_order_callback (jack_client_t *, JackGraphOrderCallback graph_callback, void *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_xrun_callback (jack_client_t *, JackXRunCallback xrun_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT

8.3.1 Function Documentation

8.3.1.1 void jack_on_info_shutdown (jack_client_t * *client*, JackInfoShutdownCallback *function*, void * *arg*)

Parameters

client pointer to JACK client structure.

function The jack_shutdown function pointer.

arg The arguments for the jack_shutdown function.

Register a function (and argument) to be called if and when the JACK server shuts down the client thread. The function must be written as if it were an asynchonrous POSIX signal handler --- use only async-safe functions, and remember that it is executed from another thread. A typical function might set a flag or write to a pipe so that the rest of the application knows that the JACK client thread has shut down.

NOTE: clients do not need to call this. It exists only to help more complex clients understand what is going on. It should be called before jack_client_activate().

NOTE: if a client calls this AND jack_on_shutdown(), then in the event of a client thread shutdown, the callback passed to this function will be called, and the one passed to jack_on_shutdown() will not.

8.3.1.2 void jack_on_shutdown (jack_client_t * *client*, JackShutdownCallback *function*, void * *arg*)

Parameters

client pointer to JACK client structure.

function The jack_shutdown function pointer.

arg The arguments for the jack_shutdown function.

Register a function (and argument) to be called if and when the JACK server shuts down the client thread. The function must be written as if it were an asynchonrous POSIX signal handler --- use only async-safe functions, and remember that it is executed from another thread. A typical function might set a flag or write to a pipe so that the rest of the application knows that the JACK client thread has shut down.

NOTE: clients do not need to call this. It exists only to help more complex clients understand what is going on. It should be called before jack_client_activate().

NOTE: if a client calls this AND jack_on_info_shutdown(), then the event of a client thread shutdown, the callback passed to this function will not be called, and the one passed to jack_on_info_shutdown() will.

Referenced by main().

8.3.1.3 int jack_set_buffer_size_callback (jack_client_t * *client*, JackBufferSizeCallback *bufsize_callback*, void * *arg*)

Tell JACK to call *bufsize_callback* whenever the size of the the buffer that will be passed to the *process_callback* is about to change. Clients that depend on knowing the buffer size must supply a *bufsize_callback* before activating themselves.

Parameters

client pointer to JACK client structure.

bufsize_callback function to call when the buffer size changes.

arg argument for *bufsize_callback*.

Returns

0 on success, otherwise a non-zero error code

8.3.1.4 int jack_set_client_registration_callback (jack_client_t * , JackClientRegistrationCallback *registration_callback*, void * *arg*)

Tell the JACK server to call *registration_callback* whenever a port is registered or unregistered, passing *arg* as a parameter.

Returns

0 on success, otherwise a non-zero error code

8.3.1.5 int jack_set_freewheel_callback (jack_client_t * *client*, JackFreewheelCallback *freewheel_callback*, void * *arg*)

Tell the Jack server to call *freewheel_callback* whenever we enter or leave "freewheel" mode, passing *arg* as the second argument. The first argument to the callback will be non-zero if JACK is entering freewheel mode, and zero otherwise.

Returns

0 on success, otherwise a non-zero error code.

8.3.1.6 int jack_set_graph_order_callback (jack_client_t * , JackGraphOrderCallback *graph_callback,* void *)

Tell the JACK server to call *graph_callback* whenever the processing graph is reordered, passing *arg* as a parameter.

Returns

0 on success, otherwise a non-zero error code

8.3.1.7 int jack_set_port_connect_callback (jack_client_t * , JackPortConnectCallback *connect_callback,* void * *arg*)

Tell the JACK server to call *connect_callback* whenever a port is connected or disconnected, passing *arg* as a parameter.

Returns

0 on success, otherwise a non-zero error code

8.3.1.8 int jack_set_port_registration_callback (jack_client_t * , JackPortRegistrationCallback *registration_callback,* void * *arg*)

Tell the JACK server to call *registration_callback* whenever a port is registered or unregistered, passing *arg* as a parameter.

Returns

0 on success, otherwise a non-zero error code

8.3.1.9 int jack_set_process_callback (jack_client_t * *client,* JackProcessCallback *process_callback,* void * *arg*)

Tell the Jack server to call *process_callback* whenever there is work be done, passing *arg* as the second argument.

The code in the supplied function must be suitable for real-time execution. That means that it cannot call functions that might block for a long time. This includes malloc, free, printf, pthread_mutex_lock, sleep, wait, poll, select, pthread_join, pthread_cond_wait, etc, etc.

Returns

0 on success, otherwise a non-zero error code, causing JACK to remove that client from the process() graph.

Referenced by jack_initialize(), and main().

8.3.1.10 int jack_set_sample_rate_callback (jack_client_t * *client*, JackSampleRateCallback *srate_callback*, void * *arg*)

Tell the Jack server to call *srate_callback* whenever the system sample rate changes.

Returns

0 on success, otherwise a non-zero error code

8.3.1.11 int jack_set_thread_init_callback (jack_client_t * *client*, JackThreadInitCallback *thread_init_callback*, void * *arg*)

Tell JACK to call *thread_init_callback* once just after the creation of the thread in which all other callbacks will be handled.

The code in the supplied function does not need to be suitable for real-time execution.

Returns

0 on success, otherwise a non-zero error code, causing JACK to remove that client from the process() graph.

8.3.1.12 int jack_set_xrun_callback (jack_client_t *, JackXRunCallback *xrun_callback*, void * *arg*)

Tell the JACK server to call *xrun_callback* whenever there is a xrun, passing *arg* as a parameter.

Returns

0 on success, otherwise a non-zero error code

8.4 Controlling & querying JACK server operation

Functions

- int jack_set_freewheel (jack_client_t *client, int onoff) JACK_OPTIONAL_-WEAK_EXPORT
- int jack_set_buffer_size (jack_client_t *client, jack_nframes_t nframes) JACK_-OPTIONAL_WEAK_EXPORT

- jack_nframes_t jack_get_sample_rate (jack_client_t *) JACK_OPTIONAL_-WEAK_EXPORT
- jack_nframes_t jack_get_buffer_size (jack_client_t *) JACK_OPTIONAL_-WEAK_EXPORT
- int jack_engine_takeover_timebase (jack_client_t *) JACK_OPTIONAL_-WEAK_DEPRECATED_EXPORT
- float jack_cpu_load (jack_client_t *client) JACK_OPTIONAL_WEAK_-EXPORT

8.4.1 Function Documentation

8.4.1.1 float jack_cpu_load (jack_client_t * *client*)

Returns

the current CPU load estimated by JACK. This is a running average of the time it takes to execute a full process cycle for all clients as a percentage of the real time available per cycle determined by the buffer size and sample rate.

8.4.1.2 int jack_engine_takeover_timebase (jack_client_t *)

Old-style interface to become the timebase for the entire JACK subsystem.

Deprecated

This function still exists for compatibility with the earlier transport interface, but it does nothing. Instead, see transport.h and use jack_set_timebase_callback().

Returns

ENOSYS, function not implemented.

8.4.1.3 jack_nframes_t jack_get_buffer_size (jack_client_t *)

Returns

the current maximum size that will ever be passed to the *process_callback*. It should only be used *before* the client has been activated. This size may change, clients that depend on it must register a *bufsize_callback* so they will be notified if it does.

See also

jack_set_buffer_size_callback()

8.4.1.4 jack_nframes_t jack_get_sample_rate (jack_client_t *)

Returns

the sample rate of the jack system, as set by the user when jackd was started.

Referenced by main().

8.4.1.5 int jack_set_buffer_size (jack_client_t * *client*, jack_nframes_t *nframes*)

Change the buffer size passed to the *process_callback*.

This operation stops the JACK engine process cycle, then calls all registered *bufsize_callback* functions before restarting the process cycle. This will cause a gap in the audio flow, so it should only be done at appropriate stopping points.

See also

jack_set_buffer_size_callback()

Parameters

client pointer to JACK client structure.

nframes new buffer size. Must be a power of two.

Returns

0 on success, otherwise a non-zero error code

8.4.1.6 int jack_set_freewheel (jack_client_t * *client*, int *onoff*)

Start/Stop JACK's "freewheel" mode.

When in "freewheel" mode, JACK no longer waits for any external event to begin the start of the next process cycle.

As a result, freewheel mode causes "faster than realtime" execution of a JACK graph. If possessed, real-time scheduling is dropped when entering freewheel mode, and if appropriate it is reacquired when stopping.

IMPORTANT: on systems using capabilities to provide real-time scheduling (i.e. Linux kernel 2.4), if onoff is zero, this function must be called from the thread that originally called jack_activate(). This restriction does not apply to other systems (e.g. Linux kernel 2.6 or OS X).

Parameters

client pointer to JACK client structure

onoff if non-zero, freewheel mode starts. Otherwise freewheel mode ends.

Returns

0 on success, otherwise a non-zero error code.

8.5 Creating & manipulating ports

Functions

- jack_port_t * jack_port_register (jack_client_t *client, const char *port_name, const char *port_type, unsigned long flags, unsigned long buffer_size) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_unregister (jack_client_t *, jack_port_t *) JACK_OPTIONAL_WEAK_EXPORT
- void * jack_port_get_buffer (jack_port_t *, jack_nframes_t) JACK_OPTIONAL_WEAK_EXPORT
- const char * jack_port_name (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- const char * jack_port_short_name (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_flags (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- const char * jack_port_type (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_is_mine (const jack_client_t *, const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_connected (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_connected_to (const jack_port_t *port, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- const char ** jack_port_get_connections (const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- const char ** jack_port_get_all_connections (const jack_client_t *client, const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_tie (jack_port_t *src, jack_port_t *dst) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_port_untie (jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- jack_nframes_t jack_port_get_latency (jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_port_get_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT

- void jack_port_set_latency (jack_port_t *, jack_nframes_t) JACK_OPTIONAL_WEAK_EXPORT
- int jack_recompute_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_recompute_total_latencies (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_set_name (jack_port_t *port, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_set_alias (jack_port_t *port, const char *alias) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_unset_alias (jack_port_t *port, const char *alias) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_get_aliases (const jack_port_t *port, char *const aliases[2]) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_request_monitor (jack_port_t *port, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_request_monitor_by_name (jack_client_t *client, const char *port_name, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_ensure_monitor (jack_port_t *port, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_monitoring_input (jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_connect (jack_client_t *, const char *source_port, const char *destination_port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_disconnect (jack_client_t *, const char *source_port, const char *destination_port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_disconnect (jack_client_t *, jack_port_t *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_name_size (void) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_type_size (void) JACK_OPTIONAL_WEAK_EXPORT

8.5.1 Function Documentation

8.5.1.1 int jack_connect (jack_client_t * , const char * *source_port,* const char * *destination_port*)

Establish a connection between two ports.

When a connection exists, data written to the source port will be available to be read at the destination port.

Precondition

The port types must be identical.
The JackPortFlags of the *source_port* must include JackPortIsOutput.
The JackPortFlags of the *destination_port* must include JackPortIsInput.

Returns

0 on success, EEXIST if the connection is already made, otherwise a non-zero error code

Referenced by jack_initialize(), and main().

8.5.1.2 int jack_disconnect (jack_client_t * , const char * *source_port*, const char * *destination_port*)

Remove a connection between two ports.

Precondition

The port types must be identical.
The JackPortFlags of the *source_port* must include JackPortIsOutput.
The JackPortFlags of the *destination_port* must include JackPortIsInput.

Returns

0 on success, otherwise a non-zero error code

8.5.1.3 int jack_port_connected (const jack_port_t * *port*)

Returns

number of connections to or from *port*.

Precondition

The calling client must own *port*.

8.5.1.4 int jack_port_connected_to (const jack_port_t * *port*, const char * *port_name*)

Returns

TRUE if the locally-owned *port* is **directly** connected to the *port_name*.

See also

jack_port_name_size()

8.5.1.5 int jack_port_disconnect (jack_client_t * , jack_port_t *)

Perform the same function as jack_disconnect() using port handles rather than names. This avoids the name lookup inherent in the name-based version.

Clients connecting their own ports are likely to use this function, while generic connection clients (e.g. patchbays) would use jack_disconnect().

8.5.1.6 int jack_port_ensure_monitor (jack_port_t * *port*, int *onoff*)

If JackPortCanMonitor is set for a port, this function turns on input monitoring if it was off, and turns it off if only one request has been made to turn it on. Otherwise it does nothing.

Returns

0 on success, otherwise a non-zero error code

8.5.1.7 int jack_port_flags (const jack_port_t * *port*)

Returns

the JackPortFlags of the jack_port_t.

8.5.1.8 int jack_port_get_aliases (const jack_port_t * *port*, char *const *aliases[2]*)

8.5.1.9 const char** jack_port_get_all_connections (const jack_client_t * *client*, const jack_port_t * *port*)

Returns

a null-terminated array of full port names to which the *port* is connected. If none, returns NULL.

The caller is responsible for calling jack_free(3) on any non-NULL returned value.

This differs from jack_port_get_connections() in two important respects:

1) You may not call this function from code that is executed in response to a JACK event. For example, you cannot use it in a GraphReordered handler.

2) You need not be the owner of the port to get information about its connections.

See also

jack_port_name_size()

8.5.1.10 void* jack_port_get_buffer (jack_port_t * , jack_nframes_t)

This returns a pointer to the memory area associated with the specified port. For an output port, it will be a memory area that can be written to; for an input port, it will be an area containing the data from the port's connection(s), or zero-filled. if there are multiple inbound connections, the data will be mixed appropriately.

Do not cache the returned address across process() callbacks. Port buffers have to be retrieved in each callback for proper functionning.

Referenced by inprocess(), and process().

8.5.1.11 const char** jack_port_get_connections (const jack_port_t * *port*)

Returns

a null-terminated array of full port names to which the *port* is connected. If none, returns NULL.

The caller is responsible for calling jack_free(3) on any non-NULL returned value.

Parameters

port locally owned jack_port_t pointer.

See also

jack_port_name_size(), jack_port_get_all_connections()

8.5.1.12 jack_nframes_t jack_port_get_latency (jack_port_t * *port*)

Returns

the time (in frames) between data being available or delivered at/to a port, and the time at which it arrived at or is delivered to the "other side" of the port. E.g. for a physical audio output port, this is the time between writing to the port and when the signal will leave the connector. For a physical audio input port, this is the time between the sound arriving at the connector and the corresponding frames being readable from the port.

8.5.1.13 jack_nframes_t jack_port_get_total_latency (jack_client_t * , jack_port_t * *port*)

The maximum of the sum of the latencies in every connection path that can be drawn between the port and other ports with the JackPortIsTerminal flag set.

8.5.1.14 int jack_port_is_mine (const jack_client_t * , const jack_port_t * *port*)

Returns

TRUE if the jack_port_t belongs to the jack_client_t.

8.5.1.15 int jack_port_monitoring_input (jack_port_t * *port*)

Returns

TRUE if input monitoring has been requested for *port*.

8.5.1.16 const char* jack_port_name (const jack_port_t * *port*)

Returns

the full name of the jack_port_t (including the *"client_name:"* prefix).

See also

jack_port_name_size().

Referenced by jack_initialize(), and main().

8.5.1.17 int jack_port_name_size (void)

Returns

the maximum number of characters in a full JACK port name including the final NULL character. This value is a constant.

A port's full name contains the owning client name concatenated with a colon (:) followed by its short name and a NULL character.

8.5.1.18 jack_port_t* jack_port_register (jack_client_t * *client,* const char * *port_name,* const char * *port_type,* unsigned long *flags,* unsigned long *buffer_size*)

Create a new port for the client. This is an object used for moving data of any type in or out of the client. Ports may be connected in various ways.

Each port has a short name. The port's full name contains the name of the client concatenated with a colon (:) followed by its short name. The jack_port_name_size()

is the maximum length of this full name. Exceeding that will cause the port registration to fail and return NULL.

All ports have a type, which may be any non-NULL and non-zero length string, passed as an argument. Some port types are built into the JACK API, like JACK_DEFAULT_-AUDIO_TYPE or JACK_DEFAULT_MIDI_TYPE

Parameters

client pointer to JACK client structure.

port_name non-empty short name for the new port (not including the leading *"client_name:"*).

port_type port type name. If longer than jack_port_type_size(), only that many characters are significant.

flags JackPortFlags bit mask.

buffer_size must be non-zero if this is not a built-in *port_type*. Otherwise, it is ignored.

Returns

jack_port_t pointer on success, otherwise NULL.

Referenced by jack_initialize(), and main().

8.5.1.19 int jack_port_request_monitor (jack_port_t * *port,* int *onoff*)

If JackPortCanMonitor is set for this *port*, turn input monitoring on or off. Otherwise, do nothing.

8.5.1.20 int jack_port_request_monitor_by_name (jack_client_t * *client,* const char * *port_name,* int *onoff*)

If JackPortCanMonitor is set for this *port_name*, turn input monitoring on or off. Otherwise, do nothing.

Returns

0 on success, otherwise a non-zero error code.

See also

jack_port_name_size()

8.5.1.21 int jack_port_set_alias (jack_port_t * *port*, const char * *alias*)

Set *alias* as an alias for *port*. May be called at any time. If the alias is longer than jack_port_name_size(), it will be truncated.

After a successful call, and until JACK exits or jack_port_unset_alias() is called, may be used as a alternate name for the port.

Ports can have up to two aliases - if both are already set, this function will return an error.

Returns

0 on success, otherwise a non-zero error code.

8.5.1.22 void jack_port_set_latency (jack_port_t *, jack_nframes_t)

The port latency is zero by default. Clients that control physical hardware with non-zero latency should call this to set the latency to its correct value. Note that the value should include any systemic latency present "outside" the physical hardware controlled by the client. For example, for a client controlling a digital audio interface connected to an external digital converter, the latency setting should include both buffering by the audio interface *and* the converter.

8.5.1.23 int jack_port_set_name (jack_port_t * *port*, const char * *port_name*)

Modify a port's short name. May be called at any time. If the resulting full name (including the *"client_name:"* prefix) is longer than jack_port_name_size(), it will be truncated.

Returns

0 on success, otherwise a non-zero error code.

8.5.1.24 const char* jack_port_short_name (const jack_port_t * *port*)

Returns

the short name of the jack_port_t (not including the *"client_name:"* prefix).

See also

jack_port_name_size().

8.5.1.25 int jack_port_tie (jack_port_t * *src*, jack_port_t * *dst*)

Deprecated

This function will be removed from a future version of JACK. Do not use it. There is no replacement. It has turned out to serve essentially no purpose in real-life JACK clients.

8.5.1.26 const char* jack_port_type (const jack_port_t * *port*)

Returns

the *port* type, at most jack_port_type_size() characters including a final NULL.

8.5.1.27 int jack_port_type_size (void)

Returns

the maximum number of characters in a JACK port type name including the final NULL character. This value is a constant.

8.5.1.28 int jack_port_unregister (jack_client_t *, jack_port_t *)

Remove the port from the client, disconnecting any existing connections.

Returns

0 on success, otherwise a non-zero error code

8.5.1.29 int jack_port_unset_alias (jack_port_t * *port*, const char * *alias*)

Remove *alias* as an alias for *port*. May be called at any time.

After a successful call, *alias* can no longer be used as a alternate name for the port.

Returns

0 on success, otherwise a non-zero error code.

8.5.1.30 int jack_port_untie (jack_port_t * *port*)

Deprecated

This function will be removed from a future version of JACK. Do not use it. There is no replacement. It has turned out to serve essentially no purpose in real-life JACK clients.

8.5.1.31 int jack_recompute_total_latencies (jack_client_t *)

Request a complete recomputation of all port latencies. This can be called by a client that has just changed the internal latency of its port using jack_port_set_latency and wants to ensure that all signal pathways in the graph are updated with respect to the values that will be returned by jack_port_get_total_latency. It allows a client to change multiple port latencies without triggering a recompute for each change.

Returns

zero for successful execution of the request. non-zero otherwise.

8.5.1.32 int jack_recompute_total_latency (jack_client_t * , jack_port_t * *port*)

Request a complete recomputation of a port's total latency. This can be called by a client that has just changed the internal latency of its port using jack_port_set_latency and wants to ensure that all signal pathways in the graph are updated with respect to the values that will be returned by jack_port_get_total_latency.

Returns

zero for successful execution of the request. non-zero otherwise.

8.6 Looking up ports

Functions

- const char ** jack_get_ports (jack_client_t *, const char *port_name_pattern, const char *type_name_pattern, unsigned long flags) JACK_OPTIONAL_WEAK_EXPORT
- jack_port_t * jack_port_by_name (jack_client_t *, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- jack_port_t * jack_port_by_id (jack_client_t *client, jack_port_id_t port_id) JACK_OPTIONAL_WEAK_EXPORT

8.6.1 Function Documentation

8.6.1.1 const char** jack_get_ports (jack_client_t * , const char * *port_name_pattern*, const char * *type_name_pattern*, unsigned long *flags*)

Parameters

port_name_pattern A regular expression used to select ports by name. If NULL or of zero length, no selection based on name will be carried out.

type_name_pattern A regular expression used to select ports by type. If NULL or of zero length, no selection based on type will be carried out.

flags A value used to select ports by their flags. If zero, no selection based on flags will be carried out.

Returns

a NULL-terminated array of ports that match the specified arguments. The caller is responsible for calling jack_free(3) any non-NULL returned value.

See also

jack_port_name_size(), jack_port_type_size()

Referenced by main().

8.6.1.2 jack_port_t* jack_port_by_id (jack_client_t * *client*, jack_port_id_t *port_id*)

Returns

address of the jack_port_t of a *port_id*.

8.6.1.3 jack_port_t* jack_port_by_name (jack_client_t * , const char * *port_name*)

Returns

address of the jack_port_t named *port_name*.

See also

jack_port_name_size()

8.7 Handling time

Functions

- jack_nframes_t jack_frames_since_cycle_start (const jack_client_t *) JACK_-OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_frame_time (const jack_client_t *) JACK_OPTIONAL_-WEAK_EXPORT
- jack_nframes_t jack_last_frame_time (const jack_client_t *client) JACK_-OPTIONAL_WEAK_EXPORT
- jack_time_t jack_frames_to_time (const jack_client_t *client, jack_nframes_t) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_time_to_frames (const jack_client_t *client, jack_time_t) JACK_OPTIONAL_WEAK_EXPORT
- jack_time_t jack_get_time () JACK_OPTIONAL_WEAK_EXPORT

8.7.1 Detailed Description

JACK time is in units of 'frames', according to the current sample rate. The absolute value of frame times is meaningless, frame times have meaning only relative to each other.

8.7.2 Function Documentation

8.7.2.1 jack_nframes_t jack_frame_time (const jack_client_t *)

Returns

the estimated current time in frames. This function is intended for use in other threads (not the process callback). The return value can be compared with the value of jack_last_frame_time to relate time in other threads to JACK time.

8.7.2.2 jack_nframes_t jack_frames_since_cycle_start (const jack_client_t *)

Returns

the estimated time in frames that has passed since the JACK server began the current process cycle.

8.7.2.3 jack_time_t jack_frames_to_time (const jack_client_t * *client,* jack_nframes_t)

Returns

the estimated time in microseconds of the specified frame time

8.7.2.4 jack_time_t jack_get_time ( )

Returns

return JACK's current system time in microseconds, using the JACK clock source.

The value returned is guaranteed to be monotonic, but not linear.

8.7.2.5 jack_nframes_t jack_last_frame_time (const jack_client_t * *client*)

Returns

the precise time at the start of the current process cycle. This function may only be used from the process callback, and can be used to interpret timestamps generated by jack_frame_time() in other threads with respect to the current process cycle.

This is the only jack time function that returns exact time: when used during the process callback it always returns the same value (until the next process callback, where it will return that value + nframes, etc). The return value is guaranteed to be monotonic and linear in this fashion unless an xrun occurs. If an xrun occurs, clients must check this value again, as time may have advanced in a non-linear way (e.g. cycles may have been skipped).

8.7.2.6 jack_nframes_t jack_time_to_frames (const jack_client_t * *client,* jack_time_t)

Returns

the estimated time in frames for the specified system time.

8.8 Controlling error/information output

Functions

- void jack_set_error_function (void(*func)(const char *)) JACK_OPTIONAL_-WEAK_EXPORT

- void jack_set_info_function (void(*func)(const char *)) JACK_OPTIONAL_-WEAK_EXPORT

Variables

- void(* jack_error_callback)(const char *msg) JACK_OPTIONAL_WEAK_-EXPORT
- void(* jack_info_callback)(const char *msg) JACK_OPTIONAL_WEAK_-EXPORT

8.8.1 Function Documentation

8.8.1.1 void jack_set_error_function (void(*)(const char *) *func*)

Set the jack_error_callback for error message display.

The JACK library provides two built-in callbacks for this purpose: default_jack_error_-callback() and silent_jack_error_callback().

8.8.1.2 void jack_set_info_function (void(*)(const char *) *func*)

Set the jack_info_callback for info message display.

8.8.2 Variable Documentation

8.8.2.1 void(* jack_error_callback)(const char *msg) JACK_OPTIONAL_WEAK_EXPORT

Display JACK error message.

Set via jack_set_error_function(), otherwise a JACK-provided default will print *msg* (plus a newline) to stderr.

Parameters

msg error message text (no newline at end).

8.8.2.2 void(* jack_info_callback)(const char *msg) JACK_OPTIONAL_WEAK_EXPORT

Display JACK info message.

Set via jack_set_info_function(), otherwise a JACK-provided default will print *msg* (plus a newline) to stdout.

Parameters

msg info message text (no newline at end).

8.9 Creating and managing client threads

Typedefs

- typedef int(∗ jack_thread_creator_t)(pthread_t ∗, const pthread_attr_t ∗, void ∗(∗function)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT

Functions

- int jack_client_real_time_priority (jack_client_t ∗) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_max_real_time_priority (jack_client_t ∗) JACK_OPTIONAL_WEAK_EXPORT
- int jack_acquire_real_time_scheduling (pthread_t thread, int priority) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_create_thread (jack_client_t ∗client, pthread_t ∗thread, int priority, int realtime, void ∗(∗start_routine)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_drop_real_time_scheduling (pthread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- void jack_set_thread_creator (jack_thread_creator_t creator) JACK_OPTIONAL_WEAK_EXPORT

8.9.1 Typedef Documentation

8.9.1.1 typedef int(∗ jack_thread_creator_t)(pthread_t ∗, const pthread_attr_t ∗, void ∗(∗function)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT

8.9.2 Function Documentation

8.9.2.1 int jack_acquire_real_time_scheduling (pthread_t *thread*, int *priority*)

Attempt to enable realtime scheduling for a thread. On some systems that may require special privileges.

Parameters

thread POSIX thread ID.

priority requested thread priority.

Returns

0, if successful; EPERM, if the calling process lacks required realtime privileges; otherwise some other error number.

8.9.2.2 int jack_client_create_thread (jack_client_t * *client,* pthread_t * *thread,* int *priority,* int *realtime,* void *(*)(void *) *start_routine,* void * *arg*)

Create a thread for JACK or one of its clients. The thread is created executing *start_routine* with *arg* as its sole argument.

Parameters

client the JACK client for whom the thread is being created. May be NULL if the client is being created within the JACK server.

thread place to return POSIX thread ID.

priority thread priority, if realtime.

realtime true for the thread to use realtime scheduling. On some systems that may require special privileges.

start_routine function the thread calls when it starts.

arg parameter passed to the *start_routine*.

Returns

0, if successful; otherwise some error number.

8.9.2.3 int jack_client_max_real_time_priority (jack_client_t *)

Returns

if JACK is running with realtime scheduling, this returns the maximum priority that a JACK client thread should use if the thread is subject to realtime scheduling. Otherwise returns -1.

8.9.2.4 int jack_client_real_time_priority (jack_client_t *)

Returns

if JACK is running with realtime scheduling, this returns the priority that any JACK-created client threads will run at. Otherwise returns -1.

8.9.2.5 int jack_drop_real_time_scheduling (pthread_t *thread*)

Drop realtime scheduling for a thread.

Parameters

thread POSIX thread ID.

Returns

0, if successful; otherwise an error number.

8.9.2.6 void jack_set_thread_creator (jack_thread_creator_t *creator*)

This function can be used in very very specialized cases where it is necessary that client threads created by JACK are created by something other than pthread_create(). After it is used, any threads that JACK needs for the client will will be created by calling the function passed to this function.

No normal application/client should consider calling this. The specific case for which it was created involves running win32/x86 plugins under Wine on Linux, where it is necessary that all threads that might call win32 functions are known to Wine.

Parameters

creator a function that creates a new thread

8.10 Transport and Timebase control

Typedefs

- typedef int(∗ JackSyncCallback)(jack_transport_state_t state, jack_position_t ∗pos, void ∗arg)
- typedef void(∗ JackTimebaseCallback)(jack_transport_state_t state, jack_nframes_t nframes, jack_position_t ∗pos, int new_pos, void ∗arg)

Functions

- int jack_release_timebase (jack_client_t ∗client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sync_callback (jack_client_t ∗client, JackSyncCallback sync_callback, void ∗arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sync_timeout (jack_client_t ∗client, jack_time_t timeout) JACK_OPTIONAL_WEAK_EXPORT

- int jack_set_timebase_callback (jack_client_t *client, int conditional, JackTimebaseCallback timebase_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_transport_locate (jack_client_t *client, jack_nframes_t frame) JACK_OPTIONAL_WEAK_EXPORT
- jack_transport_state_t jack_transport_query (const jack_client_t *client, jack_position_t *pos) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_get_current_transport_frame (const jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_transport_reposition (jack_client_t *client, jack_position_t *pos) JACK_OPTIONAL_WEAK_EXPORT
- void jack_transport_start (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- void jack_transport_stop (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT

8.10.1 Typedef Documentation

8.10.1.1 typedef int(* JackSyncCallback)(jack_transport_state_t state, jack_position_t *pos, void *arg)

Prototype for the *sync_callback* defined by slow-sync clients. When the client is active, this callback is invoked just before process() in the same thread. This occurs once after registration, then subsequently whenever some client requests a new position, or the transport enters the JackTransportStarting state. This realtime function must not wait.

The transport *state* will be:

- JackTransportStopped when a new position is requested;
- JackTransportStarting when the transport is waiting to start;
- JackTransportRolling when the timeout has expired, and the position is now a moving target.

Parameters

state current transport state.

pos new transport position.

arg the argument supplied by jack_set_sync_callback().

Returns

TRUE (non-zero) when ready to roll.

8.10.1.2 typedef void(∗ JackTimebaseCallback)(jack_transport_state_t state, jack_nframes_t nframes, jack_position_t ∗pos, int new_pos, void ∗arg)

Prototype for the *timebase_callback* used to provide extended position information. Its output affects all of the following process cycle. This realtime function must not wait.

This function is called immediately after process() in the same thread whenever the transport is rolling, or when any client has requested a new position in the previous cycle. The first cycle after jack_set_timebase_callback() is also treated as a new position, or the first cycle after jack_activate() if the client had been inactive.

The timebase master may not use its *pos* argument to set *pos->frame*. To change position, use jack_transport_reposition() or jack_transport_locate(). These functions are realtime-safe, the *timebase_callback* can call them directly.

Parameters

state current transport state.

nframes number of frames in current period.

pos address of the position structure for the next cycle; *pos->frame* will be its frame number. If *new_pos* is FALSE, this structure contains extended position information from the current cycle. If TRUE, it contains whatever was set by the requester. The *timebase_callback's* task is to update the extended information here.

new_pos TRUE (non-zero) for a newly requested *pos*, or for the first cycle after the *timebase_callback* is defined.

arg the argument supplied by jack_set_timebase_callback().

8.10.2 Function Documentation

8.10.2.1 jack_nframes_t jack_get_current_transport_frame (const jack_client_t ∗ *client*)

Return an estimate of the current transport frame, including any time elapsed since the last transport positional update.

Parameters

client the JACK client structure

8.10.2.2 int jack_release_timebase (jack_client_t ∗ *client*)

Called by the timebase master to release itself from that responsibility.

If the timebase master releases the timebase or leaves the JACK graph for any reason, the JACK engine takes over at the start of the next process cycle. The transport state does not change. If rolling, it continues to play, with frame numbers as the only available position information.

See also

jack_set_timebase_callback

Parameters

client the JACK client structure.

Returns

0 on success, otherwise a non-zero error code.

8.10.2.3 int jack_set_sync_callback (jack_client_t * *client*, JackSyncCallback *sync_callback*, void * *arg*)

Register (or unregister) as a slow-sync client, one that cannot respond immediately to transport position changes.

The *sync_callback* will be invoked at the first available opportunity after its registration is complete. If the client is currently active this will be the following process cycle, otherwise it will be the first cycle after calling jack_activate(). After that, it runs according to the JackSyncCallback rules. Clients that don't set a *sync_callback* are assumed to be ready immediately any time the transport wants to start.

Parameters

client the JACK client structure.

sync_callback is a realtime function that returns TRUE when the client is ready. Setting *sync_callback* to NULL declares that this client no longer requires slow-sync processing.

arg an argument for the *sync_callback* function.

Returns

0 on success, otherwise a non-zero error code.

8.10.2.4 int jack_set_sync_timeout (jack_client_t * *client*, jack_time_t *timeout*)

Set the timeout value for slow-sync clients.

This timeout prevents unresponsive slow-sync clients from completely halting the transport mechanism. The default is two seconds. When the timeout expires, the transport starts rolling, even if some slow-sync clients are still unready. The *sync_callbacks* of these clients continue being invoked, giving them a chance to catch up.

See also

jack_set_sync_callback

Parameters

client the JACK client structure.

timeout is delay (in microseconds) before the timeout expires.

Returns

0 on success, otherwise a non-zero error code.

8.10.2.5 int jack_set_timebase_callback (jack_client_t * *client,* int *conditional,* JackTimebaseCallback *timebase_callback,* void * *arg*)

Register as timebase master for the JACK subsystem.

The timebase master registers a callback that updates extended position information such as beats or timecode whenever necessary. Without this extended information, there is no need for this function.

There is never more than one master at a time. When a new client takes over, the former *timebase_callback* is no longer called. Taking over the timebase may be done conditionally, so it fails if there was a master already.

Parameters

client the JACK client structure.

conditional non-zero for a conditional request.

timebase_callback is a realtime function that returns position information.

arg an argument for the *timebase_callback* function.

Returns

- 0 on success;
- EBUSY if a conditional request fails because there was already a timebase master;
- other non-zero error code.

8.10.2.6 int jack_transport_locate (jack_client_t * *client,* jack_nframes_t *frame*)

Reposition the transport to a new frame number.

May be called at any time by any client. The new position takes effect in two process cycles. If there are slow-sync clients and the transport is already rolling, it will enter the JackTransportStarting state and begin invoking their *sync_callbacks* until ready. This function is realtime-safe.

See also

jack_transport_reposition, jack_set_sync_callback

Parameters

client the JACK client structure.

frame frame number of new transport position.

Returns

0 if valid request, non-zero otherwise.

8.10.2.7 jack_transport_state_t jack_transport_query (const jack_client_t * *client,* jack_position_t * *pos*)

Query the current transport state and position.

This function is realtime-safe, and can be called from any thread. If called from the process thread, *pos* corresponds to the first frame of the current cycle and the state returned is valid for the entire cycle.

Parameters

client the JACK client structure.

pos pointer to structure for returning current transport position; *pos->valid* will show which fields contain valid data. If *pos* is NULL, do not return position information.

Returns

Current transport state.

8.10.2.8 int jack_transport_reposition (jack_client_t * *client,* jack_position_t * *pos*)

Request a new transport position.

May be called at any time by any client. The new position takes effect in two process cycles. If there are slow-sync clients and the transport is already rolling, it will enter the JackTransportStarting state and begin invoking their *sync_callbacks* until ready. This function is realtime-safe.

See also

jack_transport_locate, jack_set_sync_callback

Parameters

client the JACK client structure.

pos requested new transport position.

Returns

0 if valid request, EINVAL if position structure rejected.

8.10.2.9 void jack_transport_start (jack_client_t * *client*)

Start the JACK transport rolling.

Any client can make this request at any time. It takes effect no sooner than the next process cycle, perhaps later if there are slow-sync clients. This function is realtime-safe.

See also

jack_set_sync_callback

Parameters

client the JACK client structure.

8.10.2.10 void jack_transport_stop (jack_client_t * *client*)

Stop the JACK transport.

Any client can make this request at any time. It takes effect on the next process cycle. This function is realtime-safe.

Parameters

client the JACK client structure.

8.11 Reading and writing MIDI data

Functions

- jack_nframes_t jack_midi_get_event_count (void *port_buffer) JACK_OPTIONAL_WEAK_EXPORT
- int jack_midi_event_get (jack_midi_event_t *event, void *port_buffer, jack_nframes_t event_index) JACK_OPTIONAL_WEAK_EXPORT
- void jack_midi_clear_buffer (void *port_buffer) JACK_OPTIONAL_WEAK_EXPORT
- size_t jack_midi_max_event_size (void *port_buffer) JACK_OPTIONAL_WEAK_EXPORT
- jack_midi_data_t * jack_midi_event_reserve (void *port_buffer, jack_nframes_t time, size_t data_size) JACK_OPTIONAL_WEAK_EXPORT
- int jack_midi_event_write (void *port_buffer, jack_nframes_t time, const jack_midi_data_t *data, size_t data_size) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_midi_get_lost_event_count (void *port_buffer) JACK_OPTIONAL_WEAK_EXPORT

8.11.1 Function Documentation

8.11.1.1 void jack_midi_clear_buffer (void * *port_buffer*)

Clear an event buffer.

This should be called at the beginning of each process cycle before calling jack_midi_event_reserve or jack_midi_event_write. This function may not be called on an input port's buffer.

Parameters

port_buffer Port buffer to clear (must be an output port buffer).

8.11.1.2 int jack_midi_event_get (jack_midi_event_t * *event,* void * *port_buffer,* jack_nframes_t *event_index*)

Get a MIDI event from an event port buffer.

Jack MIDI is normalised, the MIDI event returned by this function is guaranteed to be a complete MIDI event (the status byte will always be present, and no realtime events will interspered with the event).

Parameters

event Event structure to store retrieved event in.

port_buffer Port buffer from which to retrieve event.

event_index Index of event to retrieve.

Returns

0 on success, ENODATA if buffer is empty.

8.11.1.3 jack_midi_data_t* jack_midi_event_reserve (void * *port_buffer,* jack_nframes_t *time,* size_t *data_size*)

Allocate space for an event to be written to an event port buffer.

Clients are to write the actual event data to be written starting at the pointer returned by this function. Clients must not write more than *data_size* bytes into this buffer. Clients must write normalised MIDI data to the port - no running status and no (1-byte) realtime messages interspersed with other messages (realtime messages are fine when they occur on their own, like other messages).

Events must be written in order, sorted by their sample offsets. JACK will not sort the events for you, and will refuse to store out-of-order events.

Parameters

port_buffer Buffer to write event to.

time Sample offset of event.

data_size Length of event's raw data in bytes.

Returns

Pointer to the beginning of the reserved event's data buffer, or NULL on error (ie not enough space).

8.11.1.4 int jack_midi_event_write (void * *port_buffer,* jack_nframes_t *time,* const jack_midi_data_t * *data,* size_t *data_size*)

Write an event into an event port buffer.

This function is simply a wrapper for jack_midi_event_reserve which writes the event data into the space reserved in the buffer.

Clients must not write more than *data_size* bytes into this buffer. Clients must write normalised MIDI data to the port - no running status and no (1-byte) realtime messages interspersed with other messages (realtime messages are fine when they occur on their own, like other messages).

Events must be written in order, sorted by their sample offsets. JACK will not sort the events for you, and will refuse to store out-of-order events.

Parameters

port_buffer Buffer to write event to.

time Sample offset of event.

data Message data to be written.

data_size Length of *data* in bytes.

Returns

0 on success, ENOBUFS if there's not enough space in buffer for event.

8.11.1.5 jack_nframes_t jack_midi_get_event_count (void * *port_buffer*)

8.11.1.6 jack_nframes_t jack_midi_get_lost_event_count (void * *port_buffer*)

Get the number of events that could not be written to *port_buffer*.

This function returning a non-zero value implies *port_buffer* is full. Currently the only way this can happen is if events are lost on port mixdown.

Parameters

port_buffer Port to receive count for.

Returns

Number of events that could not be written to *port_buffer*.

8.11.1.7 size_t jack_midi_max_event_size (void * *port_buffer*)

Get the size of the largest event that can be stored by the port.

This function returns the current space available, taking into account events already stored in the port.

Parameters

port_buffer Port buffer to check size of.

Chapter 9

Data Structure Documentation

9.1 _jack_midi_event Struct Reference

`#include <midiport.h>`

Data Fields

- jack_nframes_t time
- size_t size
- jack_midi_data_t * buffer

9.1.1 Detailed Description

A Jack MIDI event.

9.1.2 Field Documentation

9.1.2.1 jack_midi_data_t* _jack_midi_event::buffer

Raw MIDI data

9.1.2.2 size_t _jack_midi_event::size

Number of bytes of data in *buffer*

9.1.2.3 jack_nframes_t _jack_midi_event::time

Sample index at which event is valid

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

- midiport.h

9.2 jack_position_t Struct Reference

`#include <transport.h>`

Data Fields

- jack_unique_t unique_1
- jack_time_t usecs
- jack_nframes_t frame_rate
- jack_nframes_t frame
- jack_position_bits_t valid
- int32_t bar
- int32_t beat
- int32_t tick
- double bar_start_tick
- float beats_per_bar
- float beat_type
- double ticks_per_beat
- double beats_per_minute
- double frame_time
- double next_time
- jack_nframes_t bbt_offset
- float audio_frames_per_video_frame
- jack_nframes_t video_offset
- int32_t padding [7]
- jack_unique_t unique_2

9.2.1 Detailed Description

Struct for transport position information.

9.2.2 Field Documentation

9.2.2.1 float jack_position_t::audio_frames_per_video_frame

number of audio frames per video frame. Should be assumed zero if JackAudioVideoRatio is not set. If JackAudioVideoRatio is set and the value is zero, no video data exists within the JACK graph

9.2.2.2 int32_t jack_position_t::bar

current bar

9.2.2.3 double jack_position_t::bar_start_tick

9.2.2.4 jack_nframes_t jack_position_t::bbt_offset

frame offset for the BBT fields (the given bar, beat, and tick values actually refer to a time frame_offset frames before the start of the cycle), should be assumed to be 0 if JackBBTFrameOffset is not set. If JackBBTFrameOffset is set and this value is zero, the BBT time refers to the first frame of this cycle. If the value is positive, the BBT time refers to a frame that many frames before the start of the cycle.

9.2.2.5 int32_t jack_position_t::beat

current beat-within-bar

9.2.2.6 float jack_position_t::beat_type

time signature "denominator"

9.2.2.7 float jack_position_t::beats_per_bar

time signature "numerator"

9.2.2.8 double jack_position_t::beats_per_minute

9.2.2.9 jack_nframes_t jack_position_t::frame

frame number, always present

9.2.2.10 jack_nframes_t jack_position_t::frame_rate

current frame rate (per second)

9.2.2.11 double jack_position_t::frame_time

current time in seconds

9.2.2.12 double jack_position_t::next_time

next sequential frame_time (unless repositioned)

9.2.2.13 int32_t jack_position_t::padding[7]

9.2.2.14 int32_t jack_position_t::tick

current tick-within-beat

9.2.2.15 double jack_position_t::ticks_per_beat

9.2.2.16 jack_unique_t jack_position_t::unique_1

unique ID

9.2.2.17 jack_unique_t jack_position_t::unique_2

unique ID

9.2.2.18 jack_time_t jack_position_t::usecs

monotonic, free-rolling

9.2.2.19 jack_position_bits_t jack_position_t::valid

which other fields are valid

9.2.2.20 jack_nframes_t jack_position_t::video_offset

audio frame at which the first video frame in this cycle occurs. Should be assumed to be 0 if JackVideoFrameOffset is not set. If JackVideoFrameOffset is set, but the value

is zero, there is no video frame within this cycle.

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

- transport.h

9.3 jack_ringbuffer_data_t Struct Reference

`#include <ringbuffer.h>`

Data Fields

- char * buf
- size_t len

9.3.1 Field Documentation

9.3.1.1 char* jack_ringbuffer_data_t::buf

9.3.1.2 size_t jack_ringbuffer_data_t::len

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

- ringbuffer.h

9.4 jack_ringbuffer_t Struct Reference

`#include <ringbuffer.h>`

Data Fields

- char * buf
- volatile size_t write_ptr
- volatile size_t read_ptr
- size_t size
- size_t size_mask
- int mlocked

9.4.1 Field Documentation

9.4.1.1 char* jack_ringbuffer_t::buf

9.4.1.2 int jack_ringbuffer_t::mlocked

9.4.1.3 volatile size_t jack_ringbuffer_t::read_ptr

9.4.1.4 size_t jack_ringbuffer_t::size

9.4.1.5 size_t jack_ringbuffer_t::size_mask

9.4.1.6 volatile size_t jack_ringbuffer_t::write_ptr

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

- ringbuffer.h

9.5 jack_transport_info_t Struct Reference

`#include <transport.h>`

Data Fields

- jack_nframes_t frame_rate
- jack_time_t usecs
- jack_transport_bits_t valid
- jack_transport_state_t transport_state
- jack_nframes_t frame
- jack_nframes_t loop_start
- jack_nframes_t loop_end
- long smpte_offset
- float smpte_frame_rate
- int bar
- int beat
- int tick
- double bar_start_tick
- float beats_per_bar
- float beat_type
- double ticks_per_beat
- double beats_per_minute

9.5.1 Detailed Description

Deprecated struct for transport position information.

Deprecated

This is for compatibility with the earlier transport interface. Use the jack_position_t struct, instead.

9.5.2 Field Documentation

9.5.2.1 int jack_transport_info_t::bar

9.5.2.2 double jack_transport_info_t::bar_start_tick

9.5.2.3 int jack_transport_info_t::beat

9.5.2.4 float jack_transport_info_t::beat_type

9.5.2.5 float jack_transport_info_t::beats_per_bar

9.5.2.6 double jack_transport_info_t::beats_per_minute

9.5.2.7 jack_nframes_t jack_transport_info_t::frame

9.5.2.8 jack_nframes_t jack_transport_info_t::frame_rate

current frame rate (per second)

9.5.2.9 jack_nframes_t jack_transport_info_t::loop_end

9.5.2.10 jack_nframes_t jack_transport_info_t::loop_start

9.5.2.11 float jack_transport_info_t::smpte_frame_rate

29.97, 30, 24 etc.

9.5.2.12 long jack_transport_info_t::smpte_offset

SMPTE offset (from frame 0)

9.5.2.13 int jack_transport_info_t::tick

9.5.2.14 double jack_transport_info_t::ticks_per_beat

9.5.2.15 jack_transport_state_t jack_transport_info_t::transport_state

9.5.2.16 jack_time_t jack_transport_info_t::usecs

monotonic, free-rolling

9.5.2.17 jack_transport_bits_t jack_transport_info_t::valid

which fields are legal to read

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

- transport.h

9.6 port_pair_t Struct Reference

Data Fields

- jack_port_t * input_port
- jack_port_t * output_port

9.6.1 Detailed Description

For the sake of example, an instance of this struct is allocated in jack_initialize(), passed to inprocess() as an argument, then freed in jack_finish().

9.6.2 Field Documentation

9.6.2.1 jack_port_t* port_pair_t::input_port

Referenced by inprocess(), and jack_initialize().

9.6.2.2 jack_port_t* port_pair_t::output_port

Referenced by inprocess(), and jack_initialize().

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

- inprocess.c

Chapter 10

File Documentation

10.1 inprocess.c File Reference

This demonstrates the basic concepts for writing a client that runs within the JACK server process.

```
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <jack/jack.h>
```

Data Structures

- struct port_pair_t

Functions

- int inprocess (jack_nframes_t nframes, void *arg)
- int jack_initialize (jack_client_t *client, const char *load_init)
- void jack_finish (void *arg)

10.1.1 Detailed Description

This demonstrates the basic concepts for writing a client that runs within the JACK server process. For the sake of example, a port_pair_t is allocated in jack_initialize(), passed to inprocess() as an argument, then freed in jack_finish().

10.1.2 Function Documentation

10.1.2.1 int inprocess (jack_nframes_t *nframes*, void * *arg*)

Called in the realtime thread on every process cycle. The entry point name was passed to jack_set_process_callback() from jack_initialize(). Although this is an internal client, its process() interface is identical to simple_client.c.

Returns

0 if successful; otherwise jack_finish() will be called and the client terminated immediately.

References port_pair_t::input_port, jack_port_get_buffer(), and port_pair_t::output_port.

Referenced by jack_initialize().

10.1.2.2 void jack_finish (void * *arg*)

This required entry point is called immediately before the client is unloaded, which could happen due to a call to jack_internal_client_unload(), or a nonzero return from either jack_initialize() or inprocess().

Parameters

arg the same parameter provided to inprocess().

10.1.2.3 int jack_initialize (jack_client_t * *client*, const char * *load_init*)

This required entry point is called after the client is loaded by jack_internal_client_load().

Parameters

client pointer to JACK client structure.

load_init character string passed to the load operation.

Returns

0 if successful; otherwise jack_finish() will be called and the client terminated immediately.

References inprocess(), port_pair_t::input_port, jack_activate(), jack_connect(), JACK_DEFAULT_AUDIO_TYPE, jack_port_name(), jack_port_register(), jack_set_process_callback(), JackPortIsInput, JackPortIsOutput, and port_pair_t::output_port.

10.2 intclient.h File Reference

```
#include <jack/types.h>
```

Functions

- char * jack_get_internal_client_name (jack_client_t *client, jack_intclient_t intclient)
- jack_intclient_t jack_internal_client_handle (jack_client_t *client, const char *client_name, jack_status_t *status)
- jack_intclient_t jack_internal_client_load (jack_client_t *client, const char *client_name, jack_options_t options, jack_status_t *status,...)
- jack_status_t jack_internal_client_unload (jack_client_t *client, jack_intclient_t intclient)

10.2.1 Function Documentation

10.2.1.1 char* jack_get_internal_client_name (jack_client_t * *client*, jack_intclient_t *intclient*)

Get an internal client's name. This is useful when JackUseExactName was not specified on jack_internal_client_load() and JackNameNotUnique status was returned. In that case, the actual name will differ from the *client_name* requested.

Parameters

client requesting JACK client's handle.

intclient handle returned from jack_internal_client_load() or jack_internal_client_handle().

Returns

NULL if unsuccessful, otherwise pointer to the internal client name obtained from the heap via malloc(). The caller should free() this storage when no longer needed.

10.2.1.2 jack_intclient_t jack_internal_client_handle (jack_client_t * *client*, const char * *client_name*, jack_status_t * *status*)

Return the jack_intclient_t handle for an internal client running in the JACK server.

Parameters

client requesting JACK client's handle.

client_name for the internal client of no more than jack_client_name_size() characters. The name scope is local to the current server.

status (if non-NULL) an address for JACK to return information from this operation. This status word is formed by OR-ing together the relevant JackStatus bits.

Returns

Opaque internal client handle if successful. If 0, the internal client was not found, and ∗*status* includes the JackNoSuchClient and JackFailure bits.

10.2.1.3 jack_intclient_t jack_internal_client_load (jack_client_t ∗ *client*, const char ∗ *client_name*, jack_options_t *options*, jack_status_t ∗ *status*, ...)

Load an internal client into the JACK server.

Internal clients run inside the JACK server process. They can use most of the same functions as external clients. Each internal client is built as a shared object module, which must declare jack_initialize() and jack_finish() entry points called at load and unload times. See inprocess.c for an example.

Parameters

client loading JACK client's handle.

client_name of at most jack_client_name_size() characters for the internal client to load. The name scope is local to the current server.

options formed by OR-ing together JackOptions bits. Only the JackLoadOptions bits are valid.

status (if non-NULL) an address for JACK to return information from the load operation. This status word is formed by OR-ing together the relevant JackStatus bits.

Optional parameters: depending on corresponding [*options* bits] additional parameters may follow *status* (in this order).

- [JackLoadName] *(char ∗) load_name* is the shared object file from which to load the new internal client (otherwise use the *client_name*).
- [JackLoadInit] *(char ∗) load_init* an arbitary string passed to the internal client's jack_initialize() routine (otherwise NULL), of no more than JACK_LOAD_INIT_LIMIT bytes.

Returns

Opaque internal client handle if successful. If this is 0, the load operation failed, the internal client was not loaded, and ∗*status* includes the JackFailure bit.

10.2.1.4 jack_status_t jack_internal_client_unload (jack_client_t * *client,* jack_intclient_t *intclient*)

Unload an internal client from a JACK server. This calls the intclient's jack_finish() entry point then removes it. See inprocess.c for an example.

Parameters

client unloading JACK client's handle.

intclient handle returned from jack_internal_client_load() or jack_internal_client_handle().

Returns

0 if successful, otherwise JackStatus bits.

10.3 jack.h File Reference

```
#include <pthread.h>
#include <jack/types.h>
#include <jack/transport.h>
#include <jack/weakmacros.h>
```

Functions

- jack_client_t * jack_client_open (const char *client_name, jack_options_t options, jack_status_t *status,...) JACK_OPTIONAL_WEAK_EXPORT
- jack_client_t * jack_client_new (const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_client_close (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_name_size (void) JACK_OPTIONAL_WEAK_EXPORT
- char * jack_get_client_name (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_internal_client_new (const char *client_name, const char *load_name, const char *load_init) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- void jack_internal_client_close (const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_activate (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_deactivate (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT

- pthread_t jack_client_thread_id (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_is_realtime (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_thread_wait (jack_client_t *, int status) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_cycle_wait (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- void jack_cycle_signal (jack_client_t *client, int status) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_process_thread (jack_client_t *client, JackThreadCallback fun, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_thread_init_callback (jack_client_t *client, JackThreadInitCallback thread_init_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- void jack_on_shutdown (jack_client_t *client, JackShutdownCallback function, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- void jack_on_info_shutdown (jack_client_t *client, JackInfoShutdownCallback function, void *arg) JACK_WEAK_EXPORT
- int jack_set_process_callback (jack_client_t *client, JackProcessCallback process_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_freewheel_callback (jack_client_t *client, JackFreewheelCallback freewheel_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_buffer_size_callback (jack_client_t *client, JackBufferSizeCallback bufsize_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sample_rate_callback (jack_client_t *client, JackSampleRateCallback srate_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_client_registration_callback (jack_client_t *, JackClientRegistrationCallback registration_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_port_registration_callback (jack_client_t *, JackPortRegistrationCallback registration_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_port_connect_callback (jack_client_t *, JackPortConnectCallback connect_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_graph_order_callback (jack_client_t *, JackGraphOrderCallback graph_callback, void *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_xrun_callback (jack_client_t *, JackXRunCallback xrun_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_freewheel (jack_client_t *client, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_buffer_size (jack_client_t *client, jack_nframes_t nframes) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_get_sample_rate (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT

- jack_nframes_t jack_get_buffer_size (jack_client_t *) JACK_OPTIONAL_-WEAK_EXPORT
- int jack_engine_takeover_timebase (jack_client_t *) JACK_OPTIONAL_-WEAK_DEPRECATED_EXPORT
- float jack_cpu_load (jack_client_t *client) JACK_OPTIONAL_WEAK_-EXPORT
- jack_port_t * jack_port_register (jack_client_t *client, const char *port_name, const char *port_type, unsigned long flags, unsigned long buffer_size) JACK_-OPTIONAL_WEAK_EXPORT
- int jack_port_unregister (jack_client_t *, jack_port_t *) JACK_OPTIONAL_-WEAK_EXPORT
- void * jack_port_get_buffer (jack_port_t *, jack_nframes_t) JACK_-OPTIONAL_WEAK_EXPORT
- const char * jack_port_name (const jack_port_t *port) JACK_OPTIONAL_-WEAK_EXPORT
- const char * jack_port_short_name (const jack_port_t *port) JACK_-OPTIONAL_WEAK_EXPORT
- int jack_port_flags (const jack_port_t *port) JACK_OPTIONAL_WEAK_-EXPORT
- const char * jack_port_type (const jack_port_t *port) JACK_OPTIONAL_-WEAK_EXPORT
- int jack_port_is_mine (const jack_client_t *, const jack_port_t *port) JACK_-OPTIONAL_WEAK_EXPORT
- int jack_port_connected (const jack_port_t *port) JACK_OPTIONAL_WEAK_-EXPORT
- int jack_port_connected_to (const jack_port_t *port, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- const char ** jack_port_get_connections (const jack_port_t *port) JACK_-OPTIONAL_WEAK_EXPORT
- const char ** jack_port_get_all_connections (const jack_client_t *client, const jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_tie (jack_port_t *src, jack_port_t *dst) JACK_OPTIONAL_-WEAK_DEPRECATED_EXPORT
- int jack_port_untie (jack_port_t *port) JACK_OPTIONAL_WEAK_-DEPRECATED_EXPORT
- jack_nframes_t jack_port_get_latency (jack_port_t *port) JACK_OPTIONAL_-WEAK_EXPORT
- jack_nframes_t jack_port_get_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- void jack_port_set_latency (jack_port_t *, jack_nframes_t) JACK_-OPTIONAL_WEAK_EXPORT
- int jack_recompute_total_latency (jack_client_t *, jack_port_t *port) JACK_-OPTIONAL_WEAK_EXPORT

- int jack_recompute_total_latencies (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_set_name (jack_port_t *port, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_set_alias (jack_port_t *port, const char *alias) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_unset_alias (jack_port_t *port, const char *alias) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_get_aliases (const jack_port_t *port, char *const aliases[2]) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_request_monitor (jack_port_t *port, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_request_monitor_by_name (jack_client_t *client, const char *port_name, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_ensure_monitor (jack_port_t *port, int onoff) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_monitoring_input (jack_port_t *port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_connect (jack_client_t *, const char *source_port, const char *destination_port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_disconnect (jack_client_t *, const char *source_port, const char *destination_port) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_disconnect (jack_client_t *, jack_port_t *) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_name_size (void) JACK_OPTIONAL_WEAK_EXPORT
- int jack_port_type_size (void) JACK_OPTIONAL_WEAK_EXPORT
- const char ** jack_get_ports (jack_client_t *, const char *port_name_pattern, const char *type_name_pattern, unsigned long flags) JACK_OPTIONAL_WEAK_EXPORT
- jack_port_t * jack_port_by_name (jack_client_t *, const char *port_name) JACK_OPTIONAL_WEAK_EXPORT
- jack_port_t * jack_port_by_id (jack_client_t *client, jack_port_id_t port_id) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_frames_since_cycle_start (const jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_frame_time (const jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_last_frame_time (const jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- jack_time_t jack_frames_to_time (const jack_client_t *client, jack_nframes_t) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_time_to_frames (const jack_client_t *client, jack_time_t) JACK_OPTIONAL_WEAK_EXPORT
- jack_time_t jack_get_time () JACK_OPTIONAL_WEAK_EXPORT

- void jack_set_error_function (void(∗func)(const char ∗)) JACK_OPTIONAL_-WEAK_EXPORT
- void jack_set_info_function (void(∗func)(const char ∗)) JACK_OPTIONAL_-WEAK_EXPORT
- void jack_free (void ∗ptr) JACK_OPTIONAL_WEAK_EXPORT

Variables

- void(∗ jack_error_callback)(const char ∗msg) JACK_OPTIONAL_WEAK_-EXPORT
- void(∗ jack_info_callback)(const char ∗msg) JACK_OPTIONAL_WEAK_-EXPORT

10.3.1 Function Documentation

10.3.1.1 void jack_free (void ∗ *ptr*)

The free function to be used on memory returned by jack_port_get_connections, jack_-port_get_all_connections and jack_get_ports functions. This is MANDATORY on Windows when otherwise all nasty runtime version related crashes can occur. Developers are strongly encouraged to use this function instead of the standard "free" function in new code.

10.3.1.2 int jack_is_realtime (jack_client_t ∗ *client*)

Parameters

client pointer to JACK client structure.

Check if the JACK subsystem is running with -R (--realtime).

Returns

1 if JACK is running realtime, 0 otherwise

10.4 mainpage.dox File Reference

10.5 midiport.h File Reference

```
#include <jack/weakmacros.h>
#include <jack/types.h>
#include <stdlib.h>
```

Data Structures

- struct _jack_midi_event

Typedefs

- typedef unsigned char jack_midi_data_t
- typedef struct _jack_midi_event jack_midi_event_t

Functions

- jack_nframes_t jack_midi_get_event_count (void *port_buffer) JACK_-OPTIONAL_WEAK_EXPORT
- int jack_midi_event_get (jack_midi_event_t *event, void *port_buffer, jack_-nframes_t event_index) JACK_OPTIONAL_WEAK_EXPORT
- void jack_midi_clear_buffer (void *port_buffer) JACK_OPTIONAL_WEAK_-EXPORT
- size_t jack_midi_max_event_size (void *port_buffer) JACK_OPTIONAL_-WEAK_EXPORT
- jack_midi_data_t * jack_midi_event_reserve (void *port_buffer, jack_nframes_t time, size_t data_size) JACK_OPTIONAL_WEAK_EXPORT
- int jack_midi_event_write (void *port_buffer, jack_nframes_t time, const jack_-midi_data_t *data, size_t data_size) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_midi_get_lost_event_count (void *port_buffer) JACK_-OPTIONAL_WEAK_EXPORT

10.5.1 Typedef Documentation

10.5.1.1 typedef unsigned char jack_midi_data_t

Type for raw event data contained in jack_midi_event_t.

10.5.1.2 typedef struct _jack_midi_event jack_midi_event_t

A Jack MIDI event.

10.6 porting.dox File Reference

10.7 ringbuffer.h File Reference

```
#include <sys/types.h>
```

Data Structures

- struct jack_ringbuffer_data_t
- struct jack_ringbuffer_t

Functions

- jack_ringbuffer_t * jack_ringbuffer_create (size_t sz)
- void jack_ringbuffer_free (jack_ringbuffer_t *rb)
- void jack_ringbuffer_get_read_vector (const jack_ringbuffer_t *rb, jack_ringbuffer_data_t *vec)
- void jack_ringbuffer_get_write_vector (const jack_ringbuffer_t *rb, jack_ringbuffer_data_t *vec)
- size_t jack_ringbuffer_read (jack_ringbuffer_t *rb, char *dest, size_t cnt)
- size_t jack_ringbuffer_peek (jack_ringbuffer_t *rb, char *dest, size_t cnt)
- void jack_ringbuffer_read_advance (jack_ringbuffer_t *rb, size_t cnt)
- size_t jack_ringbuffer_read_space (const jack_ringbuffer_t *rb)
- int jack_ringbuffer_mlock (jack_ringbuffer_t *rb)
- void jack_ringbuffer_reset (jack_ringbuffer_t *rb)
- size_t jack_ringbuffer_write (jack_ringbuffer_t *rb, const char *src, size_t cnt)
- void jack_ringbuffer_write_advance (jack_ringbuffer_t *rb, size_t cnt)
- size_t jack_ringbuffer_write_space (const jack_ringbuffer_t *rb)

10.7.1 Detailed Description

A set of library functions to make lock-free ringbuffers available to JACK clients. The 'capture_client.c' (in the example_clients directory) is a fully functioning user of this API.

The key attribute of a ringbuffer is that it can be safely accessed by two threads simultaneously -- one reading from the buffer and the other writing to it -- without using any synchronization or mutual exclusion primitives. For this to work correctly, there can only be a single reader and a single writer thread. Their identities cannot be interchanged.

10.7.2 Function Documentation

10.7.2.1 jack_ringbuffer_t* jack_ringbuffer_create (size_t *sz*)

Allocates a ringbuffer data structure of a specified size. The caller must arrange for a call to jack_ringbuffer_free() to release the memory associated with the ringbuffer.

Parameters

sz the ringbuffer size in bytes.

Returns

a pointer to a new jack_ringbuffer_t, if successful; NULL otherwise.

10.7.2.2 void jack_ringbuffer_free (jack_ringbuffer_t * *rb*)

Frees the ringbuffer data structure allocated by an earlier call to jack_ringbuffer_create().

Parameters

rb a pointer to the ringbuffer structure.

10.7.2.3 void jack_ringbuffer_get_read_vector (const jack_ringbuffer_t * *rb*, jack_ringbuffer_data_t * *vec*)

Fill a data structure with a description of the current readable data held in the ringbuffer. This description is returned in a two element array of jack_ringbuffer_data_t. Two elements are needed because the data to be read may be split across the end of the ringbuffer.

The first element will always contain a valid *len* field, which may be zero or greater. If the *len* field is non-zero, then data can be read in a contiguous fashion using the address given in the corresponding *buf* field.

If the second element has a non-zero *len* field, then a second contiguous stretch of data can be read from the address given in its corresponding *buf* field.

Parameters

rb a pointer to the ringbuffer structure.

vec a pointer to a 2 element array of jack_ringbuffer_data_t.

10.7.2.4 void jack_ringbuffer_get_write_vector (const jack_ringbuffer_t * *rb*, jack_ringbuffer_data_t * *vec*)

Fill a data structure with a description of the current writable space in the ringbuffer. The description is returned in a two element array of jack_ringbuffer_data_t. Two elements are needed because the space available for writing may be split across the end of the ringbuffer.

The first element will always contain a valid *len* field, which may be zero or greater. If the *len* field is non-zero, then data can be written in a contiguous fashion using the address given in the corresponding *buf* field.

If the second element has a non-zero *len* field, then a second contiguous stretch of data can be written to the address given in the corresponding *buf* field.

Parameters

rb a pointer to the ringbuffer structure.

vec a pointer to a 2 element array of jack_ringbuffer_data_t.

10.7.2.5 int jack_ringbuffer_mlock (jack_ringbuffer_t * *rb*)

Lock a ringbuffer data block into memory.

Uses the mlock() system call. This is not a realtime operation.

Parameters

rb a pointer to the ringbuffer structure.

10.7.2.6 size_t jack_ringbuffer_peek (jack_ringbuffer_t * *rb*, char * *dest*, size_t *cnt*)

Read data from the ringbuffer. Opposed to jack_ringbuffer_read() this function does not move the read pointer. Thus it's a convenient way to inspect data in the ringbuffer in a continous fashion. The price is that the data is copied into a user provided buffer. For "raw" non-copy inspection of the data in the ringbuffer use jack_ringbuffer_get_read_vector().

Parameters

rb a pointer to the ringbuffer structure.

dest a pointer to a buffer where data read from the ringbuffer will go.

cnt the number of bytes to read.

Returns

the number of bytes read, which may range from 0 to cnt.

10.7.2.7 size_t jack_ringbuffer_read (jack_ringbuffer_t * *rb*, char * *dest*, size_t *cnt*)

Read data from the ringbuffer.

Parameters

rb a pointer to the ringbuffer structure.

dest a pointer to a buffer where data read from the ringbuffer will go.

cnt the number of bytes to read.

Returns

the number of bytes read, which may range from 0 to cnt.

10.7.2.8 void jack_ringbuffer_read_advance (jack_ringbuffer_t * *rb*, size_t *cnt*)

Advance the read pointer.

After data have been read from the ringbuffer using the pointers returned by jack_ringbuffer_get_read_vector(), use this function to advance the buffer pointers, making that space available for future write operations.

Parameters

rb a pointer to the ringbuffer structure.

cnt the number of bytes read.

10.7.2.9 size_t jack_ringbuffer_read_space (const jack_ringbuffer_t * *rb*)

Return the number of bytes available for reading.

Parameters

rb a pointer to the ringbuffer structure.

Returns

the number of bytes available to read.

10.7.2.10 void jack_ringbuffer_reset (jack_ringbuffer_t * *rb*)

Reset the read and write pointers, making an empty buffer.

This is not thread safe.

Parameters

rb a pointer to the ringbuffer structure.

10.7.2.11 size_t jack_ringbuffer_write (jack_ringbuffer_t * *rb*, const char * *src*, size_t *cnt*)

Write data into the ringbuffer.

Parameters

rb a pointer to the ringbuffer structure.

src a pointer to the data to be written to the ringbuffer.

cnt the number of bytes to write.

Returns

the number of bytes write, which may range from 0 to cnt

10.7.2.12 void jack_ringbuffer_write_advance (jack_ringbuffer_t * *rb*, size_t *cnt*)

Advance the write pointer.

After data have been written the ringbuffer using the pointers returned by jack_ringbuffer_get_write_vector(), use this function to advance the buffer pointer, making the data available for future read operations.

Parameters

rb a pointer to the ringbuffer structure.

cnt the number of bytes written.

10.7.2.13 size_t jack_ringbuffer_write_space (const jack_ringbuffer_t * *rb*)

Return the number of bytes available for writing.

Parameters

rb a pointer to the ringbuffer structure.

Returns

the amount of free space (in bytes) available for writing.

10.8 simple_client.c File Reference

This simple client demonstrates the most basic features of JACK as they would be used by many applications.

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <jack/jack.h>
```

Functions

- int process (jack_nframes_t nframes, void *arg)
- void jack_shutdown (void *arg)
- int main (int argc, char *argv[ ])

Variables

- jack_port_t * input_port
- jack_port_t * output_port
- jack_client_t * client

10.8.1 Detailed Description

This simple client demonstrates the most basic features of JACK as they would be used by many applications.

10.8.2 Function Documentation

10.8.2.1 void jack_shutdown (void * *arg*)

JACK calls this shutdown_callback if the server ever shuts down or decides to disconnect the client.

Referenced by main().

10.8.2.2 int main (int *argc,* char * *argv[]*)

References client, input_port, jack_activate(), jack_client_close(), jack_client_open(), jack_connect(), JACK_DEFAULT_AUDIO_TYPE, jack_get_client_name(), jack_get_ports(), jack_get_sample_rate(), jack_on_shutdown(), jack_port_name(), jack_port_register(), jack_set_process_callback(), jack_shutdown(), JackNameNotUnique, JackPortIsInput, JackPortIsOutput, JackPortIsPhysical, JackServerFailed, JackServerStarted, output_port, and process().

10.8.2.3 int process (jack_nframes_t *nframes,* void * *arg*)

The process callback for this JACK application is called in a special realtime thread once for each audio cycle.

This client does nothing more than copy data from its input port to its output port. It will exit when stopped by the user (e.g. using Ctrl-C on a unix-ish operating system)

References input_port, jack_port_get_buffer(), and output_port.

Referenced by main().

10.8.3 Variable Documentation

10.8.3.1 jack_client_t* client

Referenced by main().

10.8.3.2 jack_port_t* input_port

Referenced by main(), and process().

10.8.3.3 jack_port_t* output_port

Referenced by main(), and process().

10.9 statistics.h File Reference

```
#include <jack/types.h>
```

Functions

- float jack_get_max_delayed_usecs (jack_client_t *client)
- float jack_get_xrun_delayed_usecs (jack_client_t *client)
- void jack_reset_max_delayed_usecs (jack_client_t *client)

10.9.1 Function Documentation

10.9.1.1 float jack_get_max_delayed_usecs (jack_client_t * *client*)

Returns

the maximum delay reported by the backend since startup or reset. When compared to the period size in usecs, this can be used to estimate the ideal period size for a given setup.

10.9.1.2 float jack_get_xrun_delayed_usecs (jack_client_t * *client*)

Returns

the delay in microseconds due to the most recent XRUN occurrence. This probably only makes sense when called from a JackXRunCallback defined using jack_set_xrun_callback().

10.9.1.3 void jack_reset_max_delayed_usecs (jack_client_t * *client*)

Reset the maximum delay counter. This would be useful to estimate the effect that a change to the configuration of a running system (e.g. toggling kernel preemption) has on the delay experienced by JACK, without having to restart the JACK engine.

10.10 thread.h File Reference

```
#include <pthread.h>
#include <jack/weakmacros.h>
```

Defines

- #define THREAD_STACK 524288

Typedefs

- typedef int(∗ jack_thread_creator_t)(pthread_t ∗, const pthread_attr_t ∗, void ∗(∗function)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT

Functions

- int jack_client_real_time_priority (jack_client_t ∗) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_max_real_time_priority (jack_client_t ∗) JACK_OPTIONAL_WEAK_EXPORT
- int jack_acquire_real_time_scheduling (pthread_t thread, int priority) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_create_thread (jack_client_t ∗client, pthread_t ∗thread, int priority, int realtime, void ∗(∗start_routine)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_drop_real_time_scheduling (pthread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- void jack_set_thread_creator (jack_thread_creator_t creator) JACK_OPTIONAL_WEAK_EXPORT

10.10.1 Detailed Description

Library functions to standardize thread creation for JACK and its clients. These interfaces hide some system variations in the handling of realtime scheduling and associated privileges.

10.10.2 Define Documentation

10.10.2.1 #define THREAD_STACK 524288

10.11 transport.dox File Reference

10.12 transport.h File Reference

```
#include <jack/types.h>
#include <jack/weakmacros.h>
```

Data Structures

- struct jack_position_t
- struct jack_transport_info_t

Defines

- #define JACK_POSITION_MASK (JackPositionBBT|JackPositionTimecode|JackBBTFrameOffset|JackAu
- #define EXTENDED_TIME_INFO

Typedefs

- typedef uint64_t jack_unique_t
- typedef int(* JackSyncCallback)(jack_transport_state_t state, jack_position_t *pos, void *arg)
- typedef void(* JackTimebaseCallback)(jack_transport_state_t state, jack_nframes_t nframes, jack_position_t *pos, int new_pos, void *arg)

Enumerations

- enum jack_transport_state_t { JackTransportStopped = 0, JackTransportRolling = 1, JackTransportLooping = 2, JackTransportStarting = 3 }
- enum jack_position_bits_t {

  JackPositionBBT = 0x10, JackPositionTimecode = 0x20, JackBBTFrameOffset = 0x40, JackAudioVideoRatio = 0x80,

  JackVideoFrameOffset = 0x100 }
- enum jack_transport_bits_t {

  JackTransportState = 0x1, JackTransportPosition = 0x2, JackTransportLoop = 0x4, JackTransportSMPTE = 0x8,

  JackTransportBBT = 0x10 }

Functions

- int jack_release_timebase (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sync_callback (jack_client_t *client, JackSyncCallback sync_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_set_sync_timeout (jack_client_t *client, jack_time_t timeout) JACK_OPTIONAL_WEAK_EXPORT

- int jack_set_timebase_callback (jack_client_t *client, int conditional, JackTimebaseCallback timebase_callback, void *arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_transport_locate (jack_client_t *client, jack_nframes_t frame) JACK_OPTIONAL_WEAK_EXPORT
- jack_transport_state_t jack_transport_query (const jack_client_t *client, jack_position_t *pos) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_get_current_transport_frame (const jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- int jack_transport_reposition (jack_client_t *client, jack_position_t *pos) JACK_OPTIONAL_WEAK_EXPORT
- void jack_transport_start (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- void jack_transport_stop (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT
- void jack_get_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- void jack_set_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

10.12.1 Define Documentation

10.12.1.1 #define EXTENDED_TIME_INFO

10.12.1.2 #define JACK_POSITION_MASK (JackPositionBBT|JackPositionTimecode|JackBBTFrameOffset|JackAudioVideoRatio|JackVideoFr

all valid position bits

10.12.2 Typedef Documentation

10.12.2.1 typedef uint64_t jack_unique_t

Unique ID (opaque)

10.12.3 Enumeration Type Documentation

10.12.3.1 enum jack_position_bits_t

Optional struct jack_position_t fields.

Enumerator:

JackPositionBBT Bar, Beat, Tick

JackPositionTimecode External timecode
JackBBTFrameOffset Frame offset of BBT information
JackAudioVideoRatio audio frames per video frame
JackVideoFrameOffset frame offset of first video frame

10.12.3.2 enum jack_transport_bits_t

Optional struct jack_transport_info_t fields.

See also

jack_position_bits_t.

Enumerator:

JackTransportState Transport state
JackTransportPosition Frame number
JackTransportLoop Loop boundaries (ignored)
JackTransportSMPTE SMPTE (ignored)
JackTransportBBT Bar, Beat, Tick

10.12.3.3 enum jack_transport_state_t

Transport states.

Enumerator:

JackTransportStopped Transport halted
JackTransportRolling Transport playing
JackTransportLooping For OLD_TRANSPORT, now ignored
JackTransportStarting Waiting for sync ready

10.12.4 Function Documentation

10.12.4.1 void jack_get_transport_info (jack_client_t * *client,* jack_transport_info_t * *tinfo*)

Gets the current transport info structure (deprecated).

Parameters

client the JACK client structure.

tinfo current transport info structure. The "valid" field describes which fields contain valid data.

Deprecated

This is for compatibility with the earlier transport interface. Use jack_transport_query(), instead.

Precondition

Must be called from the process thread.

10.12.4.2 void jack_set_transport_info (jack_client_t * *client*, jack_transport_info_t * *tinfo*)

Set the transport info structure (deprecated).

Deprecated

This function still exists for compatibility with the earlier transport interface, but it does nothing. Instead, define a JackTimebaseCallback.

10.13 types.h File Reference

```
#include <inttypes.h>
```

Defines

- #define JACK_MAX_FRAMES (4294967295U)
- #define JACK_LOAD_INIT_LIMIT 1024
- #define JackOpenOptions (JackSessionID|JackServerName|JackNoStartServer|JackUseExactName)
- #define JackLoadOptions (JackLoadInit|JackLoadName|JackUseExactName)
- #define JACK_DEFAULT_AUDIO_TYPE "32 bit float mono audio"
- #define JACK_DEFAULT_MIDI_TYPE "8 bit raw midi"

Typedefs

- typedef int32_t jack_shmsize_t
- typedef uint32_t jack_nframes_t
- typedef uint64_t jack_time_t
- typedef uint64_t jack_intclient_t

- typedef struct _jack_port jack_port_t
- typedef struct _jack_client jack_client_t
- typedef uint32_t jack_port_id_t
- typedef enum JackOptions jack_options_t
- typedef enum JackStatus jack_status_t
- typedef int(* JackProcessCallback)(jack_nframes_t nframes, void *arg)
- typedef void(* JackThreadInitCallback)(void *arg)
- typedef int(* JackGraphOrderCallback)(void *arg)
- typedef int(* JackXRunCallback)(void *arg)
- typedef int(* JackBufferSizeCallback)(jack_nframes_t nframes, void *arg)
- typedef int(* JackSampleRateCallback)(jack_nframes_t nframes, void *arg)
- typedef void(* JackPortRegistrationCallback)(jack_port_id_t port, int register, void *arg)
- typedef void(* JackClientRegistrationCallback)(const char *name, int register, void *arg)
- typedef void(* JackPortConnectCallback)(jack_port_id_t a, jack_port_id_t b, int connect, void *arg)
- typedef void(* JackFreewheelCallback)(int starting, void *arg)
- typedef void *(* JackThreadCallback)(void *arg)
- typedef void(* JackShutdownCallback)(void *arg)
- typedef void(* JackInfoShutdownCallback)(jack_status_t code, const char *reason, void *arg)
- typedef float jack_default_audio_sample_t

Enumerations

- enum JackOptions {

  JackNullOption = 0x00, JackNoStartServer = 0x01, JackUseExactName = 0x02, JackServerName = 0x04,

  JackLoadName = 0x08, JackLoadInit = 0x10, JackSessionID = 0x20 }
- enum JackStatus {

  JackFailure = 0x01, JackInvalidOption = 0x02, JackNameNotUnique = 0x04, JackServerStarted = 0x08,

  JackServerFailed = 0x10, JackServerError = 0x20, JackNoSuchClient = 0x40, JackLoadFailure = 0x80,

  JackInitFailure = 0x100, JackShmFailure = 0x200, JackVersionError = 0x400, JackBackendError = 0x800,

  JackClientZombie = 0x1000 }
- enum JackPortFlags {

  JackPortIsInput = 0x1, JackPortIsOutput = 0x2, JackPortIsPhysical = 0x4, JackPortCanMonitor = 0x8,

  JackPortIsTerminal = 0x10 }

10.13.1 Define Documentation

10.13.1.1 #define JACK_DEFAULT_AUDIO_TYPE "32 bit float mono audio"

Used for the type argument of jack_port_register() for default audio and midi ports.

Referenced by jack_initialize(), and main().

10.13.1.2 #define JACK_DEFAULT_MIDI_TYPE "8 bit raw midi"

10.13.1.3 #define JACK_LOAD_INIT_LIMIT 1024

Maximum size of *load_init* string passed to an internal client jack_initialize() function via jack_internal_client_load().

10.13.1.4 #define JACK_MAX_FRAMES (4294967295U)

Maximum value that can be stored in jack_nframes_t

10.13.1.5 #define JackLoadOptions (JackLoadInit|JackLoadName|JackUseExactName)

Valid options for loading an internal client.

10.13.1.6 #define JackOpenOptions (JackSessionID|JackServerName|JackNoStartServer|JackUseExactName)

Valid options for opening an external client.

10.13.2 Typedef Documentation

10.13.2.1 typedef struct _jack_client jack_client_t

jack_client_t is an opaque type. You may only access it using the API provided.

10.13.2.2 typedef float jack_default_audio_sample_t

For convenience, use this typedef if you want to be able to change between float and double. You may want to typedef sample_t to jack_default_audio_sample_t in your application.

10.13.2.3 typedef uint64_t jack_intclient_t

jack_intclient_t is an opaque type representing a loaded internal client. You may only access it using the API provided in <jack/intclient.h>.

10.13.2.4 typedef uint32_t jack_nframes_t

Type used to represent sample frame counts.

10.13.2.5 typedef enum JackOptions jack_options_t

Options for several JACK operations, formed by OR-ing together the relevant JackOptions bits.

10.13.2.6 typedef uint32_t jack_port_id_t

Ports have unique ids. A port registration callback is the only place you ever need to know their value.

10.13.2.7 typedef struct _jack_port jack_port_t

jack_port_t is an opaque type. You may only access it using the API provided.

10.13.2.8 typedef int32_t jack_shmsize_t

10.13.2.9 typedef enum JackStatus jack_status_t

Status word returned from several JACK operations, formed by OR-ing together the relevant JackStatus bits.

10.13.2.10 typedef uint64_t jack_time_t

Type used to represent the value of free running monotonic clock with units of microseconds.

10.13.2.11 typedef int(* JackBufferSizeCallback)(jack_nframes_t nframes, void *arg)

Prototype for the *bufsize_callback* that is invoked whenever the JACK engine buffer size changes. Although this function is called in the JACK process thread, the normal

process cycle is suspended during its operation, causing a gap in the audio flow. So, the *bufsize_callback* can allocate storage, touch memory not previously referenced, and perform other operations that are not realtime safe.

Parameters

nframes buffer size

arg pointer supplied by jack_set_buffer_size_callback().

Returns

zero on success, non-zero on error

10.13.2.12 typedef void(* JackClientRegistrationCallback)(const char *name, int register, void *arg)

Prototype for the client supplied function that is called whenever a client is registered or unregistered.

Parameters

name a null-terminated string containing the client name

register non-zero if the client is being registered, zero if the client is being unregistered

arg pointer to a client supplied data

10.13.2.13 typedef void(* JackFreewheelCallback)(int starting, void *arg)

Prototype for the client supplied function that is called whenever jackd starts or stops freewheeling.

Parameters

starting non-zero if we start starting to freewheel, zero otherwise

arg pointer to a client supplied structure

10.13.2.14 typedef int(* JackGraphOrderCallback)(void *arg)

Prototype for the client supplied function that is called whenever the processing graph is reordered.

Parameters

arg pointer to a client supplied data

Returns

zero on success, non-zero on error

10.13.2.15 typedef void(* JackInfoShutdownCallback)(jack_status_t code, const char *reason, void *arg)

Prototype for the client supplied function that is called whenever jackd is shutdown. Note that after server shutdown, the client pointer is *not* deallocated by libjack, the application is responsible to properly use jack_client_close() to release client ressources. Warning: jack_client_close() cannot be safely used inside the shutdown callback and has to be called outside of the callback context.

Parameters

code a shutdown code

reason a string describing the shutdown reason (backend failure, server crash... etc...)

arg pointer to a client supplied structure

10.13.2.16 typedef void(* JackPortConnectCallback)(jack_port_id_t a, jack_port_id_t b, int connect, void *arg)

Prototype for the client supplied function that is called whenever a client is registered or unregistered.

Parameters

a one of two ports connected or disconnected

b one of two ports connected or disconnected

connect non-zero if ports were connected zero if ports were disconnected

arg pointer to a client supplied data

10.13.2.17 typedef void(* JackPortRegistrationCallback)(jack_port_id_t port, int register, void *arg)

Prototype for the client supplied function that is called whenever a port is registered or unregistered.

Parameters

port the ID of the port

arg pointer to a client supplied data

register non-zero if the port is being registered, zero if the port is being unregistered

10.13.2.18 typedef int(* JackProcessCallback)(jack_nframes_t nframes, void *arg)

Prototype for the client supplied function that is called by the engine anytime there is work to be done.

Precondition

nframes == jack_get_buffer_size()
nframes == pow(2,x)

Parameters

nframes number of frames to process

arg pointer to a client supplied data

Returns

zero on success, non-zero on error

10.13.2.19 typedef int(* JackSampleRateCallback)(jack_nframes_t nframes, void *arg)

Prototype for the client supplied function that is called when the engine sample rate changes.

Parameters

nframes new engine sample rate

arg pointer to a client supplied data

Returns

zero on success, non-zero on error

10.13.2.20 typedef void(* JackShutdownCallback)(void *arg)

Prototype for the client supplied function that is called whenever jackd is shutdown. Note that after server shutdown, the client pointer is *not* deallocated by libjack, the application is responsible to properly use jack_client_close() to release client ressources. Warning: jack_client_close() cannot be safely used inside the shutdown callback and has to be called outside of the callback context.

Parameters

arg pointer to a client supplied structure

10.13.2.21 typedef void*(* JackThreadCallback)(void *arg)

10.13.2.22 typedef void(* JackThreadInitCallback)(void *arg)

Prototype for the client supplied function that is called once after the creation of the thread in which other callbacks will be made. Special thread characteristics can be set from this callback, for example. This is a highly specialized callback and most clients will not and should not use it.

Parameters

arg pointer to a client supplied structure

Returns

void

10.13.2.23 typedef int(* JackXRunCallback)(void *arg)

Prototype for the client-supplied function that is called whenever an xrun has occured.

See also

jack_get_xrun_delayed_usecs()

Parameters

arg pointer to a client supplied data

Returns

zero on success, non-zero on error

10.13.3 Enumeration Type Documentation

10.13.3.1 enum JackOptions

jack_options_t bits

Enumerator:

JackNullOption Null value to use when no option bits are needed.

JackNoStartServer Do not automatically start the JACK server when it is not already running. This option is always selected if $JACK_NO_START_-SERVER is defined in the calling process environment.

JackUseExactName Use the exact client name requested. Otherwise, JACK automatically generates a unique one, if needed.

JackServerName Open with optional *(char ∗) server_name* parameter.

JackLoadName Load internal client from optional *(char ∗) load_name*. Otherwise use the *client_name*.

JackLoadInit Pass optional *(char ∗) load_init* string to the jack_initialize() entry point of an internal client.

JackSessionID pass a SessionID Token this allows the sessionmanager to identify the client again.

10.13.3.2 enum JackPortFlags

A port has a set of flags that are formed by AND-ing together the desired values from the list below. The flags "JackPortIsInput" and "JackPortIsOutput" are mutually exclusive and it is an error to use them both.

Enumerator:

JackPortIsInput if JackPortIsInput is set, then the port can receive data.

JackPortIsOutput if JackPortIsOutput is set, then data can be read from the port.

JackPortIsPhysical if JackPortIsPhysical is set, then the port corresponds to some kind of physical I/O connector.

JackPortCanMonitor if JackPortCanMonitor is set, then a call to jack_port_-request_monitor() makes sense.

Precisely what this means is dependent on the client. A typical result of it being called with TRUE as the second argument is that data that would be available from an output port (with JackPortIsPhysical set) is sent to a physical output connector as well, so that it can be heard/seen/whatever.

Clients that do not control physical interfaces should never create ports with this bit set.

JackPortIsTerminal JackPortIsTerminal means:

for an input port: the data received by the port will not be passed on or made available at any other port

for an output port: the data available at the port does not originate from any other port

Audio synthesizers, I/O hardware interface clients, HDR systems are examples of clients that would set this flag for their ports.

10.13.3.3 enum JackStatus

jack_status_t bits

Enumerator:

JackFailure Overall operation failed.

JackInvalidOption The operation contained an invalid or unsupported option.

JackNameNotUnique The desired client name was not unique. With the JackUseExactName option this situation is fatal. Otherwise, the name was modified by appending a dash and a two-digit number in the range "-01" to "-99". The jack_get_client_name() function will return the exact string that was used. If the specified *client_name* plus these extra characters would be too long, the open fails instead.

JackServerStarted The JACK server was started as a result of this operation. Otherwise, it was running already. In either case the caller is now connected to jackd, so there is no race condition. When the server shuts down, the client will find out.

JackServerFailed Unable to connect to the JACK server.

JackServerError Communication error with the JACK server.

JackNoSuchClient Requested client does not exist.

JackLoadFailure Unable to load internal client

JackInitFailure Unable to initialize client

JackShmFailure Unable to access shared memory

JackVersionError Client's protocol version does not match

JackBackendError

JackClientZombie

Index