EAR

Generated by Doxygen 1.7.6.1

Tue Sep 4 2012 14:51:54

Contents

Chapter 1

EAR Audio Rectifier

Description

This project started as an academic elaboration at the University of Applied Science in Iserlohn, the Institute CV & CI (Institute for Computer Science, Vision and - Computational Intelligence). EAR tries to compensate the difference between the audio signal source as a reference and the measured input obtained by a microphone. This way, EAR tries to compensate the impacts of the loudspeakers on the music.

Jack

JACK is a system for handling real-time, low latency audio (and MIDI). It runs on GNU/Linux, Solaris, FreeBSD, OS X and Windows (and can be ported to other POSIX-conformant platforms). It can connect a number of different applications to an audio device, as well as allowing them to share audio between themselves. Its clients can run in their own processes (ie. as normal applications), or can they can run within the JACK server (ie. as a "plugin"). JACK also has support for distributing audio processing across a network, both fast & reliable LANs as well as slower, less reliable WANs.

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.

This description was copied from `Jack Audio Connection Kit - - Copyright 2001-2006 Paul Davis` at 21.09.2011.

Setting up EAR

EAR is running on top of JACK. At first, you need to download and install JACK for your operating system. Though JACK is running as a separate process and can be configured via the command line it is recommended to use QJackControl, which provides a graphical user interface that allows you to set up the JACK audio server, draw connections and view system messages.

Required JACK server Settings for Windows XP and Windows 7:

- Driver: portaudio
- Real-time: On
- Frames: 4096
- SampleRate: 44100
- Buffer: 2

Required JACK server settings for Ubuntu:

- Driver: alsa
- Real-time: On
- Frames: 4096
- SampleRate: 44100
- Buffer: 2

Make sure the server is running and launch EAR. Connect your music player of choice to the 'source' input. On Windows, the developers have successfully used Mixxx, which provides native JACK support on Windows. Connect the 'main'-output to your speakers. Now you should be able to loop through music. Finally, connect a microphone to your computer and connect it to the 'mic'-input in JACK.

Latency Calibration

To achieve best results, you will need to find out the latency for the regulating loop. - Click on 'Calibrate'. This will send out a click tone on your speakers that will be received through the microphone. As soon as the click sound will be received, the next will be send. Try to avoid making noise during the calibration process.

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Chapter 2

Deprecated List

Global jack_client_new (const char *client_name) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

Please use jack_client_open().

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_EXPORT

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_OPTIONAL_WEAK_DEPRECATED_EXPORT

Please use jack_internal_client_load().

Global jack_port_get_latency (jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_port_get_latency_range() in any existing use cases.

Global jack_port_get_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_port_get_latency_range() in any existing use cases.

Global jack_port_set_latency (jack_port_t *, jack_nframes_t) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by a latency callback that calls jack_port_set_latency_range().

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_recompute_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_recompute_total_latencies() in any existing use cases.

Global jack_set_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_EXPORT

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

Global jack_thread_wait (jack_client_t *, int status) JACK_OPTIONAL_WEAK_EXPORT

Please use jack_cycle_wait() and jack_cycle_signal() functions.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Data Structure Index

5.1 Class Hierarchy

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

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 Controling the server

Functions

- sigset_t jackctl_setup_signals (unsigned int flags)
- void jackctl_wait_signals (sigset_t signals)
- jackctl_server_t * jackctl_server_create (bool(*on_device_acquire)(const char *device_name), void(*on_device_release)(const char *device_name))
- void jackctl_server_destroy (jackctl_server_t *server)
- bool jackctl_server_open (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_start (jackctl_server_t *server)
- bool jackctl_server_stop (jackctl_server_t *server)
- bool jackctl_server_close (jackctl_server_t *server)
- const JSList * jackctl_server_get_drivers_list (jackctl_server_t *server)
- const JSList * jackctl_server_get_parameters (jackctl_server_t *server)
- const JSList * jackctl_server_get_internals_list (jackctl_server_t *server)
- bool jackctl_server_load_internal (jackctl_server_t *server, jackctl_internal_t *internal)
- bool jackctl_server_unload_internal (jackctl_server_t *server, jackctl_internal_t *internal)
- bool jackctl_server_add_slave (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_remove_slave (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_switch_master (jackctl_server_t *server, jackctl_driver_t *driver)
- const char * jackctl_driver_get_name (jackctl_driver_t *driver)
- const JSList * jackctl_driver_get_parameters (jackctl_driver_t *driver)
- const char * jackctl_internal_get_name (jackctl_internal_t *internal)
- const JSList * jackctl_internal_get_parameters (jackctl_internal_t *internal)
- const char * jackctl_parameter_get_name (jackctl_parameter_t *parameter)

- const char * jackctl_parameter_get_short_description (jackctl_parameter_t *parameter)
- const char * jackctl_parameter_get_long_description (jackctl_parameter_t *parameter)
- jackctl_param_type_t jackctl_parameter_get_type (jackctl_parameter_t *parameter)
- char jackctl_parameter_get_id (jackctl_parameter_t *parameter)
- bool jackctl_parameter_is_set (jackctl_parameter_t *parameter)
- bool jackctl_parameter_reset (jackctl_parameter_t *parameter)
- union jackctl_parameter_value jackctl_parameter_get_value (jackctl_parameter_t *parameter)
- bool jackctl_parameter_set_value (jackctl_parameter_t *parameter, const union jackctl_parameter_value *value_ptr)
- union jackctl_parameter_value jackctl_parameter_get_default_value (jackctl_parameter_t *parameter)
- bool jackctl_parameter_has_range_constraint (jackctl_parameter_t *parameter)
- bool jackctl_parameter_has_enum_constraint (jackctl_parameter_t *parameter)
- uint32_t jackctl_parameter_get_enum_constraints_count (jackctl_parameter_t *parameter)
- union jackctl_parameter_value jackctl_parameter_get_enum_constraint_value (jackctl_parameter_t *parameter, uint32_t index)
- const char * jackctl_parameter_get_enum_constraint_description (jackctl_parameter_t *parameter, uint32_t index)
- void jackctl_parameter_get_range_constraint (jackctl_parameter_t *parameter, union jackctl_parameter_value *min_ptr, union jackctl_parameter_value *max_ptr)
- bool jackctl_parameter_constraint_is_strict (jackctl_parameter_t *parameter)
- bool jackctl_parameter_constraint_is_fake_value (jackctl_parameter_t *parameter)
- void jack_error (const char *format,...)
- void jack_info (const char *format,...)
- void jack_log (const char *format,...)

8.1.1 Function Documentation

8.1.1.1 void jack_error (const char * *format*, ...)

Call this function to log an error message.

Parameters

format	string

8.1.1.2 void jack_info (const char * *format*, ...)

Call this function to log an information message.

Parameters

format	string

8.1.1.3 void jack_log (const char * *format, ...*)

Call this function to log an information message but only when verbose mode is enabled.

Parameters

format	string

8.1.1.4 const char* jackctl_driver_get_name (jackctl_driver_t * *driver*)

Call this function to get name of driver.

Parameters

driver	driver object handle to get name of

Returns

driver name. Must not be modified. Always same for same driver object.

8.1.1.5 const JSList* jackctl_driver_get_parameters (jackctl_driver_t * *driver*)

Call this function to get list of driver parameters. List node data pointers is a parameter object handle (::jackctl_parameter_t).

Parameters

driver	driver object handle to get parameters for

Returns

Single linked list of parameter object handles. Must not be modified. Always same for same driver object.

8.1.1.6 const char* jackctl_internal_get_name (jackctl_internal_t * *internal*)

Call this function to get name of internal client.

Parameters

internal	internal object handle to get name of

Returns

internal name. Must not be modified. Always same for same internal object.

8.1.1.7 const JSList* jackctl_internal_get_parameters (jackctl_internal_t * *internal*)

Call this function to get list of internal parameters. List node data pointers is a parameter object handle (::jackctl_parameter_t).

Parameters

internal	internal object handle to get parameters for

Returns

Single linked list of parameter object handles. Must not be modified. Always same for same internal object.

8.1.1.8 bool jackctl_parameter_constraint_is_fake_value (jackctl_parameter_t * *parameter*)

Call this function to check whether parameter has fake values, i.e. values have no user meaningful meaning and only value description is meaningful to user.

Parameters

parameter	parameter object handle to check

Returns

whether parameter constraint is strict.

8.1.1.9 bool jackctl_parameter_constraint_is_strict (jackctl_parameter_t * *parameter*)

Call this function to check whether parameter constraint is strict, i.e. whether supplying non-matching value will not work for sure.

Parameters

parameter	parameter object handle to check

Returns

whether parameter constraint is strict.

8.1.1.10 union jackctl_parameter_value jackctl_parameter_get_default_value (jackctl_parameter_t * *parameter*) [write]

Call this function to get parameter default value.

Parameters

parameter	parameter object handle to get default value of

Returns

parameter default value.

8.1.1.11 const char* jackctl_parameter_get_enum_constraint_description (jackctl_parameter_t * *parameter,* uint32_t *index*)

Call this function to get parameter enumeration value description.

Parameters

parameter	object handle of parameter
index	index of parameter enumeration value

Returns

enumeration value description.

8.1.1.12 union jackctl_parameter_value jackctl_parameter_get_enum_constraint_value (jackctl_parameter_t * *parameter,* uint32_t *index*) [write]

Call this function to get parameter enumeration value.

Parameters

parameter	object handle of parameter
index	index of parameter enumeration value

Returns

enumeration value.

8.1.1.13 uint32_t jackctl_parameter_get_enum_constraints_count (jackctl_parameter_t * *parameter*)

Call this function get how many enumeration values parameter has.

Parameters

parameter	object handle of parameter

Returns

number of enumeration values

8.1.1.14 char jackctl_parameter_get_id (jackctl_parameter_t * *parameter*)

Call this function to get parameter character.

Parameters

parameter	parameter object handle to get character of

Returns

character.

8.1.1.15 const char* jackctl_parameter_get_long_description (jackctl_parameter_t * *parameter*)

Call this function to get parameter long description.

Parameters

parameter	parameter object handle to get long description of

Returns

parameter long description. Must not be modified. Always same for same parameter object.

8.1.1.16 const char* jackctl_parameter_get_name (jackctl_parameter_t * *parameter*)

Call this function to get parameter name.

Parameters

parameter	parameter object handle to get name of

Returns

parameter name. Must not be modified. Always same for same parameter object.

8.1.1.17 void jackctl_parameter_get_range_constraint (jackctl_parameter_t * *parameter,* union jackctl_parameter_value * *min_ptr,* union jackctl_parameter_value * *max_ptr*)

Call this function to get parameter range.

Parameters

parameter	object handle of parameter
min_ptr	pointer to variable receiving parameter minimum value
max_ptr	pointer to variable receiving parameter maximum value

8.1.1.18 const char* jackctl_parameter_get_short_description (jackctl_parameter_t * *parameter*)

Call this function to get parameter short description.

Parameters

parameter	parameter object handle to get short description of

Returns

parameter short description. Must not be modified. Always same for same parameter object.

8.1.1.19 jackctl_param_type_t jackctl_parameter_get_type (jackctl_parameter_t * *parameter*)

Call this function to get parameter type.

Parameters

parameter	parameter object handle to get type of

Returns

parameter type. Always same for same parameter object.

8.1.1.20 union jackctl_parameter_value jackctl_parameter_get_value (jackctl_parameter_t * *parameter*) [write]

Call this function to get parameter value.

Parameters

parameter	parameter object handle to get value of

Returns

parameter value.

8.1.1.21 bool jackctl_parameter_has_enum_constraint (jackctl_parameter_t * *parameter*)

Call this function check whether parameter has enumeration constraint.

Parameters

parameter	object handle of parameter to check

Returns

whether parameter has enumeration constraint.

8.1.1.22 bool jackctl_parameter_has_range_constraint (jackctl_parameter_t * *parameter*)

Call this function check whether parameter has range constraint.

Parameters

parameter	object handle of parameter to check

Returns

whether parameter has range constraint.

8.1.1.23 bool jackctl_parameter_is_set (jackctl_parameter_t * *parameter*)

Call this function to check whether parameter has been set, or its default value is being used.

Parameters

parameter	parameter object handle to check

Returns

true - parameter is set, false - parameter is using default value.

8.1.1.24 bool jackctl_parameter_reset (jackctl_parameter_t * *parameter*)

Call this function to reset parameter to its default value.

Parameters

parameter	parameter object handle to reset value of

Returns

success status: true - success, false - fail

8.1.1.25 bool jackctl_parameter_set_value (jackctl_parameter_t * *parameter,* const union jackctl_parameter_value * *value_ptr*)

Call this function to set parameter value.

Parameters

parameter	parameter object handle to get value of
value_ptr	pointer to variable containing parameter value

Returns

success status: true - success, false - fail

8.1.1.26 bool jackctl_server_add_slave (jackctl_server_t * *server,* jackctl_driver_t * *driver*)

Call this function to add a slave in the driver slave list. (cannot be used when the server is running that is between jackctl_server_start and jackctl_server_stop)

Parameters

server	server object handle
driver	driver to add in the driver slave list.

Returns

success status: true - success, false - fail

8.1.1.27 bool jackctl_server_close (jackctl_server_t * *server*)

Call this function to close JACK server

Parameters

server	server object handle

Returns

success status: true - success, false - fail

8.1.1.28 jackctl_server_t* jackctl_server_create (bool(*)(const char *device_name) *on_device_acquire*, void(*)(const char *device_name) *on_device_release*)

Call this function to create server object.

Parameters

on_device_acquire	- Optional callback to be called before device is acquired. If false is returned, device usage will fail
on_device_release	- Optional callback to be called after device is released.

Returns

server object handle, NULL if creation of server object failed. Successfully created server object must be destroyed with paired call to jackctl_server_destroy

8.1.1.29 void jackctl_server_destroy (jackctl_server_t * *server*)

Call this function to destroy server object.

Parameters

server	server object handle to destroy

8.1.1.30 const JSList* jackctl_server_get_drivers_list (jackctl_server_t * *server*)

Call this function to get list of available drivers. List node data pointers is a driver object handle (::jackctl_driver_t).

Parameters

server	server object handle to get drivers for

Returns

Single linked list of driver object handles. Must not be modified. Always same for same server object.

8.1.1.31 const JSList* jackctl_server_get_internals_list (jackctl_server_t * *server*)

Call this function to get list of available internal clients. List node data pointers is a internal client object handle (::jackctl_internal_t).

Parameters

server	server object handle to get internal clients for

Returns

Single linked list of internal client object handles. Must not be modified. Always same for same server object.

8.1.1.32 const JSList* jackctl_server_get_parameters (jackctl_server_t * *server*)

Call this function to get list of server parameters. List node data pointers is a parameter object handle (::jackctl_parameter_t).

Parameters

server	server object handle to get parameters for

Returns

Single linked list of parameter object handles. Must not be modified. Always same for same server object.

8.1.1.33 bool jackctl_server_load_internal (jackctl_server_t * *server,* jackctl_internal_t * *internal*)

Call this function to load one internal client. (can be used when the server is running)

Parameters

server	server object handle
internal	internal to use

Returns

success status: true - success, false - fail

8.1.1.34 bool jackctl_server_open (jackctl_server_t * *server,* jackctl_driver_t * *driver*)

Call this function to open JACK server

Parameters

server	server object handle
driver	driver to use

Returns

success status: true - success, false - fail

8.1.1.35 bool jackctl_server_remove_slave (jackctl_server_t * *server,* jackctl_driver_t * *driver*)

Call this function to remove a slave from the driver slave list. (cannot be used when the server is running that is between jackctl_server_start and jackctl_server_stop)

Parameters

server	server object handle
driver	driver to remove from the driver slave list.

Returns

success status: true - success, false - fail

8.1.1.36 bool jackctl_server_start (jackctl_server_t * *server*)

Call this function to start JACK server

Parameters

server	server object handle

Returns

success status: true - success, false - fail

8.1.1.37 bool jackctl_server_stop (jackctl_server_t * *server*)

Call this function to stop JACK server

Parameters

server	server object handle

Returns

success status: true - success, false - fail

8.1.1.38 bool jackctl_server_switch_master (jackctl_server_t * *server,* jackctl_driver_t * *driver*)

Call this function to switch master driver.

Parameters

server	server object handle
driver	driver to switch to

Returns

success status: true - success, false - fail

8.1.1.39 bool jackctl_server_unload_internal (jackctl_server_t * *server,* jackctl_internal_t * *internal*)

Call this function to unload one internal client. (can be used when the server is running)

Parameters

server	server object handle
internal	internal to unload

Returns

success status: true - success, false - fail

8.1.1.40 sigset_t jackctl_setup_signals (unsigned int *flags*)

Call this function to setup process signal handling. As a general rule, it is required for proper operation for the server object.

Parameters

flags	signals setup flags, use 0 for none. Currently no flags are defined

Returns

the configurated signal set.

8.1.1.41 void jackctl_wait_signals (sigset_t *signals*)

Call this function to wait on a signal set.

Parameters

signals	signals set to wait on

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 thread_callback, 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 *thread_callback,* 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. See http://jackit.sourceforge.net/docs/design/design.-html#SECTION00411000000000000000 for more information.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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.

Deprecated Please use 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 shutdown_callback, void *arg) JACK_WEAK_EXPORT
- void jack_on_info_shutdown (jack_client_t *client, JackInfoShutdownCallback shutdown_callback, void *arg) JACK_OPTIONAL_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_port_rename_callback (jack_client_t *, JackPortRenameCallback rename_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 *shutdown_callback*, void * *arg*)

Parameters

client	pointer to JACK client structure.
function	The jack_info_shutdown function pointer.
arg	The arguments for the jack_info_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.

8.3.1.2 void jack_on_shutdown (jack_client_t * *client*, JackShutdownCallback *shutdown_callback*, 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.

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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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

Referenced by JackAdapter::connectToServer().

Here is the caller graph for this function:

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

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

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

Returns

0 on success, otherwise a non-zero error code

8.3.1.9 int jack_set_port_rename_callback (jack_client_t * , JackPortRenameCallback *rename_callback,* void * *arg*)

Tell the JACK server to call *rename_callback* whenever a port is renamed, passing *arg* as a parameter.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

Returns

0 on success, otherwise a non-zero error code

8.3.1.10 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. See http://jackit.sourceforge.net/docs/design/design.html#SECTION00411000000000000000 for more information.

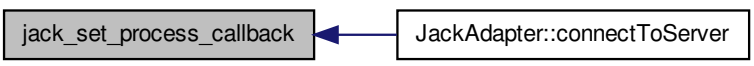
NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

Returns

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

Referenced by JackAdapter::connectToServer().

Here is the caller graph for this function:



8.3.1.11 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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

Returns

0 on success, otherwise a non-zero error code

Referenced by JackAdapter::connectToServer().

Here is the caller graph for this function:

8.3.1.12 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.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

Returns

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

8.3.1.13 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.

All "notification events" are received in a seperated non RT thread, the code in the supplied function does not need to be suitable for real-time execution.

NOTE: this function cannot be called while the client is activated (after jack_activate has been called.)

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

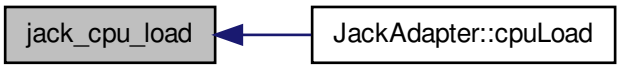
8.4.1.1 float jack_cpu_load (jack_client_t * *client*)

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.

Referenced by JackAdapter::cpuLoad().

Here is the caller graph for this function:



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.

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
- jack_port_type_id_t jack_port_type_id (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
- 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
- size_t jack_port_type_get_buffer_size (jack_client_t *client, const char *port_type) JACK_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

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]*)

Get any aliases known for .

Returns

the number of aliases discovered for the port

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)

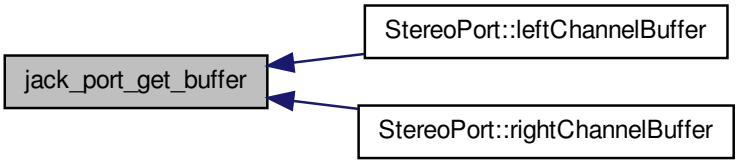
This returns a pointer to the memory area associated with the specified port. For an output port, it will be a memory area that can be written to; for an input port, it will be an area containing the data from the port's connection(s), or zero-filled. if there are multiple inbound connections, the data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

FOR OUTPUT PORTS ONLY : DEPRECATED in Jack 2.0 !! ---------- ---------- You may cache the value returned, but only between calls to your "blocksize" callback. For this reason alone, you should either never cache the return value or ensure you have a "blocksize" callback and be sure to invalidate the cached address from there.

Caching output ports is DEPRECATED in Jack 2.0, due to some new optimization (like "pipelining"). Port buffers have to be retrieved in each callback for proper functionning.

Referenced by StereoPort::leftChannelBuffer(), and StereoPort::rightChannelBuffer().

Here is the caller graph for this function:



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 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.

8.5.1.13 int jack_port_monitoring_input (jack_port_t * *port*)

Returns

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

8.5.1.14 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().

8.5.1.15 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.

8.5.1.16 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.

The *port_name* must be unique among all ports owned by this client. If the name is not unique, the registration will fail.

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, currently only JACK_DEFAULT_AUDIO_TYPE.

Parameters

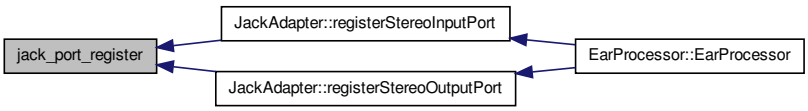
client	pointer to JACK client structure.
port_name	non-empty short name for the new port (not including the leading *"client_name:"*). Must be unique.
port_type	port type name. If longer than jack_port_type_size(), only that many characters are significant.
flags	JackPortFlags bit mask.
buffer_size	must 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 JackAdapter::registerStereoInputPort(), and JackAdapter::registerStereoOutputPort().

Here is the caller graph for this function:



8.5.1.17 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.

8.5.1.18 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()

8.5.1.19 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.

8.5.1.20 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.

8.5.1.21 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().

8.5.1.22 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.

8.5.1.23 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.

8.5.1.24 size_t jack_port_type_get_buffer_size (jack_client_t * *client*, const char * *port_type*)

Returns

the buffersize of a port of type

- port_type.

this function may only be called in a buffer_size callback.

8.5.1.25 jack_port_type_id_t jack_port_type_id (const jack_port_t * *port*)

Returns

the *port* type id.

8.5.1.26 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.

8.5.1.27 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

8.5.1.28 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.

8.5.1.29 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.

8.6 Managing and determining latency

Functions

- void jack_port_set_latency (jack_port_t *, jack_nframes_t) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- void jack_port_get_latency_range (jack_port_t *port, jack_latency_callback_mode_t mode, jack_latency_range_t *range) JACK_WEAK_EXPORT
- void jack_port_set_latency_range (jack_port_t *port, jack_latency_callback_mode_t mode, jack_latency_range_t *range) JACK_WEAK_EXPORT
- int jack_recompute_total_latencies (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_port_get_latency (jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- jack_nframes_t jack_port_get_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_recompute_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT

8.6.1 Detailed Description

The purpose of JACK's latency API is to allow clients to easily answer two questions:

- How long has it been since the data read from a port arrived at the edge of the JACK graph (either via a physical port or being synthesized from scratch)?

- How long will it be before the data written to a port arrives at the edge of a JACK graph?

To help answering these two questions, all JACK ports have two latency values associated with them, both measured in frames:

capture latency: how long since the data read from the buffer of a port arrived at at a port marked with JackPortIsTerminal. The data will have come from the "outside world" if the terminal port is also marked with JackPortIsPhysical, or will have been synthesized by the client that owns the terminal port.

playback latency: how long until the data written to the buffer of port will reach a port marked with JackPortIsTerminal.

Both latencies might potentially have more than one value because there may be multiple pathways to/from a given port and a terminal port. Latency is therefore generally expressed a min/max pair.

In most common setups, the minimum and maximum latency are the same, but this design accomodates more complex routing, and allows applications (and thus users) to detect cases where routing is creating an anomalous situation that may either need fixing or more sophisticated handling by clients that care about latency.

See also jack_set_latency_callback for details on how clients that add latency to the signal path should interact with JACK to ensure that the correct latency figures are used.

8.6.2 Function Documentation

8.6.2.1 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.

Deprecated This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_port_get_latency_range() in any existing use cases.

8.6.2.2 void jack_port_get_latency_range (jack_port_t * *port,* jack_latency_callback_mode_t *mode,* jack_latency_range_t * *range*)

return the latency range defined by *mode* for *port*, in frames.

See Managing and determining latency for the definition of each latency value.

This is normally used in the LatencyCallback. and therefor safe to execute from callbacks.

8.6.2.3 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.

Deprecated This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_port_get_latency_range() in any existing use cases.

8.6.2.4 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.

Deprecated This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by a latency callback that calls jack_port_set_latency_range().

8.6.2.5 void jack_port_set_latency_range (jack_port_t ∗ *port*, jack_latency_callback_mode_t *mode*, jack_latency_range_t ∗ *range*)

set the minimum and maximum latencies defined by *mode* for *port*, in frames.

See Managing and determining latency for the definition of each latency value.

This function should ONLY be used inside a latency callback. The client should determine the current value of the latency using jack_port_get_latency_range() (called using the same mode as *mode*) and then add some number of frames to that reflects latency added by the client.

How much latency a client adds will vary dramatically. For most clients, the answer is zero and there is no reason for them to register a latency callback and thus they should never call this function.

More complex clients that take an input signal, transform it in some way and output the result but not during the same process() callback will generally know a single constant value to add to the value returned by jack_port_get_latency_range().

Such clients would register a latency callback (see jack_set_latency_callback) and must know what input ports feed which output ports as part of their internal state. Their latency callback will update the ports' latency values appropriately.

A pseudo-code example will help. The *mode* argument to the latency callback will determine whether playback or capture latency is being set. The callback will use jack_port_set_latency_range() as follows:

```
jack_latency_range_t range;
if (mode == JackPlaybackLatency) {
 foreach input_port in (all self-registered port) {
  jack_port_get_latency_range (port_feeding_input_port, JackPlaybackLatency, &
     range);
  range.min += min_delay_added_as_signal_flows_from port_feeding to input_port
     ;
  range.max += max_delay_added_as_signal_flows_from port_feeding to input_port
     ;
  jack_port_set_latency_range (input_port, JackPlaybackLatency, &range);
 }
} else if (mode == JackCaptureLatency) {
 foreach output_port in (all self-registered port) {
  jack_port_get_latency_range (port_fed_by_output_port, JackCaptureLatency, &
     range);
  range.min += min_delay_added_as_signal_flows_from_output_port_to_fed_by_port
     ;
  range.max += max_delay_added_as_signal_flows_from_output_port_to_fed_by_port
     ;
```

```
    jack_port_set_latency_range (output_port, JackCaptureLatency, &range);
  }
 }
```

In this relatively simple pseudo-code example, it is assumed that each input port or output is connected to only 1 output or input port respectively.

If a port is connected to more than 1 other port, then the range.min and range.max values passed to jack_port_set_latency_range() should reflect the minimum and maximum values across all connected ports.

See the description of jack_set_latency_callback for more information.

8.6.2.6 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.

8.6.2.7 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.

Deprecated This method will be removed in the next major release of JACK. It should not be used in new code, and should be replaced by jack_recompute_total_latencies() in any existing use cases.

8.7 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

8.7.1 Function Documentation

8.7.1.1 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()

8.7.1.2 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*.

8.7.1.3 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()

8.8 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

8.8.1 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.

8.8.2 Function Documentation

8.8.2.1 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.

8.8.2.2 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.

8.8.2.3 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

8.8.2.4 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.

8.8.2.5 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).

8.8.2.6 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.

8.9 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

8.9.1 Function Documentation

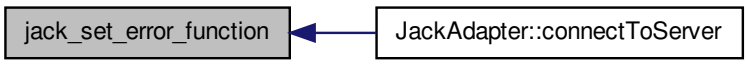
8.9.1.1 void jack_set_error_function (void(∗)(const char ∗) *func*)

Set the jack_error_callback for error message display. Set it to NULL to restore default_jack_error_callback function.

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

Referenced by JackAdapter::connectToServer().

Here is the caller graph for this function:



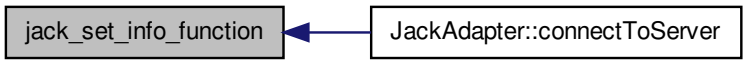
8.9.1.2 void jack_set_info_function (void(∗)(const char ∗) *func*)

Set the jack_info_callback for info message display. Set it to NULL to restore default_jack_info_callback function.

The JACK library provides two built-in callbacks for this purpose: default_jack_info_callback() and silent_jack_info_callback().

Referenced by JackAdapter::connectToServer().

Here is the caller graph for this function:



8.9.2 Variable Documentation

8.9.2.1 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).

8.9.2.2 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).

8.10 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

8.10.1 Function Documentation

8.10.1.1 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).

8.10.1.2 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.

8.10.1.3 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).

8.10.1.4 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.

8.10.1.5 jack_nframes_t jack_midi_get_event_count (void * *port_buffer*)

Get number of events in a port buffer.

Parameters

port_buffer	Port buffer from which to retrieve event.

Returns

number of events inside *port_buffer*

8.10.1.6 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*.

8.10.1.7 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.

8.11 Creating and managing client threads

Typedefs

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

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 (jack_native_thread_t thread, int priority) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_create_thread (jack_client_t ∗client, jack_native_thread_t ∗thread, int priority, int realtime, void ∗(∗start_routine)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_drop_real_time_scheduling (jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_stop_thread (jack_client_t ∗client, jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_kill_thread (jack_client_t ∗client, jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- void jack_set_thread_creator (jack_thread_creator_t creator) JACK_OPTIONAL_WEAK_EXPORT

8.11.1 Typedef Documentation

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

Definition at line 127 of file thread.h.

8.11.2 Function Documentation

8.11.2.1 int jack_acquire_real_time_scheduling (jack_native_thread_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.

8.11.2.2 int jack_client_create_thread (jack_client_t * *client,* jack_native_thread_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.

8.11.2.3 int jack_client_kill_thread (jack_client_t * *client,* jack_native_thread_t *thread*)

Cancel the thread then waits for the thread handler to terminate.

Parameters

thread	POSIX thread ID.

Returns

0, if successful; otherwise an error number.

8.11.2.4 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.

8.11.2.5 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.

8.11.2.6 int jack_client_stop_thread (jack_client_t * *client,* jack_native_thread_t *thread*)

Stop the thread, waiting for the thread handler to terminate.

Parameters

thread	POSIX thread ID.

Returns

0, if successful; otherwise an error number.

8.11.2.7 int jack_drop_real_time_scheduling (jack_native_thread_t *thread*)

Drop realtime scheduling for a thread.

Parameters

thread	POSIX thread ID.

Returns

0, if successful; otherwise an error number.

8.11.2.8 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.

Set it to NULL to restore thread creation function.

Parameters

creator	a function that creates a new thread

8.12 Transport and Timebase control

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_EXPORT
- void jack_set_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_EXPORT

8.12.1 Function Documentation

8.12.1.1 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

8.12.1.2 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.

8.12.1.3 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.

8.12.1.4 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.

8.12.1.5 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.

8.12.1.6 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.

8.12.1.7 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.

8.12.1.8 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.

8.12.1.9 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.

8.12.1.10 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.

8.12.1.11 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.

8.12.1.12 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.

8.13 managing support for newer/older versions of JACK

One challenge faced by developers is that of taking advantage of new features introduced in new versions of [JACK] while still supporting older versions of the system. Normally, if an application uses a new feature in a library/API, it is unable to run on earlier versions of the library/API that do not support that feature. Such applications would either fail to launch or crash when an attempt to use the feature was made. This problem cane be solved using weakly-linked symbols.

When a symbol in a framework is defined as weakly linked, the symbol does not have to be present at runtime for a process to continue running. The static linker identifies a weakly linked symbol as such in any code module that references the symbol. The dynamic linker uses this same information at runtime to determine whether a process can continue running. If a weakly linked symbol is not present in the framework, the code module can continue to run as long as it does not reference the symbol. However, if the symbol is present, the code can use it normally.

(adapted from: http://developer.apple.com/library/mac/#documentation/-MacOSX/Conceptual/BPFrameworks/Concepts/WeakLinking.html)

A concrete example will help. Suppose that someone uses a version of a JACK client we'll call "Jill". Jill was linked against a version of JACK that contains a newer part of the API (say, jack_set_latency_callback()) and would like to use it if it is available.

When Jill is run on a system that has a suitably "new" version of JACK, this function will be available entirely normally. But if Jill is run on a system with an old version of JACK, the function isn't available.

With normal symbol linkage, this would create a startup error whenever someone tries to run Jill with the "old" version of JACK. However, functions added to JACK after version 0.116.2 are all declared to have "weak" linkage which means that their abscence doesn't cause an error during program startup. Instead, Jill can test whether or not the symbol jack_set_latency_callback is null or not. If its null, it means that the JACK installed on this machine is too old to support this function. If its not null, then Jill can use it just like any other function in the API. For example:

```
if (jack_set_latency_callback) {
      jack_set_latency_callback (jill_client, jill_latency_callback, arg);
}
```

However, there are clients that may want to use this approach to parts of the the JACK API that predate 0.116.2. For example, they might want to see if even really old basic parts of the API like jack_client_open() exist at runtime.

Such clients should include <jack/weakjack.h> before any other JACK header. This will make the **entire** JACK API be subject to weak linkage, so that any and all functions can be checked for existence at runtime. It is important to understand that very few clients need to do this - if you use this feature you should have a clear reason to do so.

Chapter 9

Namespace Documentation

9.1 FftwAdapter Namespace Reference

Functions

- void blit (fftw_complex *fftw_complexIn, jack_default_audio_sample_t *jack_default_audio_sample_tsOut, int n)
- void blit (jack_default_audio_sample_t *jack_default_audio_sample_tsIn, fftw_complex *fftw_complexOut, int n)
- void blit (jack_default_audio_sample_t *jack_default_audio_sample_tsIn, jack_default_audio_sample_t *jack_default_audio_sample_tsOut, int n)
- void blit (fftw_complex *fftw_complexIn, fftw_complex *fftw_complexOut, int n)
- void performFFT (fftw_complex *input, fftw_complex *result, int n)
- void performInverseFFT (fftw_complex *input, fftw_complex *result, int n)

9.1.1 Function Documentation

9.1.1.1 void FftwAdapter::blit (fftw_complex * *fftw_complexIn*, jack_default_audio_sample_t * *jack_default_audio_sample_tsOut*, int *n*)

fftw works with arrays of fftw_complex numbers. In order to use fftw, you have to convert between JACK samples, which are mere real numbers and fftw_complex arrays. This method copies over from fftw_complex to JACK sample values.

Parameters

fftw_complexIn	Input array of fftw_complex numbers.
jack_default_audio_sample_tsOut	Output array of JACK samples.
n	Number of samples.

Definition at line 24 of file fftwadapter.cpp.

9.1.1.2 void FftwAdapter::blit (jack_default_audio_sample_t * *jack_default_audio_sample_tsIn,* fftw_complex * *fftw_complexOut,* int *n*)

fftw works with arrays of fftw_complex numbers. In order to use fftw, you have to convert between JACK samples, which are mere real numbers and fftw_complex arrays. This method copies over from JACK sample to fftw_complex values.

Parameters

jack_default_audio_sample_tsIn	Input array of fftw_complex numbers.
fftw_complexOut	Output array of JACK samples.
n	Number of samples.

Definition at line 32 of file fftwadapter.cpp.

9.1.1.3 void FftwAdapter::blit (jack_default_audio_sample_t * *jack_default_audio_sample_tsIn,* jack_default_audio_sample_t * *jack_default_audio_sample_tsOut,* int *n*)

Definition at line 41 of file fftwadapter.cpp.

9.1.1.4 void FftwAdapter::blit (fftw_complex * *fftw_complexIn,* fftw_complex * *fftw_complexOut,* int *n*)

Definition at line 49 of file fftwadapter.cpp.

9.1.1.5 void FftwAdapter::performFFT (fftw_complex * *input,* fftw_complex * *result,* int *n*)

Performs the fft.

Parameters

fftw_complex	Input array of comples numbers.
result	Output array of fftw_complex numbers.
n	Number of samples.

Definition at line 58 of file fftwadapter.cpp.

References FFTW_ESTIMATE, and FFTW_FORWARD.

9.1.1.6 void FftwAdapter::performInverseFFT (fftw_complex * *input*, fftw_complex * *result*, int *n*)

Performs the inverse fft.

Parameters

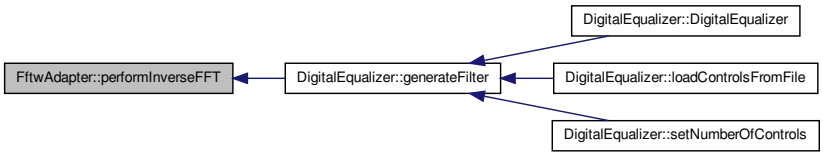
fftw_complex	Input array of comples numbers.
result	Output array of fftw_complex numbers.
n	Number of samples.

Definition at line 65 of file fftwadapter.cpp.

References FFTW_BACKWARD, and FFTW_ESTIMATE.

Referenced by DigitalEqualizer::generateFilter().

Here is the caller graph for this function:



9.2 Ui Namespace Reference

Chapter 10

Data Structure Documentation

10.1 _jack_midi_event Struct Reference

`#include <midiport.h>`

Data Fields

- jack_nframes_t time
- size_t size
- jack_midi_data_t * buffer

10.1.1 Detailed Description

A Jack MIDI event.

Definition at line 38 of file midiport.h.

10.1.2 Field Documentation

10.1.2.1 jack_midi_data_t* _jack_midi_event::buffer

Raw MIDI data

Definition at line 42 of file midiport.h.

10.1.2.2 size_t _jack_midi_event::size

Number of bytes of data in *buffer*

Definition at line 41 of file midiport.h.

10.1.2.3 jack_nframes_t _jack_midi_event::time

Sample index at which event is valid

Definition at line 40 of file midiport.h.

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

- 3rdparty/jack/include/jack/midiport.h

10.2 _JSList Struct Reference

`#include <jslist.h>`

Data Fields

- void * data
- JSList * next

10.2.1 Detailed Description

Definition at line 38 of file jslist.h.

10.2.2 Field Documentation

10.2.2.1 void* _JSList::data

Definition at line 40 of file jslist.h.

10.2.2.2 JSList* _JSList::next

Definition at line 41 of file jslist.h.

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

- 3rdparty/jack/include/jack/jslist.h

10.3 DigitalEqualizer Class Reference

Modifies the frequency spectrum of the sampled audio signal.

`#include <digitalequalizer.h>`

Public Member Functions

- DigitalEqualizer ()
- ~DigitalEqualizer ()
- void setNumberOfControls (int n)
- int numberOfControls ()
- void acquireControls ()
- double * controls ()
- void releaseControls ()
- bool saveControlsToFile (QString fileName)
- bool loadControlsFromFile (QString fileName)
- void generateFilter ()
- void process (fftw_complex *sampleBuffer, fftw_complex *result, int samples)

10.3.1 Detailed Description

Modifies the frequency spectrum of the sampled audio signal.

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Date

09.2011

Definition at line 37 of file digitalequalizer.h.

10.3.2 Constructor & Destructor Documentation

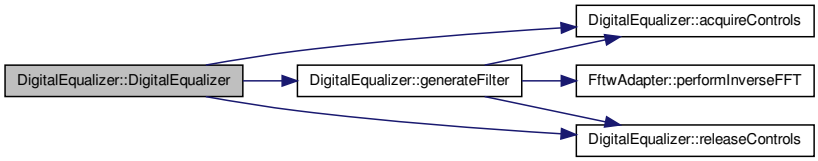
10.3.2.1 DigitalEqualizer::DigitalEqualizer ( )

Constructs a new digital equalizer.

Definition at line 27 of file digitalequalizer.cpp.

References acquireControls(), generateFilter(), and releaseControls().

Here is the call graph for this function:



10.3.2.2 DigitalEqualizer::~DigitalEqualizer ( )

Destructor.

Definition at line 42 of file digitalequalizer.cpp.

10.3.3 Member Function Documentation

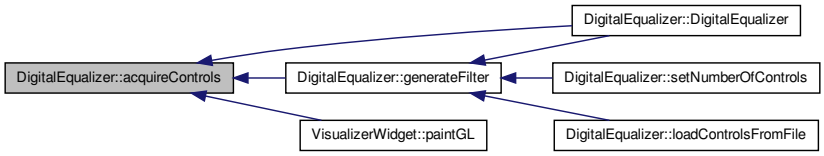
10.3.3.1 void DigitalEqualizer::acquireControls ( )

Grants exclusive access access to equalizer controls. Blocks in case anyone else has been accessing these.

Definition at line 63 of file digitalequalizer.cpp.

Referenced by DigitalEqualizer(), generateFilter(), and VisualizerWidget::paintGL().

Here is the caller graph for this function:



10.3.3.2 double * DigitalEqualizer::controls ( )

Pointer to the controls array. WARNING: Before accessing controls, make sure you have acquired exclusive access to these by calling acquireControls. Also, you have to call releaseControls after you are done.

Definition at line 67 of file digitalequalizer.cpp.

Referenced by VisualizerWidget::paintGL().

Here is the caller graph for this function:

10.3.3.3 void DigitalEqualizer::generateFilter ( )

Updates the filter from the given set of equalizer control values.

Parameters

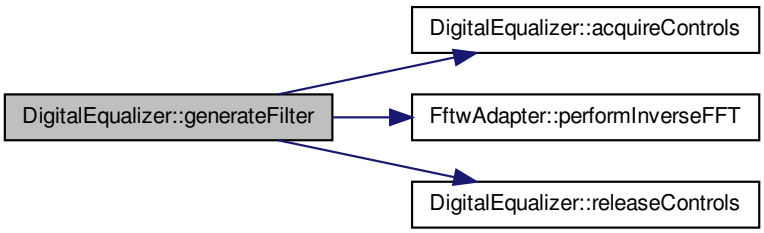
values	Equalizer control values.

Definition at line 98 of file digitalequalizer.cpp.

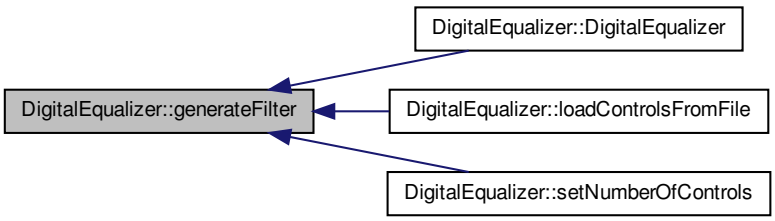
References acquireControls(), FftwAdapter::performInverseFFT(), and releaseControls().

Referenced by DigitalEqualizer(), loadControlsFromFile(), and setNumberOfControls().

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.3.4 bool DigitalEqualizer::loadControlsFromFile (QString *fileName*)

Attempts to restore control values from a file.

Parameters

fileName	File name of the file form which shall be loaded.

Returns

true on success, otherwise false.

Definition at line 86 of file digitalequalizer.cpp.

References generateFilter().

Here is the call graph for this function:

10.3.3.5 int DigitalEqualizer::numberOfControls ( )

Returns the number of controls in linear frequency steps.

Definition at line 58 of file digitalequalizer.cpp.

10.3.3.6 void DigitalEqualizer::process (fftw_complex * *sampleBuffer,* fftw_complex * *result,* int *samples*)

Processes a given number of samples. In order to function properly, this method expects a consecutive stream of samples. Do not call this method more than once on a given set of samples, since this will lead to erroneous results.

Parameters

sample-Buffer	Input sample buffer.
result	Result sample buffer.
samples	Number of samples.

Definition at line 199 of file digitalequalizer.cpp.

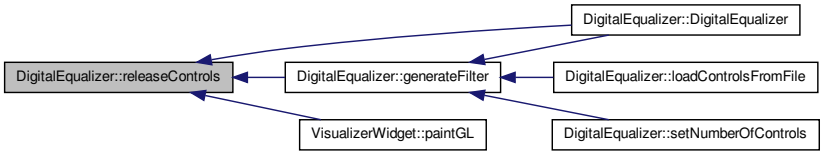
10.3.3.7 void DigitalEqualizer::releaseControls ( )

Releases exclusive access to equalizer controls.

Definition at line 71 of file digitalequalizer.cpp.

Referenced by DigitalEqualizer(), generateFilter(), and VisualizerWidget::paintGL().

Here is the caller graph for this function:



10.3.3.8 bool DigitalEqualizer::saveControlsToFile (QString *fileName*)

Attempts to write control values into a file.

Parameters

fileName	File name of the file that shall be saved.

Returns

true on success, otherwise false.

Definition at line 75 of file digitalequalizer.cpp.

10.3.3.9 void DigitalEqualizer::setNumberOfControls (int *n*)

Sets the number of controls in linear frequency steps.

Definition at line 47 of file digitalequalizer.cpp.

References generateFilter().

Here is the call graph for this function:

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

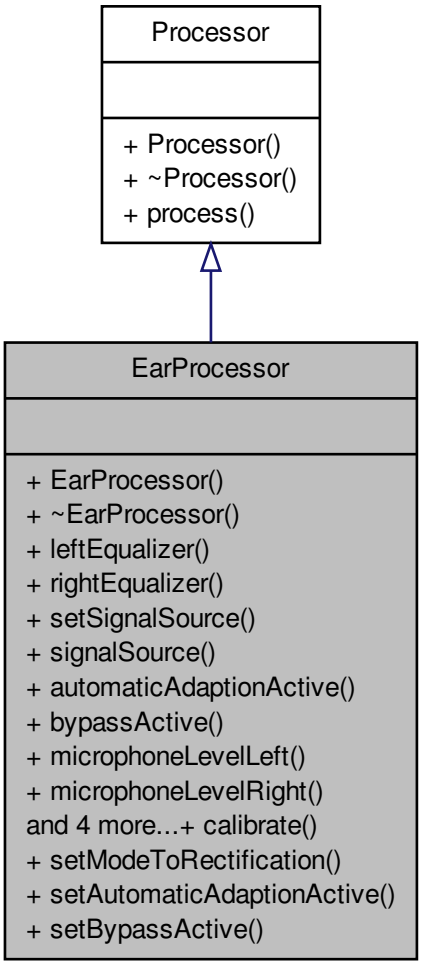
- libear/include/digitalequalizer.h
- libear/src/digitalequalizer.cpp

10.4 EarProcessor Class Reference

EarProcessor performs the central task of audio processing.

```
#include <earprocessor.h>
```

Inheritance diagram for EarProcessor:



Data Structures

- struct **Calibration**

Public Types

- enum OperationMode { CalibratingLatency, ProcessingAudio }

- enum SignalSource { ExternalSource, WhiteNoise, PinkNoise }
- enum Channel { LeftChannel = 0, RightChannel = 1 }

Public Slots

- void calibrate ()
- void setModeToRectification ()
- void setAutomaticAdaptionActive (bool on)
- void setBypassActive (bool on)

Signals

- void calibrationFinished ()

Public Member Functions

- EarProcessor ()
- ~EarProcessor ()
- DigitalEqualizer * leftEqualizer ()
- DigitalEqualizer * rightEqualizer ()
- void setSignalSource (SignalSource signalSource)
- SignalSource signalSource ()
- bool automaticAdaptionActive ()
- bool bypassActive ()
- int microphoneLevelLeft ()
- int microphoneLevelRight ()
- int signalSourceLevelLeft ()
- int signalSourceLevelRight ()
- void process (int samples)
- int leftLatency ()
- int rightLatency ()

10.4.1 Detailed Description

EarProcessor performs the central task of audio processing.

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Date

09.2011

The EAR processor compares the incoming delayed reference signal with the recorded signal usually obtained by the microphone. In order to compensate the differences in the the frequency spectrum it uses two independent equalizers.

Definition at line 50 of file earprocessor.h.

10.4.2 Member Enumeration Documentation

10.4.2.1 enum EarProcessor::Channel

Signal channels.

Enumerator:

LeftChannel

RightChannel

Definition at line 68 of file earprocessor.h.

10.4.2.2 enum EarProcessor::OperationMode

Operating modes.

Enumerator:

CalibratingLatency

ProcessingAudio

Definition at line 55 of file earprocessor.h.

10.4.2.3 enum EarProcessor::SignalSource

Different kinds of signal sources.

Enumerator:

ExternalSource

WhiteNoise

PinkNoise

Definition at line 61 of file earprocessor.h.

10.4.3 Constructor & Destructor Documentation

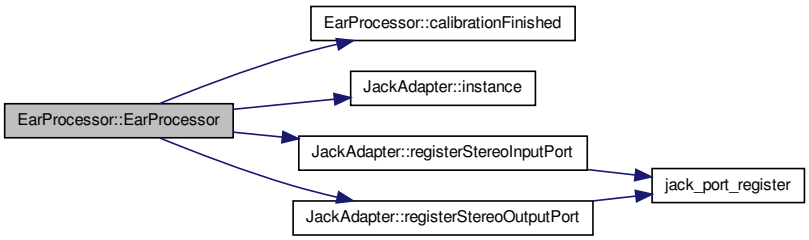
10.4.3.1 EarProcessor::EarProcessor ( )

Constructs a new EarProcessor.

Definition at line 29 of file earprocessor.cpp.

References calibrationFinished(), ExternalSource, JackAdapter::instance(), ProcessingAudio, JackAdapter::registerStereoInputPort(), and JackAdapter::registerStereoOutputPort().

Here is the call graph for this function:



10.4.3.2 EarProcessor::~EarProcessor ( )

Destructor.

Definition at line 65 of file earprocessor.cpp.

10.4.4 Member Function Documentation

10.4.4.1 bool EarProcessor::automaticAdaptionActive ( )

Returns true, when the automatic adaption is active.

Definition at line 89 of file earprocessor.cpp.

10.4.4.2 bool EarProcessor::bypassActive ( )

Returns true, when the signal is being bypassed.

Definition at line 94 of file earprocessor.cpp.

10.4.4.3 void EarProcessor::calibrate ( ) [slot]

Resets the calibration.

Definition at line 471 of file earprocessor.cpp.

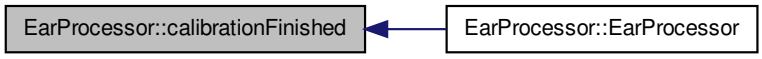
References CalibratingLatency.

10.4.4.4 void EarProcessor::calibrationFinished ( ) [signal]

This signal is emitted when the calibration finished.

Referenced by EarProcessor().

Here is the caller graph for this function:



10.4.4.5 DigitalEqualizer * EarProcessor::leftEqualizer ( )

Returns a pointer to the equalizer object for the left channel.

Definition at line 71 of file earprocessor.cpp.

Referenced by VisualizerWidget::paintGL().

Here is the caller graph for this function:

10.4.4.6 int EarProcessor::leftLatency ( ) [inline]

Provides the latency for the left channel.

Returns

Latency measured in number of samples.

See also

EarProcessor::processCalibration(int samples)

Definition at line 120 of file earprocessor.h.

10.4.4.7 int EarProcessor::microphoneLevelLeft ( )

Returns the average microphone amplitude for the left channel.

Definition at line 176 of file earprocessor.cpp.

References LeftChannel.

10.4.4.8 int EarProcessor::microphoneLevelRight ( )

Returns the average microphone amplitude for the right channel.

Definition at line 181 of file earprocessor.cpp.

References RightChannel.

10.4.4.9 void EarProcessor::process (int *samples*) [virtual]

Reimplemented from process. This will be called whenever there are new samples available for processing. Attention: This method is time critical.

Parameters

samples	Number of samples that are available.

Implements Processor.

Definition at line 196 of file earprocessor.cpp.

References CalibratingLatency, and ProcessingAudio.

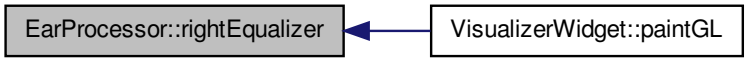
10.4.4.10 DigitalEqualizer * EarProcessor::rightEqualizer ( )

Returns a pointer to the equalizer object for the right channel.

Definition at line 75 of file earprocessor.cpp.

Referenced by VisualizerWidget::paintGL().

Here is the caller graph for this function:



10.4.4.11 int EarProcessor::rightLatency ( ) [inline]

Provides the latency for the right channel.

Returns

Latency measured in number of samples.

See also

EarProcessor::processCalibration(int samples)

Definition at line 127 of file earprocessor.h.

10.4.4.12 void EarProcessor::setAutomaticAdaptionActive (bool *on*) [slot]

Activates/deactivates automatic adaption.

Definition at line 489 of file earprocessor.cpp.

10.4.4.13 void EarProcessor::setBypassActive (bool *on*) [slot]

Activates/deactivates bypassing.

Definition at line 494 of file earprocessor.cpp.

10.4.4.14 void EarProcessor::setModeToRectification ( ) [slot]

Set mode to "Rectification".

Definition at line 480 of file earprocessor.cpp.

References ProcessingAudio.

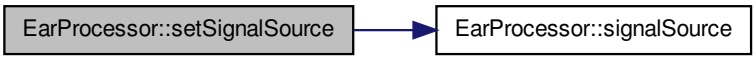
10.4.4.15 void EarProcessor::setSignalSource (SignalSource *signalSource*)

Immediately sets the signal source to use.

Definition at line 79 of file earprocessor.cpp.

References signalSource().

Here is the call graph for this function:



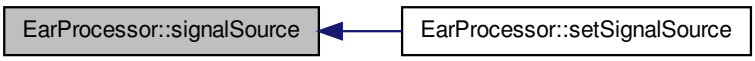
10.4.4.16 EarProcessor::SignalSource EarProcessor::signalSource ( )

Returns the the signal source that is used for processing.

Definition at line 84 of file earprocessor.cpp.

Referenced by setSignalSource().

Here is the caller graph for this function:



10.4.4.17 int EarProcessor::signalSourceLevelLeft ( )

Returns the average signal source amplitude for the left channel.

Definition at line 186 of file earprocessor.cpp.

References LeftChannel.

10.4.4.18 int EarProcessor::signalSourceLevelRight ( )

Returns the average signal source amplitude for the right channel.

Definition at line 191 of file earprocessor.cpp.

References RightChannel.

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

- libear/include/earprocessor.h
- libear/src/earprocessor.cpp

10.5 fftw_iodim64_do_not_use_me Struct Reference

```
#include <fftw3.h>
```

Data Fields

- ptrdiff_t n
- ptrdiff_t is
- ptrdiff_t os

10.5.1 Detailed Description

Definition at line 103 of file fftw3.h.

10.5.2 Field Documentation

10.5.2.1 ptrdiff_t fftw_iodim64_do_not_use_me::is

Definition at line 105 of file fftw3.h.

10.5.2.2 ptrdiff_t fftw_iodim64_do_not_use_me::n

Definition at line 104 of file fftw3.h.

10.5.2.3 ptrdiff_t fftw_iodim64_do_not_use_me::os

Definition at line 106 of file fftw3.h.

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

- 3rdparty/fftw/include/fftw3.h

10.6 fftw_iodim_do_not_use_me Struct Reference

```
#include <fftw3.h>
```

Data Fields

- int n
- int is
- int os

10.6.1 Detailed Description

Definition at line 96 of file fftw3.h.

10.6.2 Field Documentation

10.6.2.1 int fftw_iodim_do_not_use_me::is

Definition at line 98 of file fftw3.h.

10.6.2.2 int fftw_iodim_do_not_use_me::n

Definition at line 97 of file fftw3.h.

10.6.2.3 int fftw_iodim_do_not_use_me::os

Definition at line 99 of file fftw3.h.

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

- 3rdparty/fftw/include/fftw3.h

10.7 jack_ringbuffer_data_t Struct Reference

`#include <ringbuffer.h>`

Data Fields

- char * buf
- size_t len

10.7.1 Detailed Description

Definition at line 45 of file ringbuffer.h.

10.7.2 Field Documentation

10.7.2.1 char* jack_ringbuffer_data_t::buf

Definition at line 46 of file ringbuffer.h.

10.7.2.2 size_t jack_ringbuffer_data_t::len

Definition at line 47 of file ringbuffer.h.

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

- 3rdparty/jack/include/jack/ringbuffer.h

10.8 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

10.8.1 Detailed Description

Definition at line 51 of file ringbuffer.h.

10.8.2 Field Documentation

10.8.2.1 char* jack_ringbuffer_t::buf

Definition at line 52 of file ringbuffer.h.

10.8.2.2 int jack_ringbuffer_t::mlocked

Definition at line 57 of file ringbuffer.h.

10.8.2.3 volatile size_t jack_ringbuffer_t::read_ptr

Definition at line 54 of file ringbuffer.h.

10.8.2.4 size_t jack_ringbuffer_t::size

Definition at line 55 of file ringbuffer.h.

10.8.2.5 size_t jack_ringbuffer_t::size_mask

Definition at line 56 of file ringbuffer.h.

10.8.2.6 volatile size_t jack_ringbuffer_t::write_ptr

Definition at line 53 of file ringbuffer.h.

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

- 3rdparty/jack/include/jack/ringbuffer.h

10.9 JackAdapter Class Reference

C++ Wrapper for the JACK Audio Connection Kit client API.

`#include <jackadapter.h>`

Signals

- void error (const QString &errorMessage)

Public Member Functions

- bool connectToServer (QString name)
- void registerStereoInputPort (QString label)
- void registerStereoOutputPort (QString label)
- void setProcessor (Processor *processor)
- void startAudioProcessing ()
- void stopAudioProcessing ()
- int sampleRate ()
- int bufferSize ()
- float cpuLoad ()
- StereoPort stereoInputPort (QString label)
- StereoPort stereoOutputPort (QString label)

Static Public Member Functions

- static JackAdapter * instance ()

10.9.1 Detailed Description

C++ Wrapper for the JACK Audio Connection Kit client API.

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Date

09.2011

This class wraps a singleton around the C API of JACK in order to provide some additional features. Since it derives from QObject it can be integrated into Qt's signals and slots system. Also, it includes a few routines to simplify the integration of fftw.

Definition at line 114 of file jackadapter.h.

10.9.2 Member Function Documentation

10.9.2.1 int JackAdapter::bufferSize ( )

Returns the current sample buffer size.

Returns

Buffer size in samples.

Definition at line 75 of file jackadapter.cpp.

10.9.2.2 bool JackAdapter::connectToServer (QString *name*)

This method attempts to connect to the audio server.

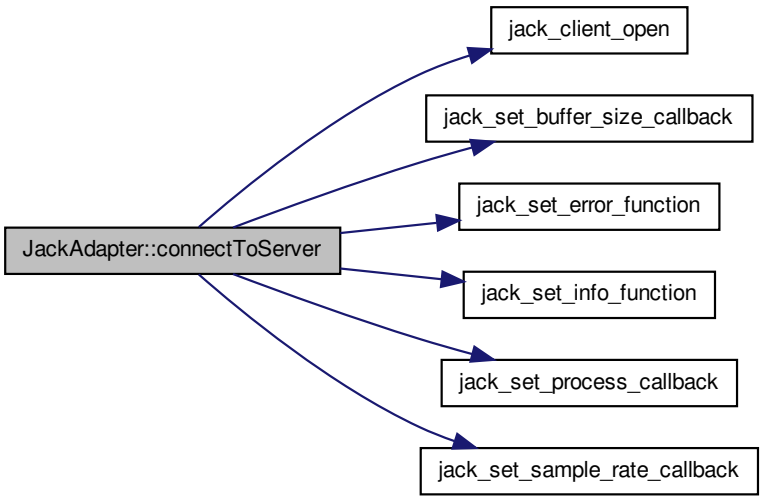
Parameters

name	Name that will be used to register this application as a client.

Definition at line 47 of file jackadapter.cpp.

References jack_client_open(), jack_set_buffer_size_callback(), jack_set_error_function(), jack_set_info_function(), jack_set_process_callback(), jack_set_sample_rate_callback(), and JackNullOption.

Here is the call graph for this function:



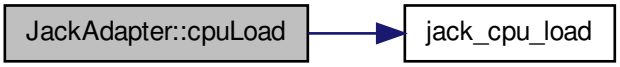
10.9.2.3 float JackAdapter::cpuLoad ( )

Returns the current CPU load in percent.

Definition at line 79 of file jackadapter.cpp.

References jack_cpu_load().

Here is the call graph for this function:



10.9.2.4 void JackAdapter::error (const QString & *errorMessage*) [signal]

This signal will be emitted when an error occurs.

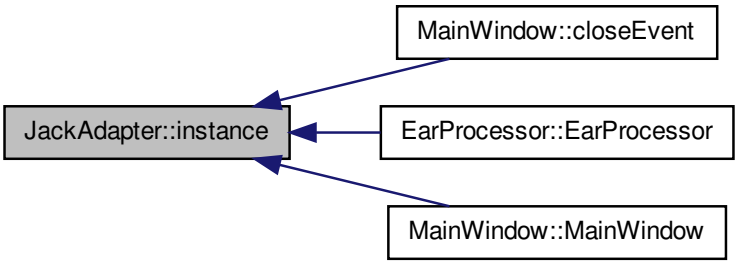
10.9.2.5 JackAdapter * JackAdapter::instance ( ) [static]

Returns the instance for this singleton.

Definition at line 43 of file jackadapter.cpp.

Referenced by MainWindow::closeEvent(), EarProcessor::EarProcessor(), and MainWindow::MainWindow().

Here is the caller graph for this function:



10.9.2.6 void JackAdapter::registerStereoInputPort (QString *label*)

Tries to register a new stereo input port.

Parameters

label	Name that will be used to register this port.

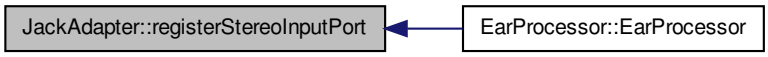
Definition at line 91 of file jackadapter.cpp.

References JACK_DEFAULT_AUDIO_TYPE, and jack_port_register().

Referenced by EarProcessor::EarProcessor().

Here is the call graph for this function:

Here is the caller graph for this function:



10.9.2.7 void JackAdapter::registerStereoOutputPort (QString *label*)

Tries to register a new stereo output port.

Parameters

label	Name that will be used to register this port.

Definition at line 102 of file jackadapter.cpp.

References JACK_DEFAULT_AUDIO_TYPE, and jack_port_register().

Referenced by EarProcessor::EarProcessor().

Here is the call graph for this function:

Here is the caller graph for this function:

10.9.2.8 int JackAdapter::sampleRate ( )

Returns the current sample rate.

Returns

Sample rate in Hz.

Definition at line 71 of file jackadapter.cpp.

10.9.2.9 void JackAdapter::setProcessor (Processor * *processor*)

Assigns a processor that will handle audio processing.

Parameters

processor	The processor that will handle audio processing.

Definition at line 113 of file jackadapter.cpp.

Referenced by MainWindow::MainWindow().

Here is the caller graph for this function:

10.9.2.10 void JackAdapter::startAudioProcessing ( )

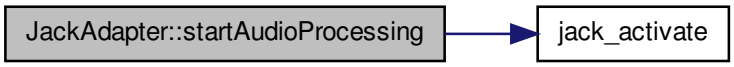
Activates audio processing.

Definition at line 61 of file jackadapter.cpp.

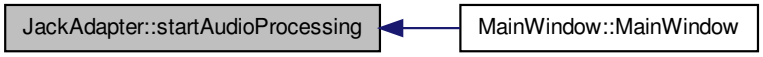
References jack_activate().

Referenced by MainWindow::MainWindow().

Here is the call graph for this function:



Here is the caller graph for this function:



10.9.2.11 StereoPort JackAdapter::stereoInputPort (QString *label*)

Returns the input stereo port associated with this label.

Parameters

label	Name of the stereo port.

Returns

Stereo port that has been requested.

Definition at line 83 of file jackadapter.cpp.

10.9.2.12 StereoPort JackAdapter::stereoOutputPort (QString *label*)

Returns the output stereo port associated with this label.

Parameters

label	Name of the stereo port.

Returns

Stereo port that has been requested.

Definition at line 87 of file jackadapter.cpp.

10.9.2.13 void JackAdapter::stopAudioProcessing ( )

Deactivates audio processing and disconnects from the audio server.

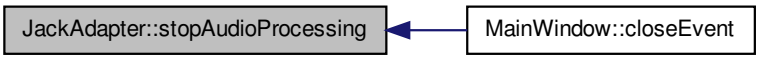
Definition at line 67 of file jackadapter.cpp.

References jack_deactivate().

Referenced by MainWindow::closeEvent().

Here is the call graph for this function:

Here is the caller graph for this function:



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

- libear/include/jackadapter.h
- libear/src/jackadapter.cpp

10.10 jackctl_parameter_value Union Reference

Type for parameter value.

`#include <control.h>`

Data Fields

- uint32_t ui

  member used for JackParamUInt
- int32_t i

  member used for JackParamInt
- char c

  member used for JackParamChar
- char str [JACK_PARAM_STRING_MAX+1]

  member used for JackParamString
- bool b

  member used for JackParamBool

10.10.1 Detailed Description

Type for parameter value.

Definition at line 56 of file control.h.

10.10.2 Field Documentation

10.10.2.1 bool jackctl_parameter_value::b

member used for JackParamBool

Definition at line 62 of file control.h.

10.10.2.2 char jackctl_parameter_value::c

member used for JackParamChar

Definition at line 60 of file control.h.

10.10.2.3 int32_t jackctl_parameter_value::i

member used for JackParamInt

Definition at line 59 of file control.h.

10.10.2.4 char jackctl_parameter_value::str[JACK_PARAM_STRING_MAX+1]

member used for JackParamString

Definition at line 61 of file control.h.

10.10.2.5 uint32_t jackctl_parameter_value::ui

member used for JackParamUInt

Definition at line 58 of file control.h.

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

- 3rdparty/jack/include/jack/control.h

10.11 JNoise Class Reference

`#include <jnoise.h>`

Public Member Functions

- JNoise ()
- void process (int n, float *whiteNoiseBufferLeft, float *whiteNoiseBufferRight, float *pinkNoiseBufferLeft, float *pinkNoiseBufferRight)

10.11.1 Detailed Description

Definition at line 31 of file jnoise.h.

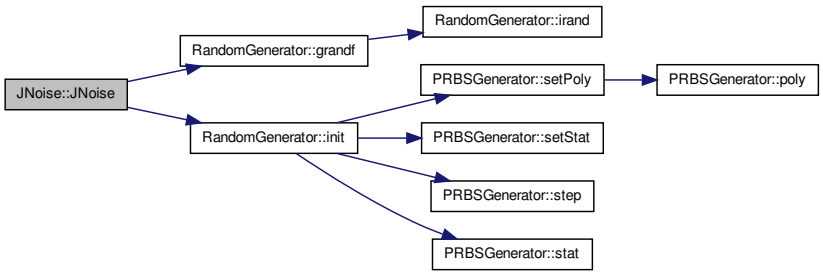
10.11.2 Constructor & Destructor Documentation

10.11.2.1 JNoise::JNoise ( )

Definition at line 22 of file jnoise.cpp.

References RandomGenerator::grandf(), RandomGenerator::init(), and LRAND.

Here is the call graph for this function:



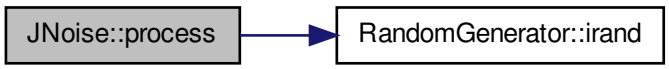
10.11.3 Member Function Documentation

10.11.3.1 void JNoise::process (int *n*, float * *whiteNoiseBufferLeft*, float * *whiteNoiseBufferRight*, float * *pinkNoiseBufferLeft*, float * *pinkNoiseBufferRight*)

Definition at line 40 of file jnoise.cpp.

References RandomGenerator::irand(), and MRAND.

Here is the call graph for this function:



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

- libear/include/jnoise/jnoise.h
- libear/src/jnoise/jnoise.cpp

10.12 MainWindow Class Reference

```
#include <mainwindow.h>
```

Public Member Functions

- MainWindow (QWidget ∗parent=0)
- ∼MainWindow ()

Protected Member Functions

- void closeEvent (QCloseEvent ∗closeEvent)

10.12.1 Detailed Description

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Date

09.2011

Main window for the whole application.

Definition at line 46 of file mainwindow.h.

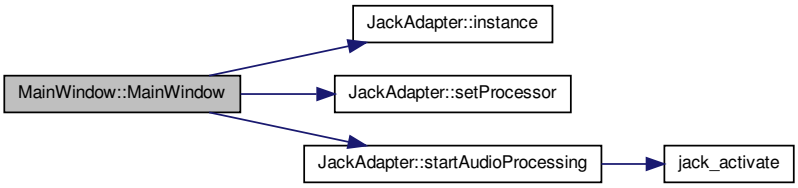
10.12.2 Constructor & Destructor Documentation

10.12.2.1 MainWindow::MainWindow (QWidget ∗ *parent* = 0) [explicit]

Definition at line 33 of file mainwindow.cpp.

References JackAdapter::instance(), MUSIC_COMBO_TEXT, PINK_NOISE_COMBO_TEXT, JackAdapter::setProcessor(), JackAdapter::startAudioProcessing(), and WHITE_NOISE_COMBO_TEXT.

Here is the call graph for this function:



10.12.2.2 MainWindow::~MainWindow ( )

Definition at line 75 of file mainwindow.cpp.

10.12.3 Member Function Documentation

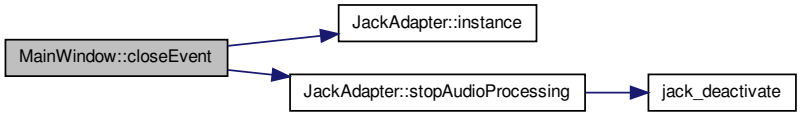
10.12.3.1 void MainWindow::closeEvent (QCloseEvent * *closeEvent*) [protected]

Reimplemented from QWidget.

Definition at line 79 of file mainwindow.cpp.

References JackAdapter::instance(), and JackAdapter::stopAudioProcessing().

Here is the call graph for this function:



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

- gui/include/mainwindow.h
- gui/src/mainwindow.cpp

10.13 PRBSGenerator Class Reference

`#include <prbsgenerator.h>`

Public Types

- enum { G7 = 0x00000041, G8 = 0x0000008E, G15 = 0x00004001, G16 = 0x00008016, G23 = 0x00400010, G24 = 0x0080000D, G31 = 0x40000004, G32 = 0x80000057 }

Public Member Functions

- PRBSGenerator (void)
- void setPoly (uint32_t poly)

- void setStat (uint32_t stat)
- void sync_forw (uint32_t bits)
- void sync_back (uint32_t bits)
- int step (void)
- void crc_in (int b)
- int crc_out (void)
- uint32_t stat (void) const
- uint32_t poly (void) const
- uint32_t mask (void) const
- uint32_t hbit (void) const
- int degr (void) const
- ~PRBSGenerator (void)

10.13.1 Detailed Description

Definition at line 102 of file prbsgenerator.h.

10.13.2 Member Enumeration Documentation

10.13.2.1 anonymous enum

Enumerator:

G7

G8

G15

G16

G23

G24

G31

G32

Definition at line 106 of file prbsgenerator.h.

10.13.3 Constructor & Destructor Documentation

10.13.3.1 PRBSGenerator::PRBSGenerator (void) [inline]

Definition at line 149 of file prbsgenerator.h.

10.13.3.2 PRBSGenerator::~PRBSGenerator (void) [inline]

Definition at line 155 of file prbsgenerator.h.

10.13.4 Member Function Documentation

10.13.4.1 void PRBSGenerator::crc_in (int *b*) [inline]

Definition at line 230 of file prbsgenerator.h.

10.13.4.2 int PRBSGenerator::crc_out (void) [inline]

Definition at line 242 of file prbsgenerator.h.

10.13.4.3 int PRBSGenerator::degr (void) const [inline]

Definition at line 262 of file prbsgenerator.h.

10.13.4.4 uint32_t PRBSGenerator::hbit (void) const [inline]

Definition at line 260 of file prbsgenerator.h.

10.13.4.5 uint32_t PRBSGenerator::mask (void) const [inline]

Definition at line 258 of file prbsgenerator.h.

10.13.4.6 uint32_t PRBSGenerator::poly (void) const [inline]

Definition at line 256 of file prbsgenerator.h.

Referenced by setPoly().

Here is the caller graph for this function:

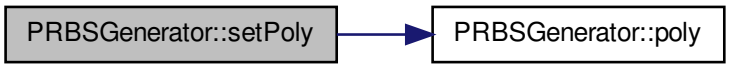
10.13.4.7 void PRBSGenerator::setPoly (uint32_t *poly*) [inline]

Definition at line 160 of file prbsgenerator.h.

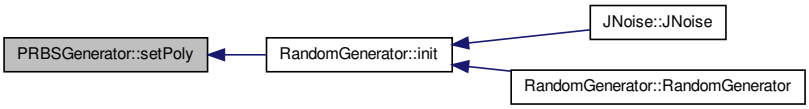
References poly().

Referenced by RandomGenerator::init().

Here is the call graph for this function:



Here is the caller graph for this function:

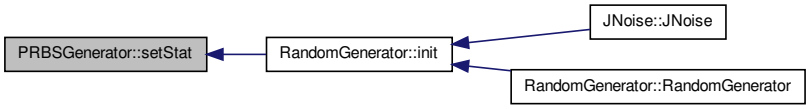


10.13.4.8 void PRBSGenerator::setStat (uint32_t *stat*) [inline]

Definition at line 178 of file prbsgenerator.h.

Referenced by RandomGenerator::init().

Here is the caller graph for this function:

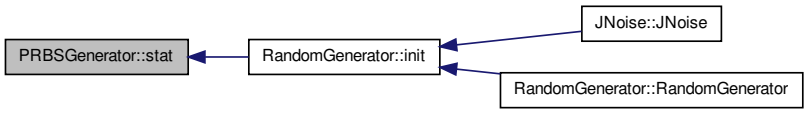


10.13.4.9 uint32_t PRBSGenerator::stat (void) const [inline]

Definition at line 254 of file prbsgenerator.h.

Referenced by RandomGenerator::init().

Here is the caller graph for this function:

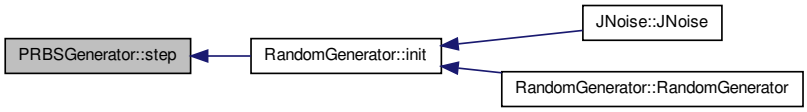


10.13.4.10 int PRBSGenerator::step (void) [inline]

Definition at line 188 of file prbsgenerator.h.

Referenced by RandomGenerator::init().

Here is the caller graph for this function:



10.13.4.11 void PRBSGenerator::sync_back (uint32_t *bits*) [inline]

Definition at line 215 of file prbsgenerator.h.

10.13.4.12 void PRBSGenerator::sync_forw (uint32_t *bits*) [inline]

Definition at line 202 of file prbsgenerator.h.

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

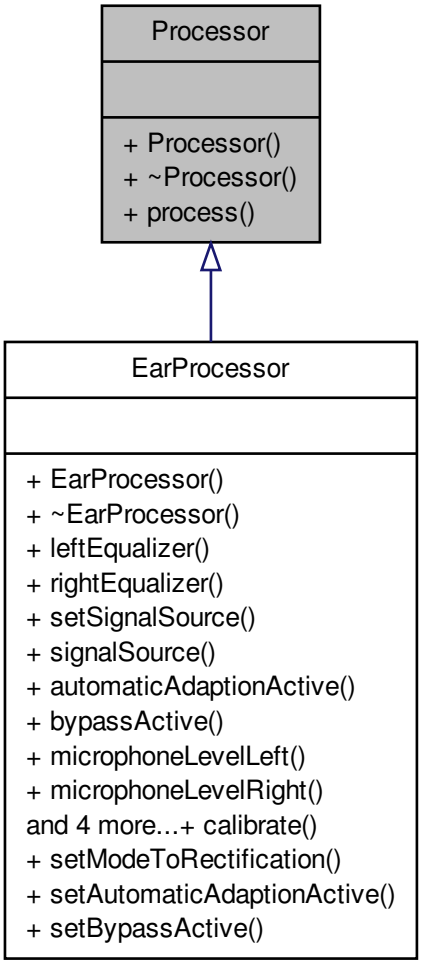
- libear/include/jnoise/prbsgenerator.h

10.14 Processor Class Reference

Processor defines an interface that must be met by processors.

`#include <processor.h>`

Inheritance diagram for Processor:



Public Member Functions

- Processor ()
- virtual ~Processor ()
- virtual void process (int samples)=0

  Called whenever samples have to be processed. Warning: This method is time-critical.

10.14.1 Detailed Description

Processor defines an interface that must be met by processors.

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net)
Otto Ritter (otto.ritter.or@googlemail.com)

Date

09.2011

Definition at line 37 of file processor.h.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 Processor::Processor ( ) [inline]

Constructs a new processor.

Definition at line 42 of file processor.h.

10.14.2.2 virtual Processor::~Processor ( ) [inline, virtual]

Destructor.

Definition at line 45 of file processor.h.

10.14.3 Member Function Documentation

10.14.3.1 virtual void Processor::process (int *samples*) [pure virtual]

Called whenever samples have to be processed. Warning: This method is time-critical.

Parameters

samples	Number of samples.

Implemented in EarProcessor.

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

- libear/include/processor.h

10.15 RandomGenerator Class Reference

`#include <randomgenerator.h>`

Public Member Functions

- RandomGenerator (void)
- void init (uint32_t s)
- uint32_t irand (void)
- double urand (void)
- double grand (void)
- void grand (double *x, double *y)
- float urandf (void)
- float grandf (void)
- void grandf (float *x, float *y)
- ~RandomGenerator (void)
- RandomGenerator (const RandomGenerator &)
- RandomGenerator & operator= (const RandomGenerator &)

10.15.1 Detailed Description

Definition at line 27 of file randomgenerator.h.

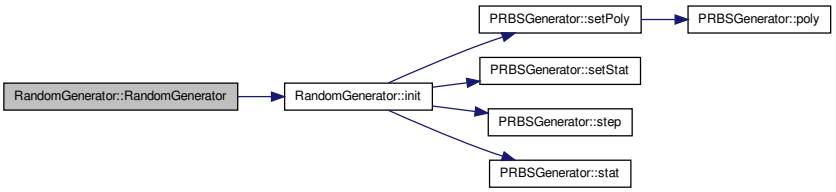
10.15.2 Constructor & Destructor Documentation

10.15.2.1 RandomGenerator::RandomGenerator (void)

Definition at line 33 of file randomgenerator.cpp.

References init().

Here is the call graph for this function:



10.15.2.2 RandomGenerator::~RandomGenerator (void)

Definition at line 39 of file randomgenerator.cpp.

10.15.2.3 RandomGenerator::RandomGenerator (const RandomGenerator &)

10.15.3 Member Function Documentation

10.15.3.1 double RandomGenerator::grand (void)

Definition at line 68 of file randomgenerator.cpp.

References irand().

Here is the call graph for this function:

10.15.3.2 void RandomGenerator::grand (double * x, double * y)

Definition at line 94 of file randomgenerator.cpp.

References irand().

Here is the call graph for this function:

10.15.3.3 float RandomGenerator::grandf (void)

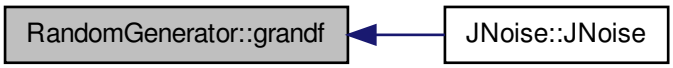
Definition at line 112 of file randomgenerator.cpp.

References irand().

Referenced by JNoise::JNoise().

Here is the call graph for this function:

Here is the caller graph for this function:



10.15.3.4 void RandomGenerator::grandf (float * x, float * y)

Definition at line 138 of file randomgenerator.cpp.

References irand().

Here is the call graph for this function:

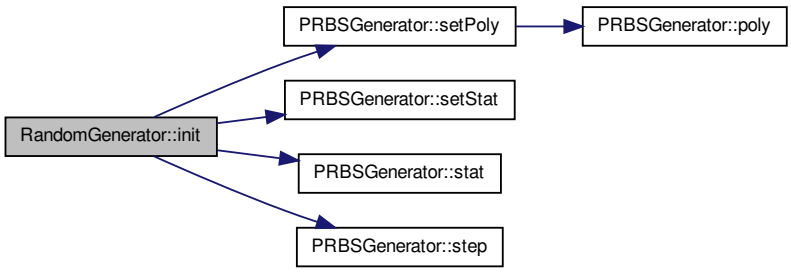
10.15.3.5 void RandomGenerator::init (uint32_t *s*)

Definition at line 44 of file randomgenerator.cpp.

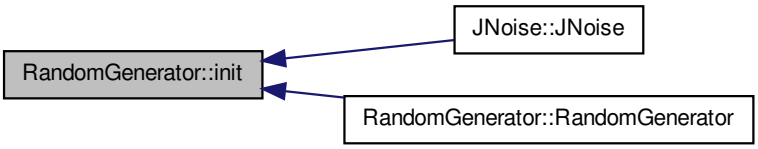
References PRBSGenerator::G32, PRBSGenerator::setPoly(), PRBSGenerator::setStat(), PRBSGenerator::stat(), and PRBSGenerator::step().

Referenced by JNoise::JNoise(), and RandomGenerator().

Here is the call graph for this function:



Here is the caller graph for this function:

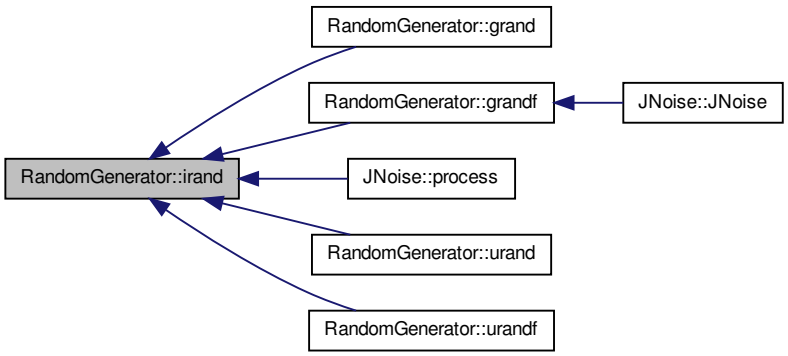


10.15.3.6 uint32_t RandomGenerator::irand (void) [inline]

Definition at line 35 of file randomgenerator.h.

Referenced by grand(), grandf(), JNoise::process(), urand(), and urandf().

Here is the caller graph for this function:



10.15.3.7 RandomGenerator& RandomGenerator::operator= (const RandomGenerator &)

10.15.3.8 double RandomGenerator::urand (void) [inline]

Definition at line 45 of file randomgenerator.h.

References irand().

Here is the call graph for this function:

10.15.3.9 float RandomGenerator::urandf (void) [inline]

Definition at line 48 of file randomgenerator.h.

References irand().

Here is the call graph for this function:

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

- libear/include/jnoise/randomgenerator.h
- libear/src/jnoise/randomgenerator.cpp

10.16 SemaphoreLocker Class Reference

`#include <jackadapter.h>`

Public Member Functions

- SemaphoreLocker (QSemaphore *semaphore)
- ~SemaphoreLocker ()

10.16.1 Detailed Description

Author

Jacob Dawid (jacob.dawid@cybercatalyst.net) Helper class for locking semaphores in scopes.

Definition at line 46 of file jackadapter.h.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 SemaphoreLocker::SemaphoreLocker (QSemaphore * *semaphore*) [inline]

Constructs a locker, locks the given semaphore on construction.

Definition at line 49 of file jackadapter.h.

10.16.2.2 SemaphoreLocker::~SemaphoreLocker ( ) [inline]

Destructor. Unlocks the semaphore.

Definition at line 57 of file jackadapter.h.

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

- libear/include/jackadapter.h

10.17 StereoPort Struct Reference

`#include <jackadapter.h>`

Public Member Functions

- jack_default_audio_sample_t * leftChannelBuffer (int samples)
- jack_default_audio_sample_t * rightChannelBuffer (int samples)

Friends

- class JackAdapter

10.17.1 Detailed Description

Defines a stereo port.

Definition at line 67 of file jackadapter.h.

10.17.2 Member Function Documentation

10.17.2.1 jack_default_audio_sample_t* StereoPort::leftChannelBuffer (int *samples*) [inline]

Provides the address to the sample bufferfor the left channel.

Parameters

samples	Number of samples.

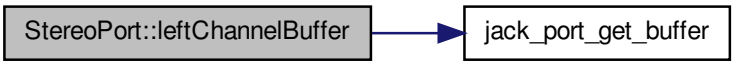
Returns

Pointer to samples buffer.

Definition at line 74 of file jackadapter.h.

References jack_port_get_buffer().

Here is the call graph for this function:



10.17.2.2 jack_default_audio_sample_t* StereoPort::rightChannelBuffer (int *samples*) [inline]

Provides the address to the sample buffer for the right channel.

Parameters

samples	Number of samples.

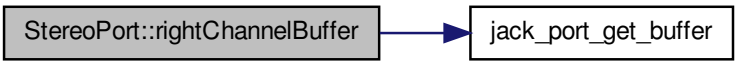
Returns

Pointer to samples buffer.

Definition at line 84 of file jackadapter.h.

References jack_port_get_buffer().

Here is the call graph for this function:



10.17.3 Friends And Related Function Documentation

10.17.3.1 friend class JackAdapter [friend]

Definition at line 68 of file jackadapter.h.

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

- libear/include/jackadapter.h

10.18 VisualizerWidget Class Reference

View counterpart for the EAR processor. This class is the view counterpart for the EAR processor, which acts as the model class.

`#include <visualizerwidget.h>`

Public Member Functions

- VisualizerWidget (EarProcessor *earProcessor, QWidget *parent=0)

Protected Member Functions

- void initializeGL ()
- void resizeGL (int w, int h)
- void paintGL ()
- void wheelEvent (QWheelEvent *wheelEvent)
- void mousePressEvent (QMouseEvent *mouseEvent)
- void mouseReleaseEvent (QMouseEvent *mouseEvent)
- void mouseMoveEvent (QMouseEvent *mouseEvent)

10.18.1 Detailed Description

View counterpart for the EAR processor. This class is the view counterpart for the EAR processor, which acts as the model class.

Definition at line 17 of file visualizerwidget.h.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 VisualizerWidget::VisualizerWidget (EarProcessor * *earProcessor*, QWidget * *parent* = 0)

Constructs a new view that will be attached to the given EAR processor.

Parameters

earProcessor	EAR processor to which this view will be attached.
parent	Parent widget for the Qt framework.

Definition at line 5 of file visualizerwidget.cpp.

10.18.3 Member Function Documentation

10.18.3.1 void VisualizerWidget::initializeGL ( ) [protected]

Reimplemented from QGLWidget.

Definition at line 26 of file visualizerwidget.cpp.

10.18.3.2 void VisualizerWidget::mouseMoveEvent (QMouseEvent * *mouseEvent*) [protected]

Reimplemented from QGLWidget

Parameters

mouseEvent	Mouse event.

Definition at line 242 of file visualizerwidget.cpp.

10.18.3.3 void VisualizerWidget::mousePressEvent (QMouseEvent * *mouseEvent*) [protected]

Reimplemented from QGLWidget.

Parameters

mouseEvent	Mouse event.

Definition at line 229 of file visualizerwidget.cpp.

10.18.3.4 void VisualizerWidget::mouseReleaseEvent (QMouseEvent * *mouseEvent*) [protected]

Reimplemented from QGLWidget.

Parameters

mouseEvent	Mouse event.

Definition at line 235 of file visualizerwidget.cpp.

10.18.3.5 void VisualizerWidget::paintGL ( ) [protected]

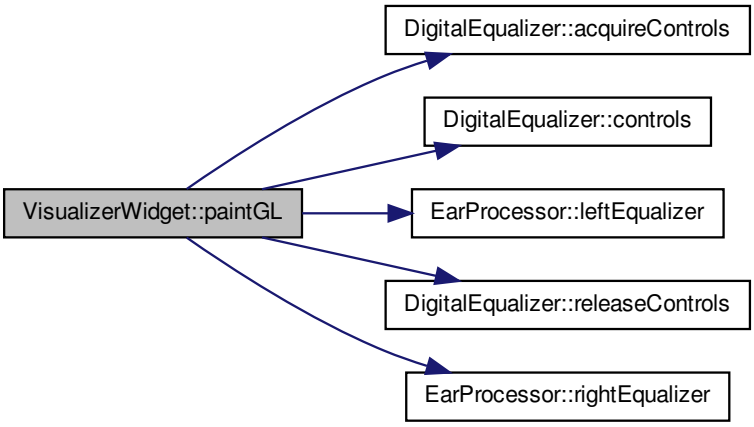
Reimplemented from QGLWidget.

Definition at line 43 of file visualizerwidget.cpp.

References DigitalEqualizer::acquireControls(), DigitalEqualizer::controls(), Ear-

Processor::leftEqualizer(), DigitalEqualizer::releaseControls(), and EarProcessor::right-Equalizer().

Here is the call graph for this function:



10.18.3.6 void VisualizerWidget::resizeGL (int *w,* int *h*) [protected]

Reimplemented from QGLWidget.

Parameters

w	Width of the new OpenGL viewport.
h	Height of the new OpenGL viewport.

Definition at line 35 of file visualizerwidget.cpp.

10.18.3.7 void VisualizerWidget::wheelEvent (QWheelEvent * *wheelEvent*) [protected]

Reimplemented from QGLWidget.

Parameters

wheelEvent	Wheel event.

Definition at line 217 of file visualizerwidget.cpp.

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

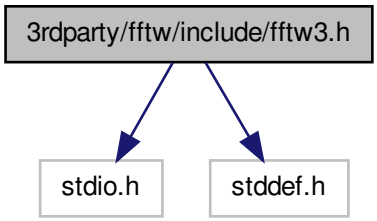
- gui/include/visualizerwidget.h
- gui/src/visualizerwidget.cpp

Chapter 11

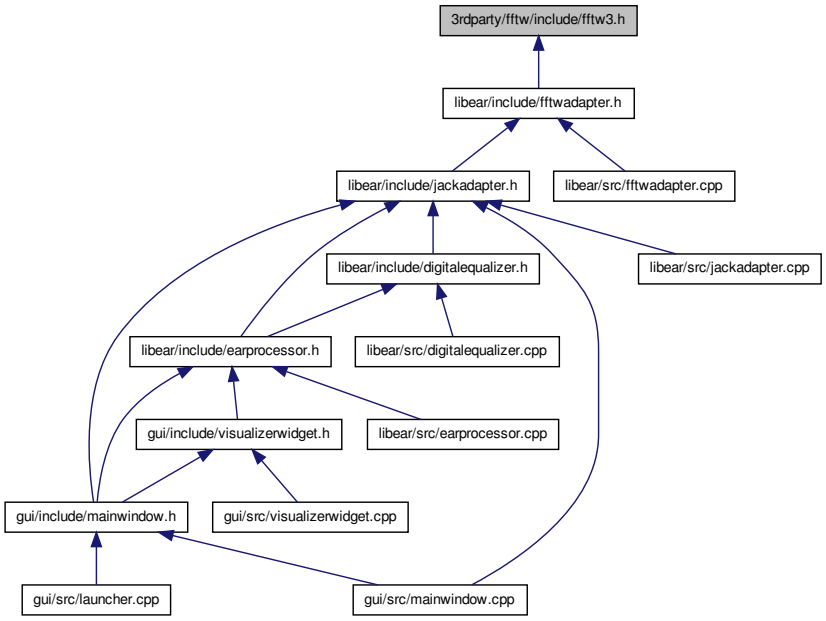
File Documentation

11.1 3rdparty/fftw/include/fftw3.h File Reference

`#include <stdio.h>` `#include <stddef.h>` Include dependency graph for fftw3.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct fftw_iodim_do_not_use_me
- struct fftw_iodim64_do_not_use_me

Defines

- #define FFTW_DEFINE_COMPLEX(R, C) typedef R C[2]
- #define FFTW_CONCAT(prefix, name) prefix ## name
- #define FFTW_MANGLE_DOUBLE(name) FFTW_CONCAT(fftw_, name)
- #define FFTW_MANGLE_FLOAT(name) FFTW_CONCAT(fftwf_, name)
- #define FFTW_MANGLE_LONG_DOUBLE(name) FFTW_CONCAT(fftwl_, name)
- #define FFTW_DLL
- #define FFTW_EXTERN extern
- #define FFTW_DEFINE_API(X, R, C)
- #define FFTW_FORWARD (-1)
- #define FFTW_BACKWARD (+1)
- #define FFTW_NO_TIMELIMIT (-1.0)
- #define FFTW_MEASURE (0U)

- #define FFTW_DESTROY_INPUT (1U << 0)
- #define FFTW_UNALIGNED (1U << 1)
- #define FFTW_CONSERVE_MEMORY (1U << 2)
- #define FFTW_EXHAUSTIVE (1U << 3) /* NO_EXHAUSTIVE is default */
- #define FFTW_PRESERVE_INPUT (1U << 4) /* cancels FFTW_DESTROY_INPUT */
- #define FFTW_PATIENT (1U << 5) /* IMPATIENT is default */
- #define FFTW_ESTIMATE (1U << 6)
- #define FFTW_ESTIMATE_PATIENT (1U << 7)
- #define FFTW_BELIEVE_PCOST (1U << 8)
- #define FFTW_NO_DFT_R2HC (1U << 9)
- #define FFTW_NO_NONTHREADED (1U << 10)
- #define FFTW_NO_BUFFERING (1U << 11)
- #define FFTW_NO_INDIRECT_OP (1U << 12)
- #define FFTW_ALLOW_LARGE_GENERIC (1U << 13) /* NO_LARGE_GENERIC is default */
- #define FFTW_NO_RANK_SPLITS (1U << 14)
- #define FFTW_NO_VRANK_SPLITS (1U << 15)
- #define FFTW_NO_VRECURSE (1U << 16)
- #define FFTW_NO_SIMD (1U << 17)
- #define FFTW_NO_SLOW (1U << 18)
- #define FFTW_NO_FIXED_RADIX_LARGE_N (1U << 19)
- #define FFTW_ALLOW_PRUNING (1U << 20)
- #define FFTW_WISDOM_ONLY (1U << 21)

Enumerations

- enum fftw_r2r_kind_do_not_use_me { FFTW_R2HC = 0, FFTW_HC2R = 1, FFTW_DHT = 2, FFTW_REDFT00 = 3, FFTW_REDFT01 = 4, FFTW_REDFT10 = 5, FFTW_REDFT11 = 6, FFTW_RODFT00 = 7, FFTW_RODFT01 = 8, FFTW_RODFT10 = 9, FFTW_RODFT11 = 10 }

11.1.1 Define Documentation

11.1.1.1 #define FFTW_ALLOW_LARGE_GENERIC (1U << 13) /* NO_LARGE_GENERIC is default */

Definition at line 369 of file fftw3.h.

11.1.1.2 #define FFTW_ALLOW_PRUNING (1U << 20)

Definition at line 376 of file fftw3.h.

11.1.1.3 #define FFTW_BACKWARD (+1)

Definition at line 348 of file fftw3.h.

Referenced by FftwAdapter::performInverseFFT().

11.1.1.4 #define FFTW_BELIEVE_PCOST (1U << 8)

Definition at line 364 of file fftw3.h.

11.1.1.5 #define FFTW_CONCAT(*prefix, name*) prefix ## name

Definition at line 65 of file fftw3.h.

11.1.1.6 #define FFTW_CONSERVE_MEMORY (1U << 2)

Definition at line 356 of file fftw3.h.

11.1.1.7 #define FFTW_DEFINE_API(*X, R, C*)

Definition at line 118 of file fftw3.h.

11.1.1.8 #define FFTW_DEFINE_COMPLEX(*R, C*) typedef R C[2]

Definition at line 62 of file fftw3.h.

11.1.1.9 #define FFTW_DESTROY_INPUT (1U << 0)

Definition at line 354 of file fftw3.h.

11.1.1.10 #define FFTW_DLL

Definition at line 72 of file fftw3.h.

11.1.1.11 #define FFTW_ESTIMATE (1U << 6)

Definition at line 360 of file fftw3.h.

Referenced by FftwAdapter::performFFT(), and FftwAdapter::performInverseFFT().

11.1.1.12 #define FFTW_ESTIMATE_PATIENT (1U << 7)

Definition at line 363 of file fftw3.h.

11.1.1.13 #define FFTW_EXHAUSTIVE (1U << 3) /* NO_EXHAUSTIVE is default */

Definition at line 357 of file fftw3.h.

11.1.1.14 #define FFTW_EXTERN extern

Definition at line 87 of file fftw3.h.

11.1.1.15 #define FFTW_FORWARD (-1)

Definition at line 347 of file fftw3.h.

Referenced by FftwAdapter::performFFT().

11.1.1.16 #define FFTW_MANGLE_DOUBLE(*name*) FFTW_CONCAT(fftw_, name)

Definition at line 66 of file fftw3.h.

11.1.1.17 #define FFTW_MANGLE_FLOAT(*name*) FFTW_CONCAT(fftwf_, name)

Definition at line 67 of file fftw3.h.

11.1.1.18 #define FFTW_MANGLE_LONG_DOUBLE(*name*) FFTW_CONCAT(fftwl_, name)

Definition at line 68 of file fftw3.h.

11.1.1.19 #define FFTW_MEASURE (0U)

Definition at line 353 of file fftw3.h.

11.1.1.20 #define FFTW_NO_BUFFERING (1U << 11)

Definition at line 367 of file fftw3.h.

11.1.1.21 #define FFTW_NO_DFT_R2HC (1U << 9)

Definition at line 365 of file fftw3.h.

11.1.1.22 #define FFTW_NO_FIXED_RADIX_LARGE_N (1U << 19)

Definition at line 375 of file fftw3.h.

11.1.1.23 #define FFTW_NO_INDIRECT_OP (1U << 12)

Definition at line 368 of file fftw3.h.

11.1.1.24 #define FFTW_NO_NONTHREADED (1U << 10)

Definition at line 366 of file fftw3.h.

11.1.1.25 #define FFTW_NO_RANK_SPLITS (1U << 14)

Definition at line 370 of file fftw3.h.

11.1.1.26 #define FFTW_NO_SIMD (1U << 17)

Definition at line 373 of file fftw3.h.

11.1.1.27 #define FFTW_NO_SLOW (1U << 18)

Definition at line 374 of file fftw3.h.

11.1.1.28 #define FFTW_NO_TIMELIMIT (-1.0)

Definition at line 350 of file fftw3.h.

11.1.1.29 #define FFTW_NO_VRANK_SPLITS (1U << 15)

Definition at line 371 of file fftw3.h.

11.1.1.30 #define FFTW_NO_VRECURSE (1U << 16)

Definition at line 372 of file fftw3.h.

11.1.1.31 #define FFTW_PATIENT (1U << 5) /* IMPATIENT is default */

Definition at line 359 of file fftw3.h.

11.1.1.32 #define FFTW_PRESERVE_INPUT (1U << 4) /* cancels FFTW_DESTROY_INPUT */

Definition at line 358 of file fftw3.h.

11.1.1.33 #define FFTW_UNALIGNED (1U << 1)

Definition at line 355 of file fftw3.h.

11.1.1.34 #define FFTW_WISDOM_ONLY (1U << 21)

Definition at line 377 of file fftw3.h.

11.1.2 Enumeration Type Documentation

11.1.2.1 enum fftw_r2r_kind_do_not_use_me

Enumerator:

FFTW_R2HC

FFTW_HC2R

FFTW_DHT

FFTW_REDFT00

FFTW_REDFT01

FFTW_REDFT10

FFTW_REDFT11

FFTW_RODFT00

FFTW_RODFT01

FFTW_RODFT10

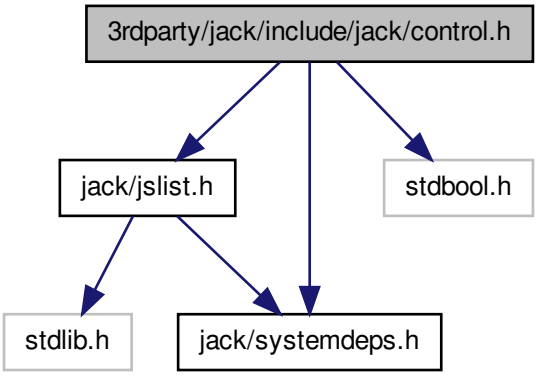
FFTW_RODFT11

Definition at line 90 of file fftw3.h.

11.2 3rdparty/jack/include/jack/control.h File Reference

JACK control API.

`#include <jack/jslist.h>` `#include <jack/systemdeps.h>` × `#include <stdbool.h>` Include dependency graph for control.h:



Data Structures

- union jackctl_parameter_value

  Type for parameter value.

Defines

- #define JACK_PARAM_MAX (JackParamBool + 1)

  Max value that jackctl_param_type_t type can have.
- #define JACK_PARAM_STRING_MAX 127

  Max length of string parameter value, excluding terminating null char.

Typedefs

- typedef struct jackctl_server
- typedef struct jackctl_driver
- typedef struct jackctl_internal
- typedef struct jackctl_parameter

Enumerations

- enum jackctl_param_type_t { JackParamInt = 1, JackParamUInt, JackParamChar, JackParamString, JackParamBool }

Functions

- sigset_t jackctl_setup_signals (unsigned int flags)
- void jackctl_wait_signals (sigset_t signals)
- jackctl_server_t * jackctl_server_create (bool(*on_device_acquire)(const char *device_name), void(*on_device_release)(const char *device_name))
- void jackctl_server_destroy (jackctl_server_t *server)
- bool jackctl_server_open (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_start (jackctl_server_t *server)
- bool jackctl_server_stop (jackctl_server_t *server)
- bool jackctl_server_close (jackctl_server_t *server)
- const JSList * jackctl_server_get_drivers_list (jackctl_server_t *server)
- const JSList * jackctl_server_get_parameters (jackctl_server_t *server)
- const JSList * jackctl_server_get_internals_list (jackctl_server_t *server)
- bool jackctl_server_load_internal (jackctl_server_t *server, jackctl_internal_t *internal)
- bool jackctl_server_unload_internal (jackctl_server_t *server, jackctl_internal_t *internal)
- bool jackctl_server_add_slave (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_remove_slave (jackctl_server_t *server, jackctl_driver_t *driver)
- bool jackctl_server_switch_master (jackctl_server_t *server, jackctl_driver_t *driver)
- const char * jackctl_driver_get_name (jackctl_driver_t *driver)
- const JSList * jackctl_driver_get_parameters (jackctl_driver_t *driver)
- const char * jackctl_internal_get_name (jackctl_internal_t *internal)
- const JSList * jackctl_internal_get_parameters (jackctl_internal_t *internal)
- const char * jackctl_parameter_get_name (jackctl_parameter_t *parameter)
- const char * jackctl_parameter_get_short_description (jackctl_parameter_t *parameter)
- const char * jackctl_parameter_get_long_description (jackctl_parameter_t *parameter)
- jackctl_param_type_t jackctl_parameter_get_type (jackctl_parameter_t *parameter)
- char jackctl_parameter_get_id (jackctl_parameter_t *parameter)
- bool jackctl_parameter_is_set (jackctl_parameter_t *parameter)
- bool jackctl_parameter_reset (jackctl_parameter_t *parameter)
- union jackctl_parameter_value jackctl_parameter_get_value (jackctl_parameter_t *parameter)
- bool jackctl_parameter_set_value (jackctl_parameter_t *parameter, const union jackctl_parameter_value *value_ptr)
- union jackctl_parameter_value jackctl_parameter_get_default_value (jackctl_parameter_t *parameter)
- bool jackctl_parameter_has_range_constraint (jackctl_parameter_t *parameter)
- bool jackctl_parameter_has_enum_constraint (jackctl_parameter_t *parameter)
- uint32_t jackctl_parameter_get_enum_constraints_count (jackctl_parameter_t *parameter)
- union jackctl_parameter_value jackctl_parameter_get_enum_constraint_value (jackctl_parameter_t *parameter, uint32_t index)

- const char * jackctl_parameter_get_enum_constraint_description (jackctl_parameter_t *parameter, uint32_t index)
- void jackctl_parameter_get_range_constraint (jackctl_parameter_t *parameter, union jackctl_parameter_value *min_ptr, union jackctl_parameter_value *max_ptr)
- bool jackctl_parameter_constraint_is_strict (jackctl_parameter_t *parameter)
- bool jackctl_parameter_constraint_is_fake_value (jackctl_parameter_t *parameter)
- void jack_error (const char *format,...)
- void jack_info (const char *format,...)
- void jack_log (const char *format,...)

11.2.1 Detailed Description

JACK control API.

Definition in file control.h.

11.2.2 Define Documentation

11.2.2.1 #define JACK_PARAM_MAX (JackParamBool + 1)

Max value that jackctl_param_type_t type can have.

Definition at line 49 of file control.h.

11.2.2.2 #define JACK_PARAM_STRING_MAX 127

Max length of string parameter value, excluding terminating null char.

Definition at line 52 of file control.h.

11.2.3 Typedef Documentation

11.2.3.1 typedef struct jackctl_driver

opaque type for driver object

Definition at line 69 of file control.h.

11.2.3.2 typedef struct jackctl_internal

opaque type for internal client object

Definition at line 72 of file control.h.

11.2.3.3 typedef struct jackctl_parameter

opaque type for parameter object

Definition at line 75 of file control.h.

11.2.3.4 typedef struct jackctl_server

opaque type for server object

Definition at line 66 of file control.h.

11.2.4 Enumeration Type Documentation

11.2.4.1 enum jackctl_param_type_t

Parameter types, intentionally similar to jack_driver_param_type_t

Enumerator:

JackParamInt value type is a signed integer

JackParamUInt value type is an unsigned integer

JackParamChar value type is a char

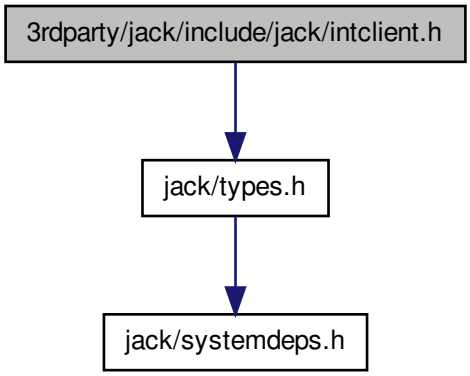
JackParamString value type is a string with max size of JACK_PARAM_STRING_MAX+1 chars

JackParamBool value type is a boolean

Definition at line 39 of file control.h.

11.3 3rdparty/jack/include/jack/intclient.h File Reference

`#include <jack/types.h>` Include dependency graph for intclient.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)

11.3.1 Function Documentation

11.3.1.1 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.

11.3.1.2 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.

11.3.1.3 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.

11.3.1.4 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().

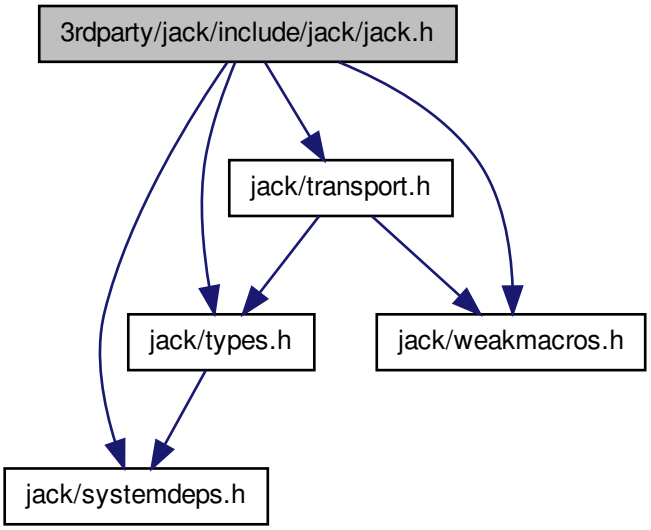
Returns

0 if successful, otherwise JackStatus bits.

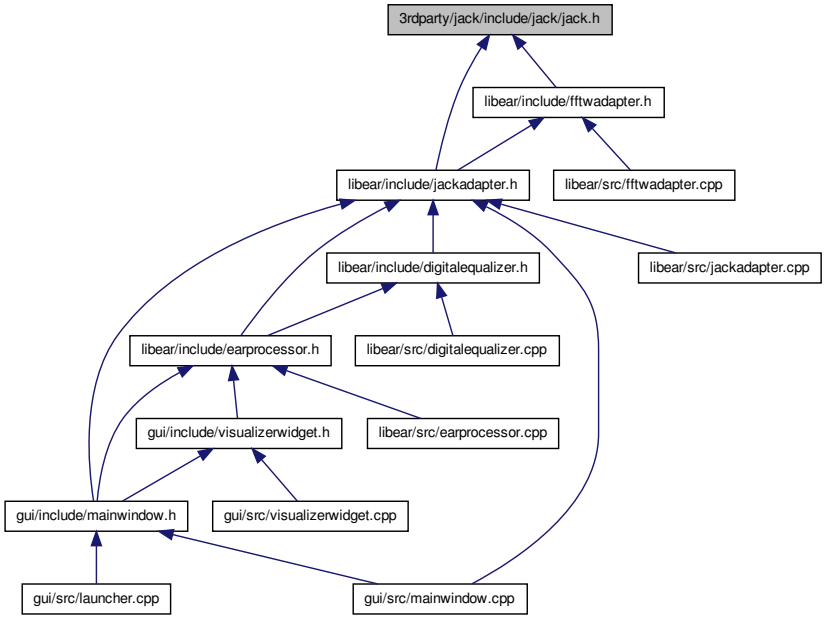
11.4 3rdparty/jack/include/jack/jack.h File Reference

```
#include <jack/systemdeps.h>    #include <jack/types.h>×
#include <jack/transport.h>   #include <jack/weakmacros.-
```

h> Include dependency graph for jack.h:



This graph shows which files directly or indirectly include this file:



Functions

- void jack_get_version (int *major_ptr, int *minor_ptr, int *micro_ptr, int *proto_-ptr) JACK_OPTIONAL_WEAK_EXPORT
- const char * jack_get_version_string () JACK_OPTIONAL_WEAK_EXPORT
- 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
- int jack_get_client_pid (const char *name) JACK_OPTIONAL_WEAK_EXPORT

- jack_native_thread_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 thread_callback, 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 shutdown_callback, void *arg) JACK_WEAK_EXPORT
- void jack_on_info_shutdown (jack_client_t *client, JackInfoShutdownCallback shutdown_callback, void *arg) JACK_OPTIONAL_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_port_rename_callback (jack_client_t *, JackPortRenameCallback rename_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_latency_callback (jack_client_t *, JackLatencyCallback latency_callback, void *) JACK_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
- jack_port_type_id_t jack_port_type_id (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
- 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
- size_t jack_port_type_get_buffer_size (jack_client_t *client, const char *port_type) JACK_WEAK_EXPORT
- void jack_port_set_latency (jack_port_t *, jack_nframes_t) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- void jack_port_get_latency_range (jack_port_t *port, jack_latency_callback_mode_t mode, jack_latency_range_t *range) JACK_WEAK_EXPORT
- void jack_port_set_latency_range (jack_port_t *port, jack_latency_callback_mode_t mode, jack_latency_range_t *range) JACK_WEAK_EXPORT
- int jack_recompute_total_latencies (jack_client_t *) JACK_OPTIONAL_WEAK_EXPORT
- jack_nframes_t jack_port_get_latency (jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- jack_nframes_t jack_port_get_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_EXPORT
- int jack_recompute_total_latency (jack_client_t *, jack_port_t *port) JACK_OPTIONAL_WEAK_DEPRECATED_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

11.4.1 Function Documentation

11.4.1.1 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 JackAdapter::startAudioProcessing().

Here is the caller graph for this function:

11.4.1.2 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

11.4.1.3 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.

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

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

Deprecated Please use jack_client_open().

11.4.1.5 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).

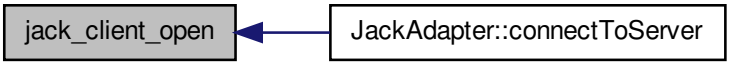
- [JackServerName] *(char *) server_name* selects from among several possible concurrent server instances. Server names are unique to each user. If unspecified, use "default" unless $JACK_DEFAULT_SERVER is defined in the process environment.

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 JackAdapter::connectToServer().

Here is the caller graph for this function:



11.4.1.6 jack_native_thread_t jack_client_thread_id (jack_client_t ∗)

Returns

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

11.4.1.7 int jack_deactivate (jack_client_t ∗ *client*)

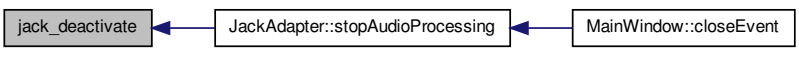
Tell the Jack server to remove this *client* from the process graph. Also, disconnect all ports belonging to it, since inactive clients have no port connections.

Returns

0 on success, otherwise a non-zero error code

Referenced by JackAdapter::stopAudioProcessing().

Here is the caller graph for this function:



11.4.1.8 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.

11.4.1.9 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.

11.4.1.10 int jack_get_client_pid (const char * *name*)

Returns

pid of client. If not available, 0 will be returned.

11.4.1.11 void jack_get_version (int * *major_ptr*, int * *minor_ptr*, int * *micro_ptr*, int * *proto_ptr*)

Note: More documentation can be found in jack/types.h. Call this function to get version of the JACK, in form of several numbers

Parameters

major_ptr	pointer to variable receiving major version of JACK.
minor_ptr	pointer to variable receiving minor version of JACK.
major_ptr	pointer to variable receiving micro version of JACK.
major_ptr	pointer to variable receiving protocol version of JACK.

11.4.1.12 const char* jack_get_version_string ( )

Call this function to get version of the JACK, in form of a string

Returns

Human readable string describing JACK version being used.

11.4.1.13 void jack_internal_client_close (const char * *client_name*)

Remove an internal client from a JACK server.

Deprecated Please use jack_internal_client_load().

11.4.1.14 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.

11.4.1.15 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

11.4.1.16 int jack_set_latency_callback (jack_client_t * , JackLatencyCallback *latency_callback*, void *)

Tell the Jack server to call *latency_callback* whenever it is necessary to recompute the latencies for some or all Jack ports.

latency_callback will be called twice each time it is needed, once being passed JackCaptureLatency and once JackPlaybackLatency. See Managing and determining latency for the definition of each type of latency and related functions.

IMPORTANT: Most JACK clients do NOT need to register a latency callback.

Clients that meet any of the following conditions do NOT need to register a latency callback:

- have only input ports
- have only output ports
- their output is totally unrelated to their input
- their output is not delayed relative to their input (i.e. data that arrives in a given process() callback is processed and output again in the same callback)

Clients NOT registering a latency callback MUST also satisfy this condition:

- have no multiple distinct internal signal pathways

This means that if your client has more than 1 input and output port, and considers them always "correlated" (e.g. as a stereo pair), then there is only 1 (e.g. stereo) signal pathway through the client. This would be true, for example, of a stereo FX rack client that has a left/right input pair and a left/right output pair.

However, this is somewhat a matter of perspective. The same FX rack client could be connected so that its two input ports were connected to entirely separate sources. - Under these conditions, the fact that the client does not register a latency callback MAY result in port latency values being incorrect.

Clients that do not meet any of those conditions SHOULD register a latency callback.

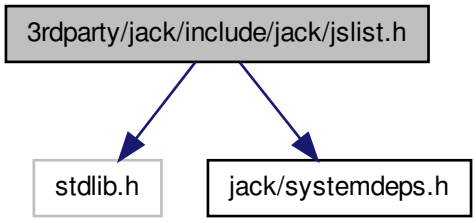
See the documentation for jack_port_set_latency_range() on how the callback should operate. Remember that the *mode* argument given to the latency callback will need to be passed into jack_port_set_latency_range()

Returns

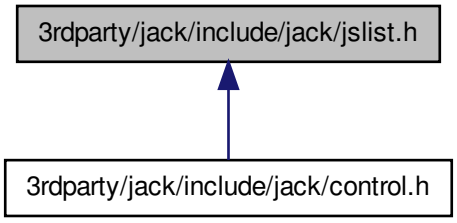
0 on success, otherwise a non-zero error code

11.5 3rdparty/jack/include/jack/jslist.h File Reference

`#include <stdlib.h>` `#include <jack/systemdeps.h>` Include dependency graph for jslist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct _JSList

Defines

- #define jack_slist_next(slist) ((slist) ? (((JSList *)(slist))->next) : NULL)

Typedefs

- typedef struct _JSList
- typedef int(* JCompareFunc)(void *a, void *b)

11.5.1 Define Documentation

11.5.1.1 #define jack_slist_next(*slist*) ((slist) ? (((JSList *)(slist))->next) : NULL)

Definition at line 74 of file jslist.h.

11.5.2 Typedef Documentation

11.5.2.1 typedef struct _JSList

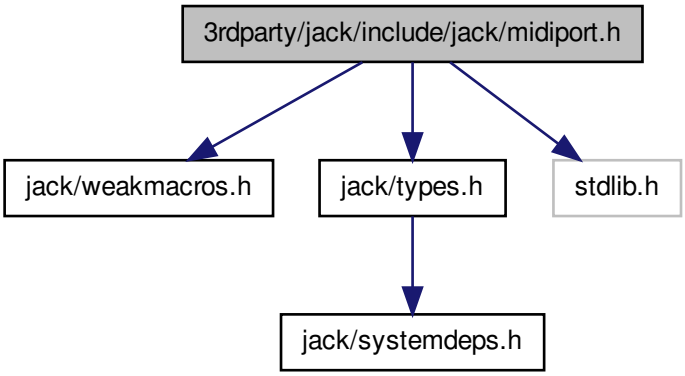
Definition at line 35 of file jslist.h.

11.5.2.2 typedef int(* JCompareFunc)(void *a, void *b)

Definition at line 37 of file jslist.h.

11.6 3rdparty/jack/include/jack/midiport.h File Reference

`#include <jack/weakmacros.h>` `#include <jack/types.h>` × `#include <stdlib.h>` Include dependency graph for midiport.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

11.6.1 Typedef Documentation

11.6.1.1 typedef unsigned char jack_midi_data_t

Type for raw event data contained in jack_midi_event_t.

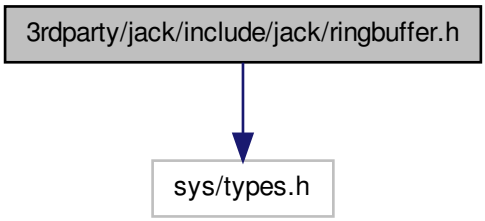
Definition at line 34 of file midiport.h.

11.6.1.2 typedef struct _jack_midi_event jack_midi_event_t

A Jack MIDI event.

11.7 3rdparty/jack/include/jack/ringbuffer.h File Reference

`#include <sys/types.h>` Include dependency graph for ringbuffer.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)
- void jack_ringbuffer_reset_size (jack_ringbuffer_t *rb, size_t sz)
- 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)

11.7.1 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.

Definition in file ringbuffer.h.

11.7.2 Function Documentation

11.7.2.1 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.

11.7.2.2 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.

11.7.2.3 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.

11.7.2.4 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.

11.7.2.5 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.

11.7.2.6 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.

11.7.2.7 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.

11.7.2.8 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.

11.7.2.9 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.

11.7.2.10 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.

11.7.2.11 void jack_ringbuffer_reset_size (jack_ringbuffer_t * rb, size_t sz)

Reset the internal "available" size, and read and write pointers, making an empty buffer.

This is not thread safe.

Parameters

rb	a pointer to the ringbuffer structure.
sz	the new size, that must be less than allocated size.

11.7.2.12 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

11.7.2.13 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.

11.7.2.14 size_t jack_ringbuffer_write_space (const jack_ringbuffer_t * *rb*)

Return the number of bytes available for writing.

Parameters

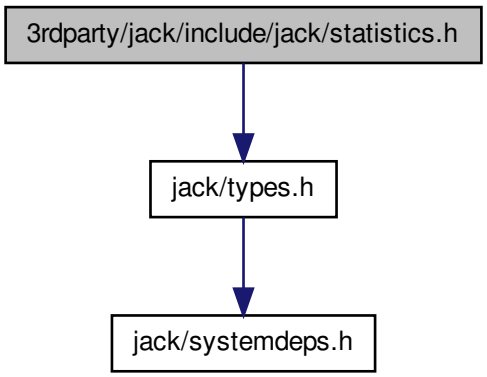
rb	a pointer to the ringbuffer structure.

Returns

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

11.8 3rdparty/jack/include/jack/statistics.h File Reference

`#include <jack/types.h>` Include dependency graph for statistics.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)

11.8.1 Function Documentation

11.8.1.1 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.

11.8.1.2 float jack_get_xrun_delayed_usecs (jack_client_t * *client*)

Returns

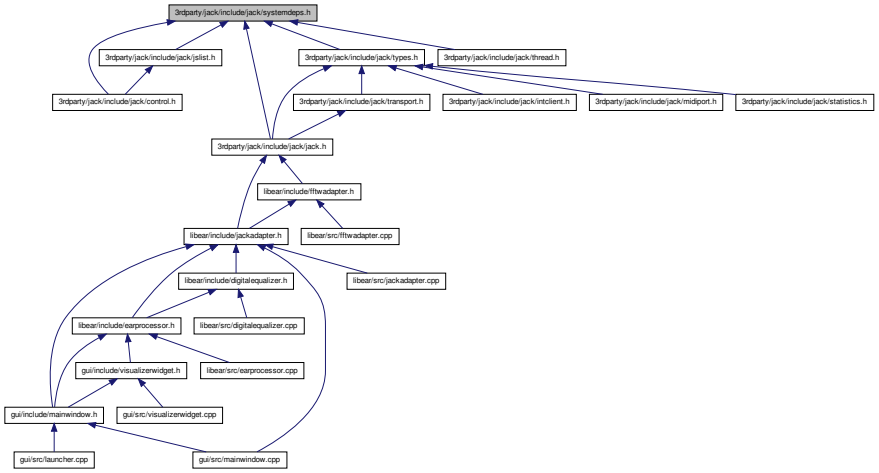
the delay in microseconds due to the most recent XRUN occurrence. This probably only makes sense when called from a JackXRunCallback defined using jack_set_xrun_callback().

11.8.1.3 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.

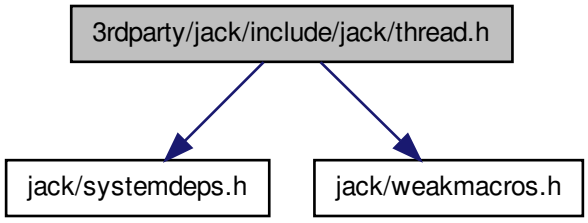
11.9 3rdparty/jack/include/jack/systemdeps.h File Reference

This graph shows which files directly or indirectly include this file:



11.10 3rdparty/jack/include/jack/thread.h File Reference

`#include <jack/systemdeps.h>` `#include <jack/weakmacros.h>` Include dependency graph for thread.h:



Defines

- #define THREAD_STACK 524288

Typedefs

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

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 (jack_native_thread_t thread, int priority) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_create_thread (jack_client_t ∗client, jack_native_thread_t ∗thread, int priority, int realtime, void ∗(∗start_routine)(void ∗), void ∗arg) JACK_OPTIONAL_WEAK_EXPORT
- int jack_drop_real_time_scheduling (jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_stop_thread (jack_client_t ∗client, jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- int jack_client_kill_thread (jack_client_t ∗client, jack_native_thread_t thread) JACK_OPTIONAL_WEAK_EXPORT
- void jack_set_thread_creator (jack_thread_creator_t creator) JACK_OPTIONAL_WEAK_EXPORT

11.10.1 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.

Definition in file thread.h.

11.10.2 Define Documentation

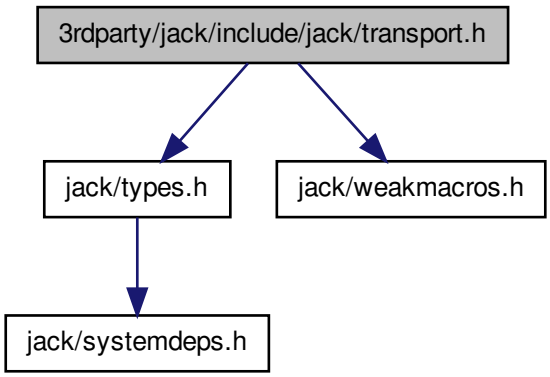
11.10.2.1 #define THREAD_STACK 524288

Definition at line 33 of file thread.h.

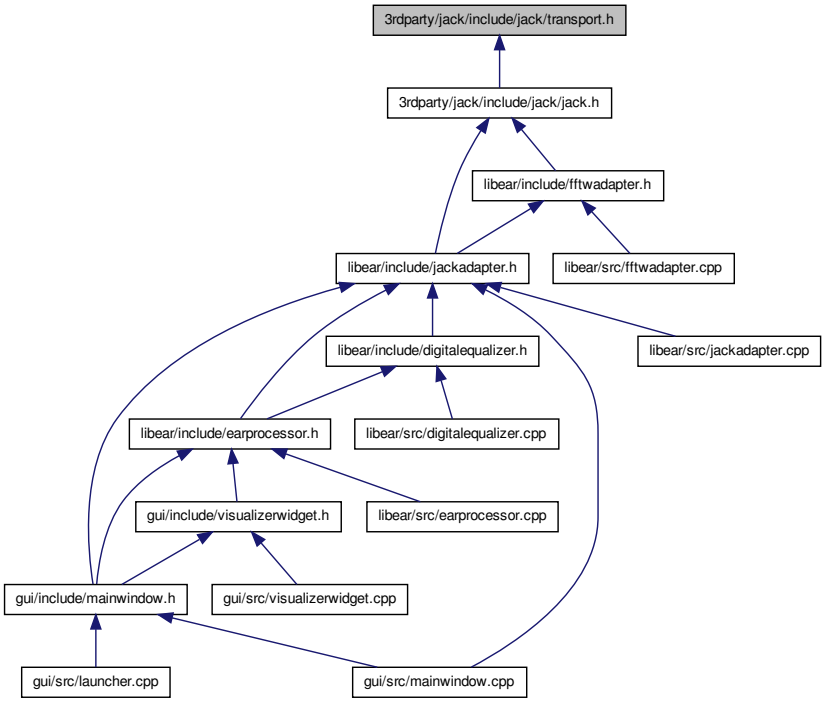
11.11 3rdparty/jack/include/jack/transport.h File Reference

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

Include dependency graph for transport.h:



This graph shows which files directly or indirectly include this file:



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

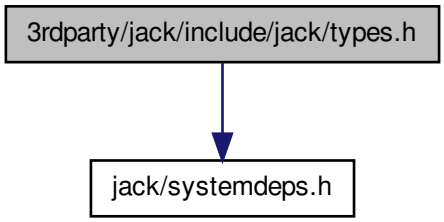
- void jack_transport_stop (jack_client_t *client) JACK_OPTIONAL_WEAK_EXPORT

- void jack_get_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_EXPORT

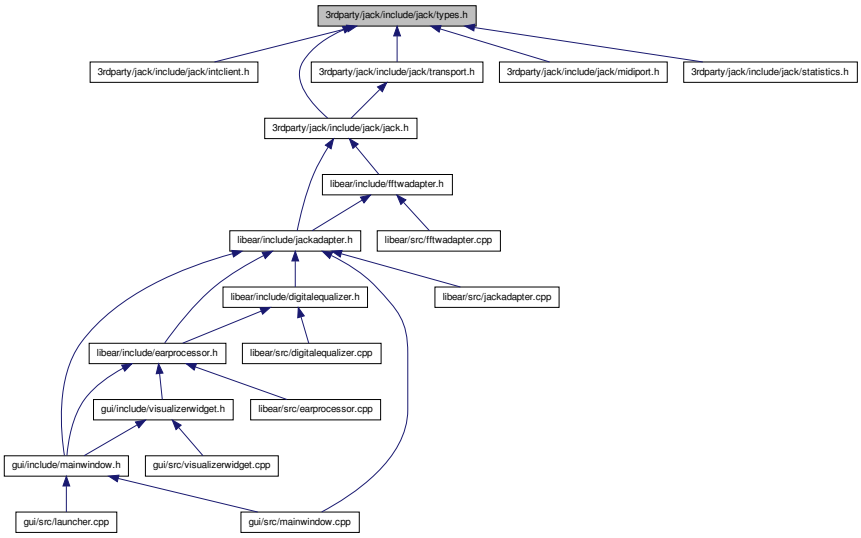
- void jack_set_transport_info (jack_client_t *client, jack_transport_info_t *tinfo) JACK_OPTIONAL_WEAK_EXPORT

11.12 3rdparty/jack/include/jack/types.h File Reference

`#include <jack/systemdeps.h>` Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define JACK_MAX_FRAMES (4294967295U) /∗ This should be UINT32_MAX, but C++ has a problem with that. ∗/
- #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"
- #define JACK_POSITION_MASK (JackPositionBBT|JackPositionTimecode)
- #define EXTENDED_TIME_INFO

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
- typedef struct _jack_client
- typedef uint32_t jack_port_id_t
- typedef uint32_t jack_port_type_id_t

Enumerations

- enum JackOptions { JackNullOption = 0x00, JackNoStartServer = 0x01, JackUseExactName = 0x02, JackServerName = 0x04, JackLoadName = 0x08, JackLoadInit = 0x10, JackSessionID = 0x20 }

11.12.1 Define Documentation

11.12.1.1 #define EXTENDED_TIME_INFO

11.12.1.2 #define JACK_DEFAULT_AUDIO_TYPE "32 bit float mono audio"

Referenced by JackAdapter::registerStereoInputPort(), and JackAdapter::registerStereoOutputPort().

11.12.1.3 #define JACK_DEFAULT_MIDI_TYPE "8 bit raw midi"

11.12.1.4 #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().

Definition at line 48 of file types.h.

11.12.1.5 #define JACK_MAX_FRAMES (4294967295U) /* This should be UINT32_MAX, but C++ has a problem with that. */

Maximum value that can be stored in jack_nframes_t

Definition at line 36 of file types.h.

11.12.1.6 #define JACK_POSITION_MASK (JackPositionBBT|JackPositionTimecode)

11.12.1.7 #define JackLoadOptions (JackLoadInit|JackLoadName|JackUseExactName)

Valid options for loading an internal client.

Definition at line 128 of file types.h.

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

Valid options for opening an external client.

Definition at line 125 of file types.h.

11.12.2 Typedef Documentation

11.12.2.1 typedef struct _jack_client

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

Definition at line 67 of file types.h.

11.12.2.2 typedef struct _jack_port

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

Definition at line 61 of file types.h.

11.12.2.3 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>.

Definition at line 55 of file types.h.

11.12.2.4 typedef uint32_t jack_nframes_t

Type used to represent sample frame counts.

Definition at line 31 of file types.h.

11.12.2.5 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.

Definition at line 73 of file types.h.

11.12.2.6 typedef uint32_t jack_port_type_id_t

Definition at line 75 of file types.h.

11.12.2.7 typedef int32_t jack_shmsize_t

Definition at line 26 of file types.h.

11.12.2.8 typedef uint64_t jack_time_t

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

Definition at line 42 of file types.h.

11.12.3 Enumeration Type Documentation

11.12.3.1 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.

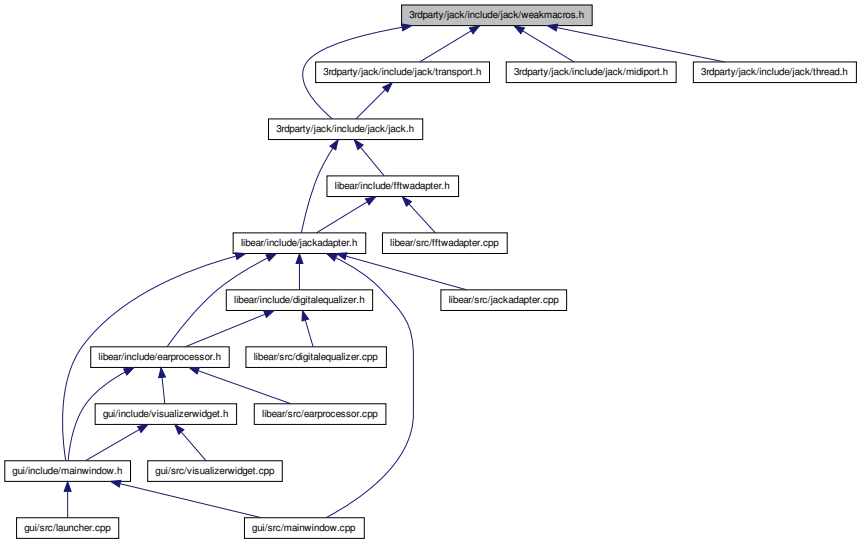
JackSessionID pass a SessionID Token this allows the sessionmanager to identify the client again.

Definition at line 80 of file types.h.

11.13 3rdparty/jack/include/jack/weakjack.h File Reference

11.14 3rdparty/jack/include/jack/weakmacros.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define JACK_OPTIONAL_WEAK_EXPORT

11.14.1 Define Documentation

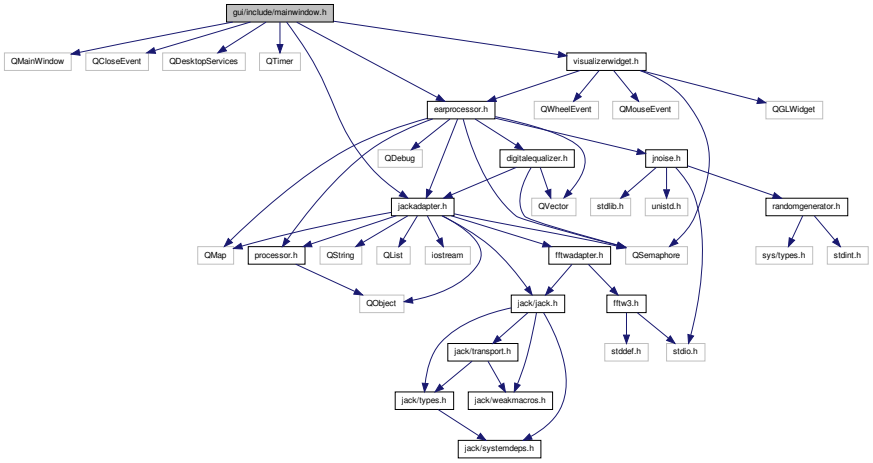
11.14.1.1 #define JACK_OPTIONAL_WEAK_EXPORT

Definition at line 55 of file weakmacros.h.

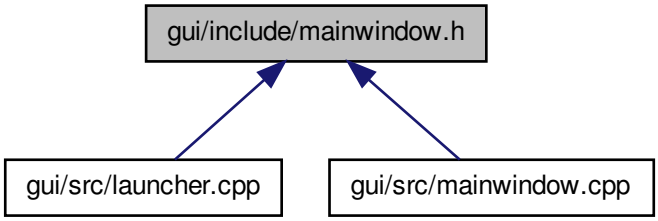
11.15 gui/include/mainwindow.h File Reference

```
#include <QMainWindow> #include <QCloseEvent> #include <-
QDesktopServices> #include <QTimer> #include "jackadapter.-
h" #include "earprocessor.h" #include "visualizerwidget.-
```

h" Include dependency graph for mainwindow.h:



This graph shows which files directly or indirectly include this file:



Data Structures

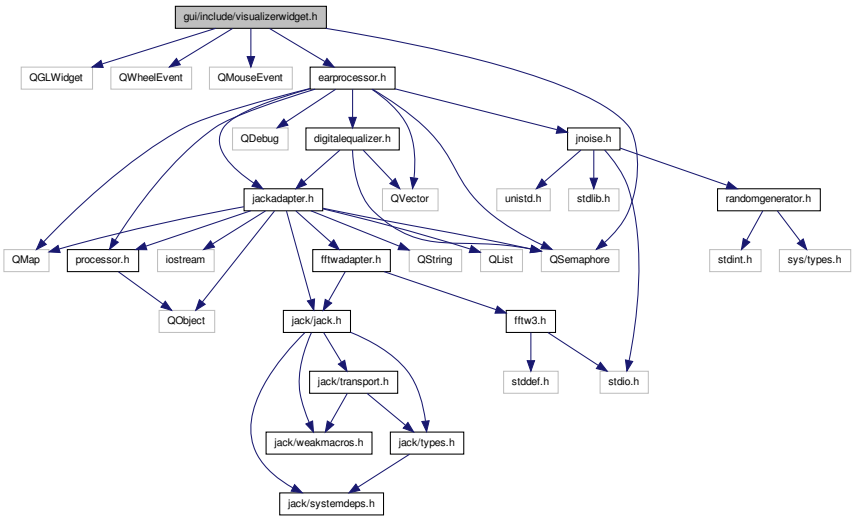
- class MainWindow

Namespaces

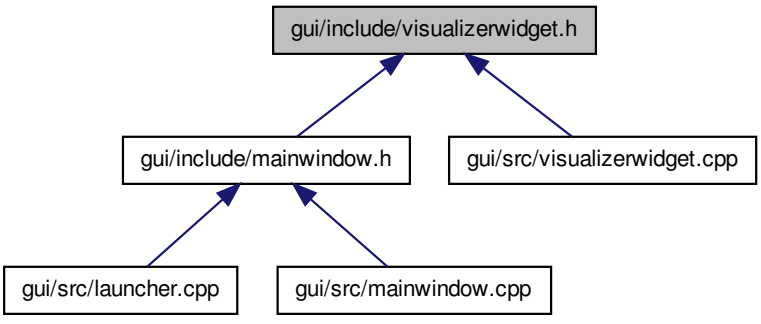
- namespace Ui

11.16 gui/include/visualizerwidget.h File Reference

`#include <QGLWidget> #include <QWheelEvent> #include <QMouseEvent> #include <QSemaphore> #include "earprocessor.h"` Include dependency graph for visualizerwidget.h:



This graph shows which files directly or indirectly include this file:



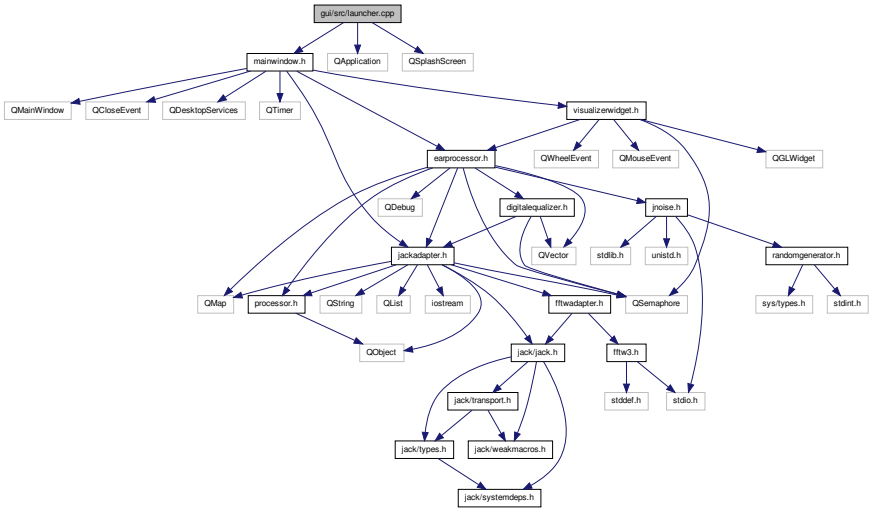
Data Structures

- class VisualizerWidget

  View counterpart for the EAR processor. This class is the view counterpart for the EAR processor, which acts as the model class.

11.17 gui/src/launcher.cpp File Reference

`#include "mainwindow.h" #include <QApplication> #include <QSplashScreen>` Include dependency graph for launcher.cpp:



Functions

- int main (int argc, char ∗argv[])

11.17.1 Function Documentation

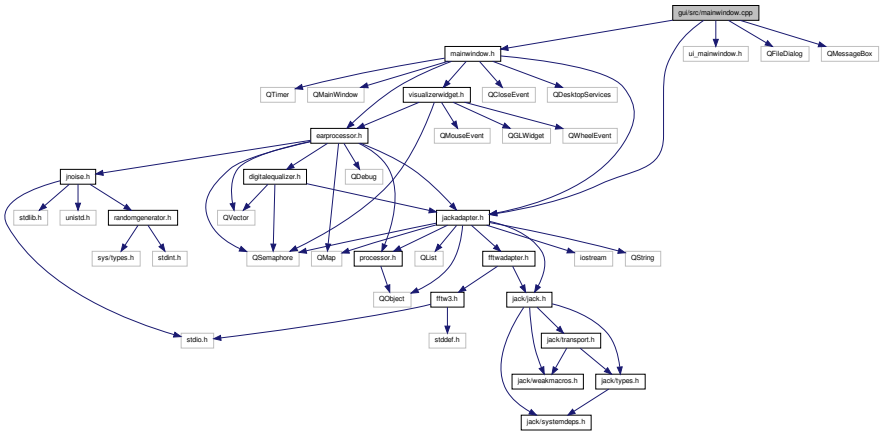
11.17.1.1 int main (int *argc*, char ∗ *argv[]*)

Definition at line 63 of file launcher.cpp.

11.18 gui/src/mainwindow.cpp File Reference

`#include "mainwindow.h" #include "ui_mainwindow.h" #include "jackadapter.h" #include <QFileDialog> #include <QMessage-`

Box> Include dependency graph for mainwindow.cpp:



Defines

- #define MUSIC_COMBO_TEXT "Ext. Source"
- #define WHITE_NOISE_COMBO_TEXT "White Noise"
- #define PINK_NOISE_COMBO_TEXT "Pink Noise"
- #define FILE_TYPES "∗.csv"

11.18.1 Define Documentation

11.18.1.1 #define FILE_TYPES "∗.csv"

Definition at line 31 of file mainwindow.cpp.

11.18.1.2 #define MUSIC_COMBO_TEXT "Ext. Source"

Definition at line 28 of file mainwindow.cpp.

Referenced by MainWindow::MainWindow().

11.18.1.3 #define PINK_NOISE_COMBO_TEXT "Pink Noise"

Definition at line 30 of file mainwindow.cpp.

Referenced by MainWindow::MainWindow().

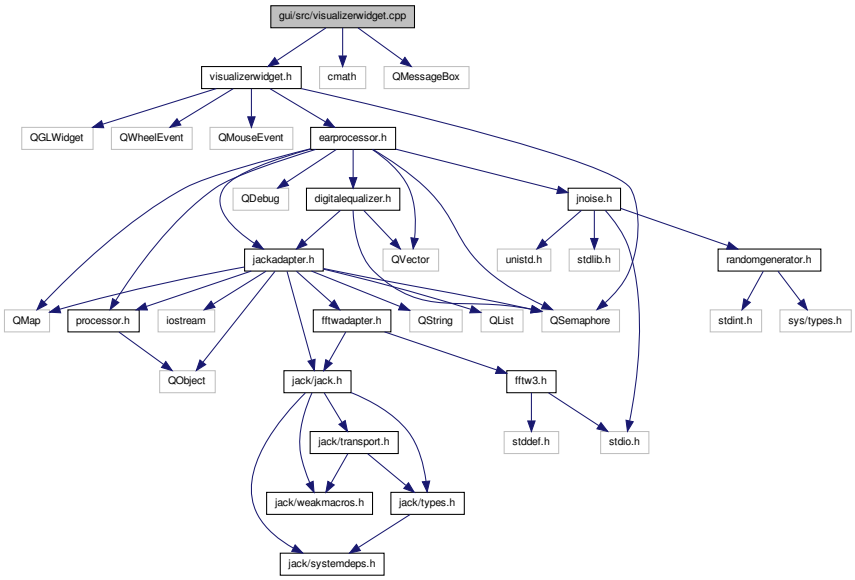
11.18.1.4 #define WHITE_NOISE_COMBO_TEXT "White Noise"

Definition at line 29 of file mainwindow.cpp.

Referenced by MainWindow::MainWindow().

11.19 gui/src/visualizerwidget.cpp File Reference

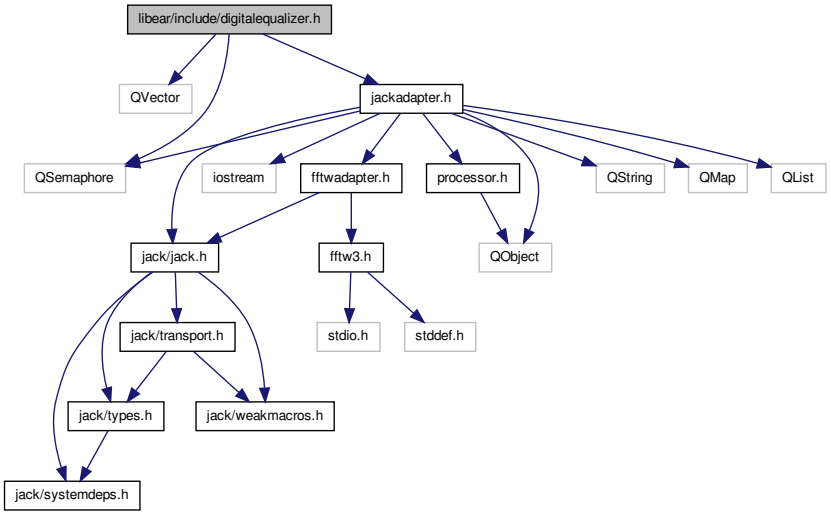
`#include "visualizerwidget.h" #include <cmath> #include <QMessageBox>` Include dependency graph for visualizerwidget.cpp:



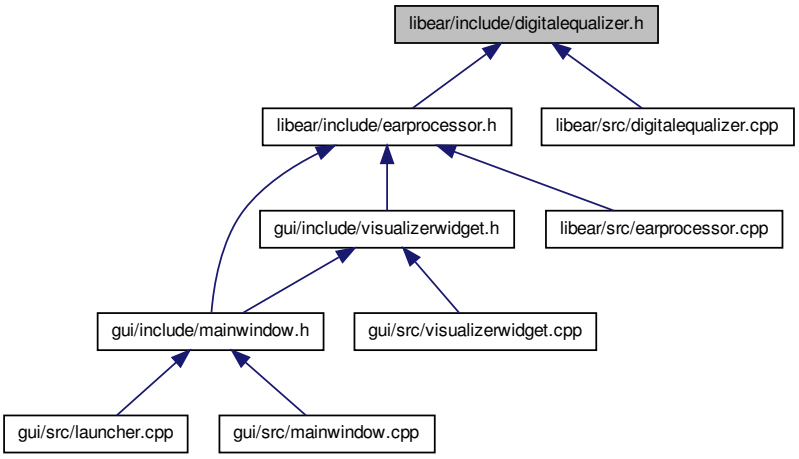
11.20 libear/include/digitalequalizer.h File Reference

`#include <QVector> #include <QSemaphore> #include "jackadapter.-`

h" Include dependency graph for digitalequalizer.h:



This graph shows which files directly or indirectly include this file:



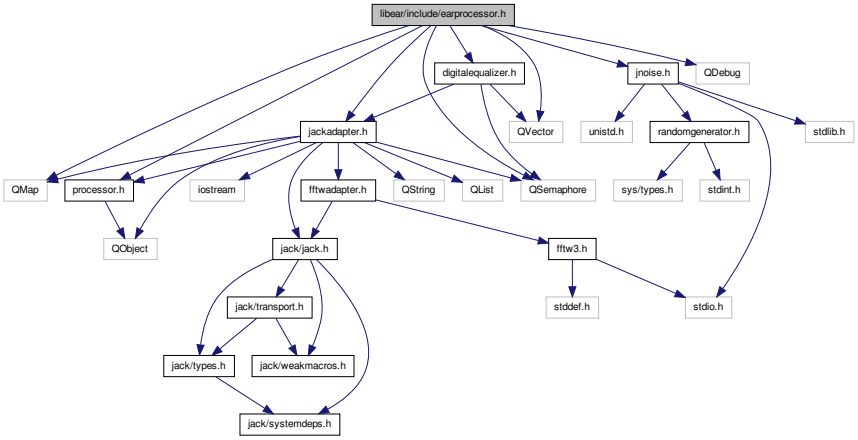
Data Structures

- class DigitalEqualizer

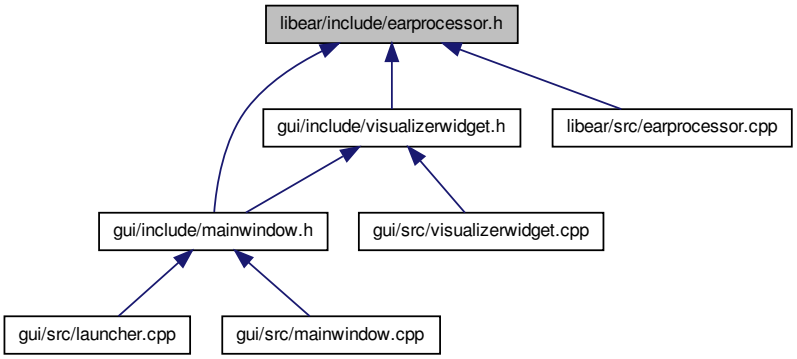
Modifies the frequency spectrum of the sampled audio signal.

11.21 libear/include/earprocessor.h File Reference

`#include "jackadapter.h" #include "processor.h" #include "digitalequalizer.h" #include "jnoise.h" #include <QVector> #include <QMap> #include <QDebug> #include <QSemaphore>` Include dependency graph for earprocessor.h:



This graph shows which files directly or indirectly include this file:



Data Structures

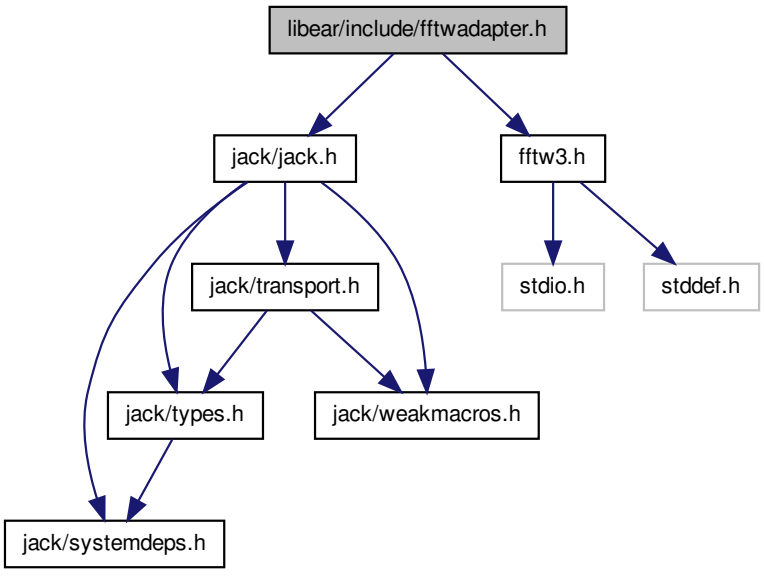
- class EarProcessor

  EarProcessor performs the central task of audio processing.

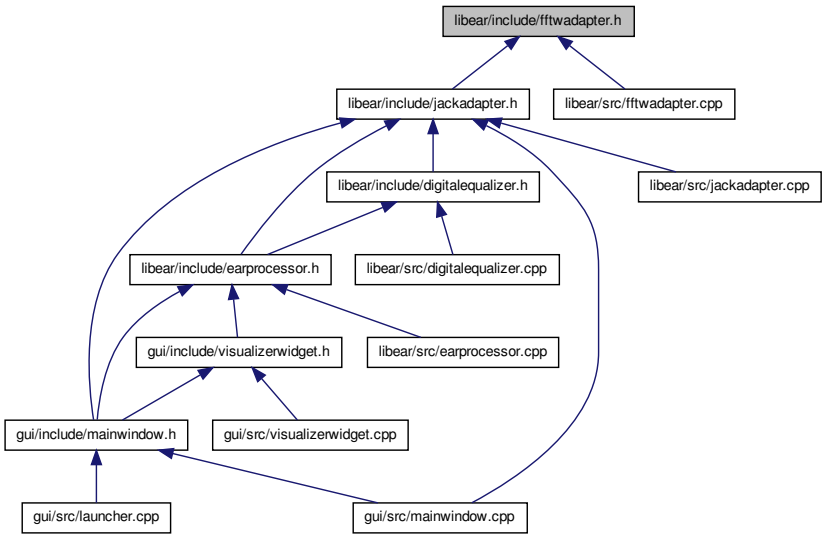
- struct **EarProcessor::Calibration**

11.22 libear/include/fftwadapter.h File Reference

`#include <jack/jack.h>` `#include "fftw3.h"` Include dependency graph for fftwadapter.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace FftwAdapter

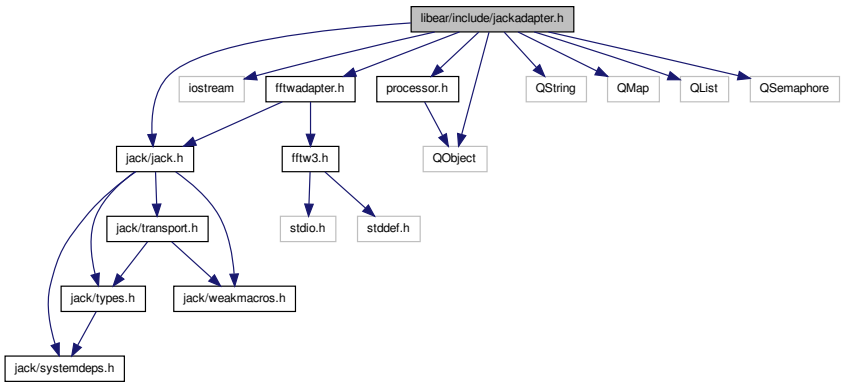
Functions

- void FftwAdapter::blit (fftw_complex *fftw_complexIn, jack_default_audio_sample_t *jack_default_audio_sample_tsOut, int n)
- void FftwAdapter::blit (jack_default_audio_sample_t *jack_default_audio_sample_tsIn, fftw_complex *fftw_complexOut, int n)
- void FftwAdapter::blit (jack_default_audio_sample_t *jack_default_audio_sample_tsIn, jack_default_audio_sample_t *jack_default_audio_sample_tsOut, int n)
- void FftwAdapter::blit (fftw_complex *fftw_complexIn, fftw_complex *fftw_complexOut, int n)
- void FftwAdapter::performFFT (fftw_complex *input, fftw_complex *result, int n)
- void FftwAdapter::performInverseFFT (fftw_complex *input, fftw_complex *result, int n)

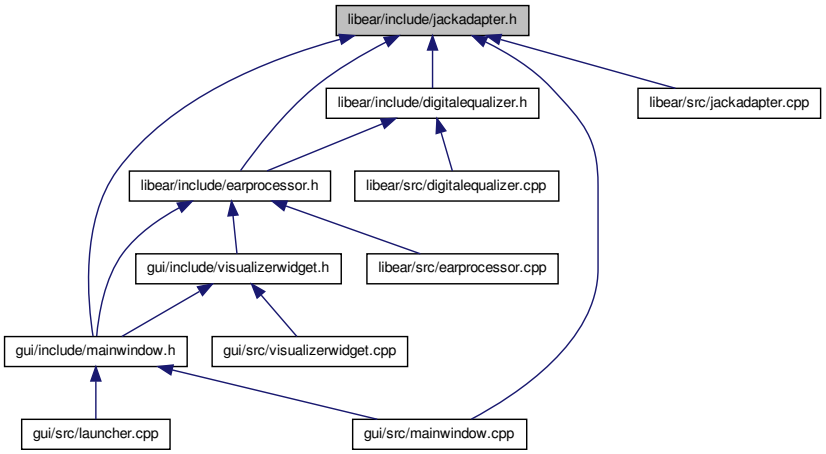
11.23 libear/include/jackadapter.h File Reference

```
#include <jack/jack.h> #include <iostream> #include "fftwadapter.-
```

h" #include "processor.h" #include <QObject> #include <-QString> #include <QMap> #include <QList> #include <Q-Semaphore> Include dependency graph for jackadapter.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- class SemaphoreLocker
- struct StereoPort
- class JackAdapter

  C++ Wrapper for the JACK Audio Connection Kit client API.

Typedefs

- typedef struct StereoPort StereoPort

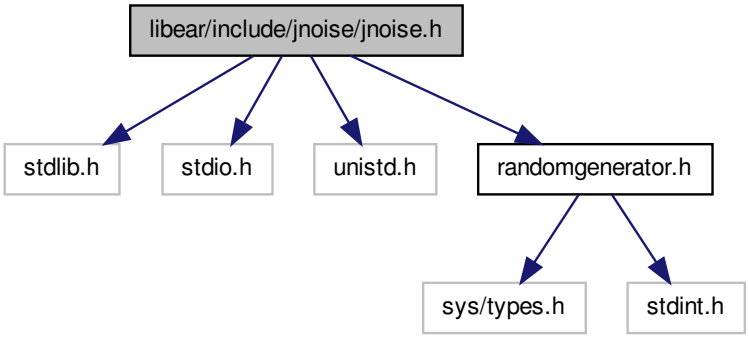
11.23.1 Typedef Documentation

11.23.1.1 typedef struct StereoPort StereoPort

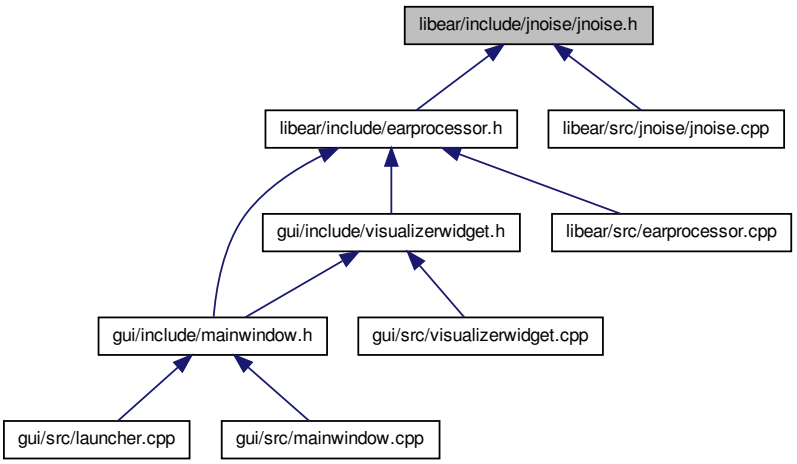
Defines a stereo port.

11.24 libear/include/jnoise/jnoise.h File Reference

`#include <stdlib.h>` `#include <stdio.h>` `#include <unistd.h>` `#include "randomgenerator.h"` Include dependency graph for jnoise.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- class JNoise

Defines

- #define LRAND 1024
- #define MRAND (LRAND - 1)

11.24.1 Define Documentation

11.24.1.1 #define LRAND 1024

Definition at line 28 of file jnoise.h.

Referenced by JNoise::JNoise().

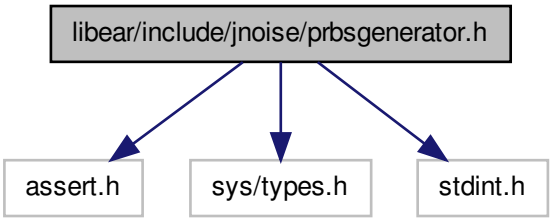
11.24.1.2 #define MRAND (LRAND - 1)

Definition at line 29 of file jnoise.h.

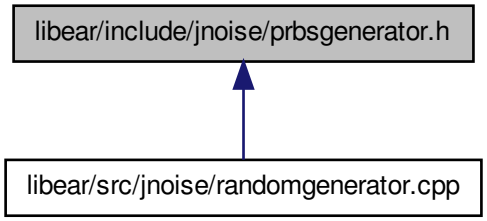
Referenced by JNoise::process().

11.25 libear/include/jnoise/prbsgenerator.h File Reference

`#include <assert.h>` `#include <sys/types.h>` `#include <stdint.-h>` Include dependency graph for prbsgenerator.h:



This graph shows which files directly or indirectly include this file:



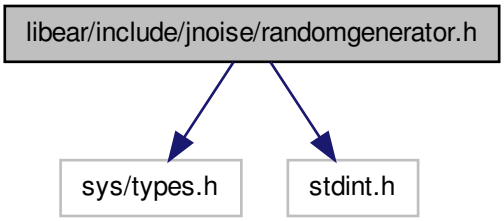
Data Structures

- class PRBSGenerator

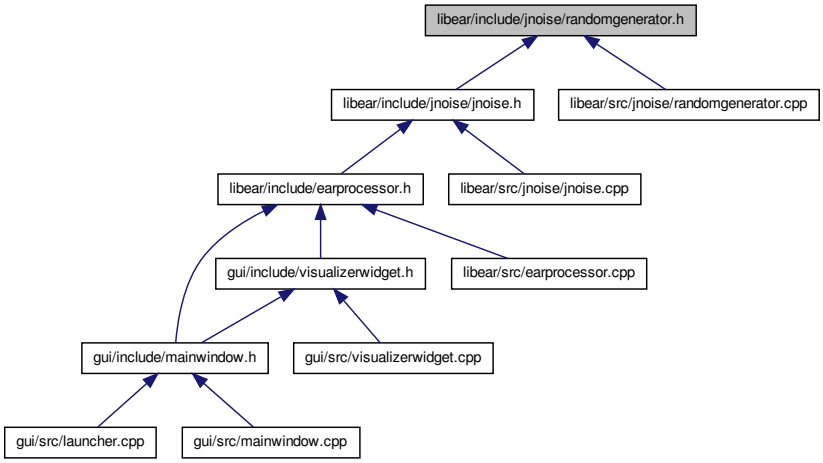
11.26 libear/include/jnoise/randomgenerator.h File Reference

`#include <sys/types.h>` `#include <stdint.h>` Include dependency

graph for randomgenerator.h:



This graph shows which files directly or indirectly include this file:

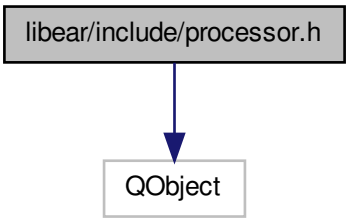


Data Structures

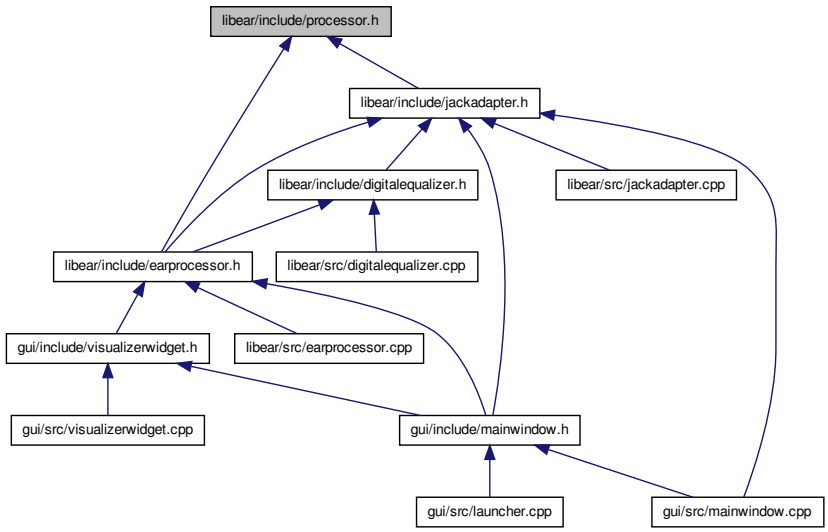
- class RandomGenerator

11.27 libear/include/processor.h File Reference

`#include <QObject>` Include dependency graph for processor.h:



This graph shows which files directly or indirectly include this file:



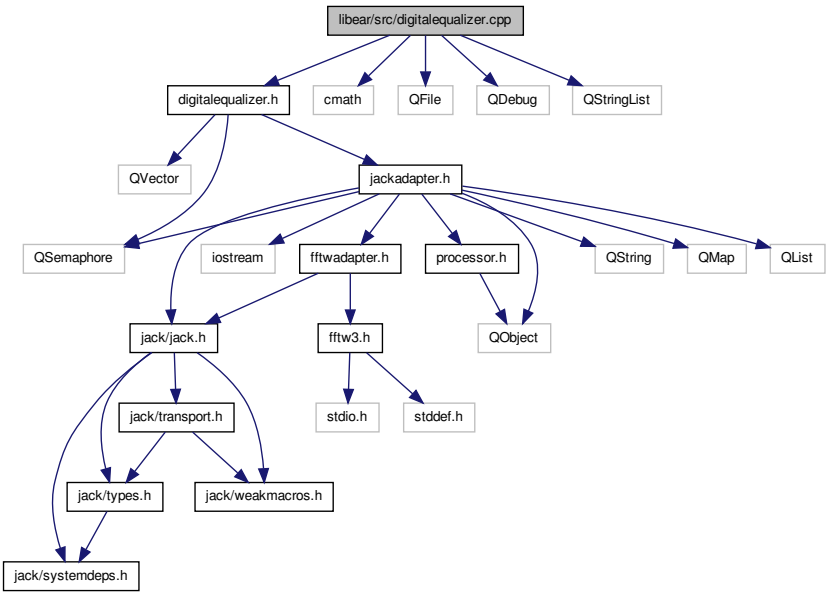
Data Structures

- class Processor

  Processor defines an interface that must be met by processors.

11.28 libear/src/digitalequalizer.cpp File Reference

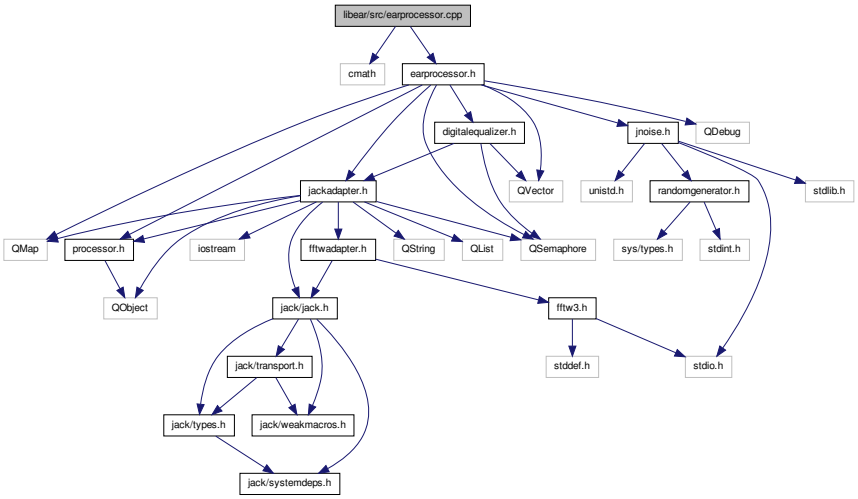
`#include "digitalequalizer.h" #include <cmath> #include <QFile> #include <QDebug> #include <QStringList>` Include dependency graph for digitalequalizer.cpp:



11.29 libear/src/earprocessor.cpp File Reference

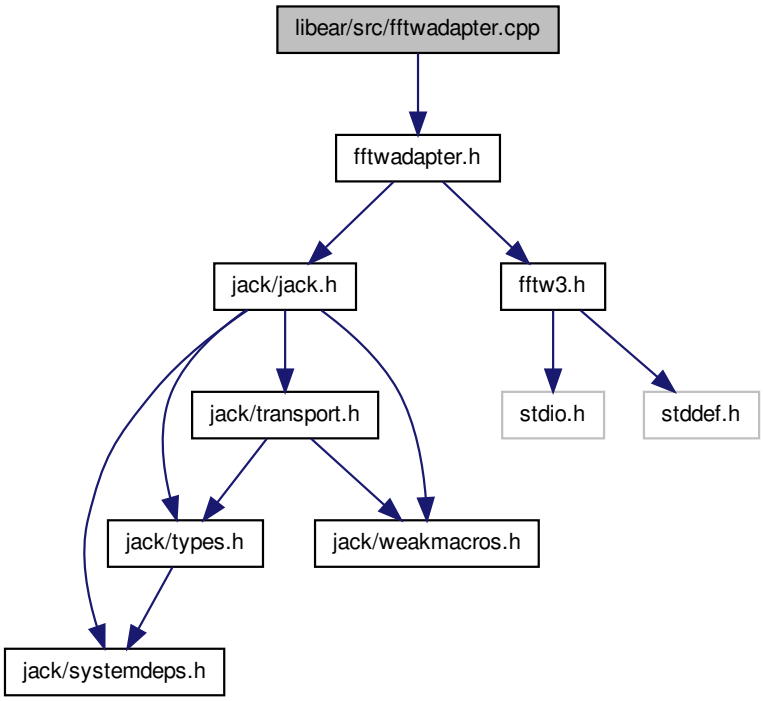
`#include <cmath> #include "earprocessor.h"` Include dependency

graph for earprocessor.cpp:



11.30 libear/src/fftwadapter.cpp File Reference

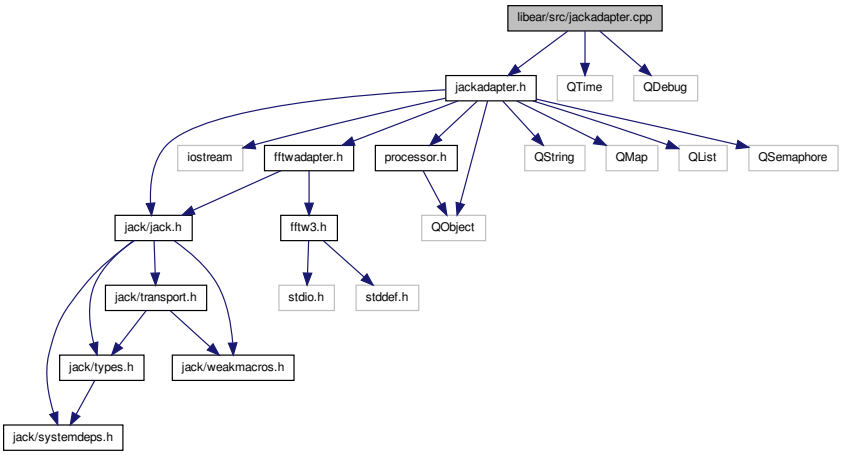
`#include "fftwadapter.h"` Include dependency graph for fftwadapter.cpp:



11.31 libear/src/jackadapter.cpp File Reference

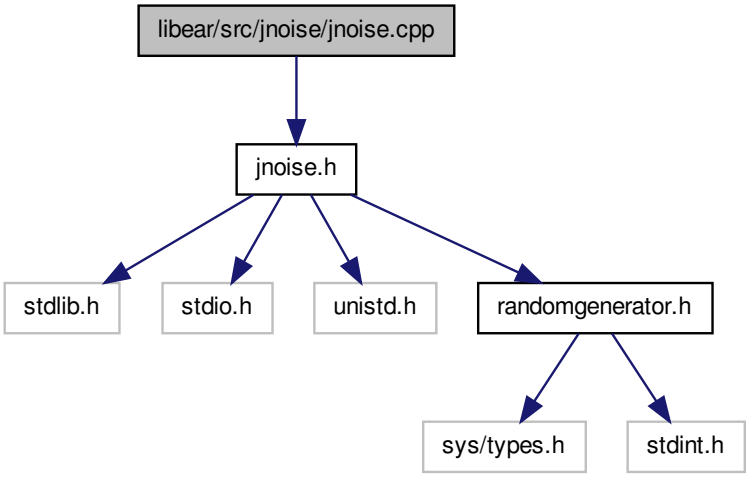
`#include "jackadapter.h"` `#include <QTime>` `#include <Q-`

Debug> Include dependency graph for jackadapter.cpp:



11.32 libear/src/jnoise/jnoise.cpp File Reference

#include "jnoise.h" Include dependency graph for jnoise.cpp:



11.33 libear/src/jnoise/randomgenerator.cpp File Reference

`#include <math.h>` `#include <time.h>` `#include "prbsgenerator.h"` `#include "randomgenerator.h"` Include dependency graph for randomgenerator.cpp: