LADCCA Reference Manual

Audio application session management and configuration
Edition 0.4.0, October 2003
Covering LADCCA 0.4.0

Bob Ham (rah@bash.sh)

Table of Contents

1 Introduction

LADCCA stands for Linux Audio Developer's Configuration and Connection API. It is a session management system for audio applications on GNU/Linux. It understands the JACK low latency audio API and the ALSA MIDI sequencer interface. The system is comprised of a server program, `ladccad`, an application library, `libladcca`, and a command line control program, `laddca_control`. The server and clients communicate over TCP sockets. There are three kinds of clients: normal clients (audio applications), user interfaces for the server, and connection patchbays.

1.1 Nomenclature

In order to describe the system, we should introduce some terminology. First of all, the *server* is the `ladccad` server program, an omni-present marshaller and database for storing arbitrary application data. The *library* is the '`libladcca`' shared library. It contains all the functions that an application uses to communicate with the server and take part in the system. Such an application is called a *client*.

The server deals with things in terms of collections of clients, called *projects*. A project has a unique string name, a current directory and a list of clients that are in that project. The server can have one client that is a *server interface* that allows the user to control the server. There is one server interface included with the system, the `ladcca_control` program.

2 Copying LADCCA

LADCCA is distributed under the GNU General Public License. A copy of the license text is provided in the file '`COPYING`' along with the software source code, or you can get a copy by writing to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

In plain english, the GPL basically restricts you from restricting other people's use of the LADCCA source code (ie, all of LADCCA), and any additions you make to the code, including linking with the LADCCA library. If you use code from this software, your software must be relased under the GPL. If you modify this software and release it, your modifications must be released under the GPL. If you release software linked against the LADCCA library, your software must be released under the GPL.

Note that this in no way restricts those people who want to release non-free LADCCA clients. LADCCA operates using a well defined protocol over TCP sockets. The high-level protocol is described within this document and the lower-level bit-wise protocol can be garnered from the source itself.

3 Installation

This chapter describes how to get LADCCA installed on your system.

3.1 Dependencies

LADCCA depends on the ALSA library, the JACK library, a unique ID library called libuuid and the XML library libxml2. You need these installed before attempting to install LADCCA.

ALSA is available from `http://www.alsa-project.org/`.

JACK is available from `http://jackit.sf.net/`.

The libuuid library is included with the ubiquitous e2fsprogs package, but if this is not installed on your system, it is available from `http://e2fsprogs.sf.net/`. The libxml2 library is available from `http://www.xmlsoft.org/`.

The LADCCA Control client depends on the GNU Readline library, available from `ftp://ftp.gnu.org/pub/gnu/readline/`.

The LADCCA GTK Test Client and the LADCCA Save Button 2 client both depend on the GTK+ 2 toolkit, available from `ftp://ftp.gtk.org/pub/gtk/v2.0/`. The LADCCA Synth client has an optional GUI which also uses the GTK+ 2 toolkit.

The LADCCA Save Button client depends on the GTK+ 1.2 toolkit, also available from `ftp://ftp.gtk.org/pub/gtk/v1.2/`.

3.2 Installation

First off, you need to download the package. It is available from the LADCCA webpage, `http://pkl.net/~node/ladcca.html`. After you have downloaded it, unpack the tarball into a directory using `tar xzf /where/ever/you/put/ladcca-0.4.0.tar.gz` and change into the source directory with `cd ladcca-0.4.0`.

The package uses the GNU autotools for configuration and makefile creation. In order to install the package, you must complete three steps: configuration; building; and installing.

3.2.1 Configuration

To configure the package, there is a shell script named '`configure`' in the top source directory. It is a standard GNU autoconf configure script, and so accepts the standard GNU configuration options (such as '`--prefix`', '`--datadir`', etc.) To run it type `./configure` and add any options. The non-standard options that the script recognises are described below. Running the script with the '`--help`' option will also provide a quick summary of the recognised options.

3.2.1.1 Configure script options

'`--disable-gtk2`'
: Prevents the configure script from checking for the presence of the GTK+ 2 toolkit and disables the building of any code that relies on it. The LADCCA Save Button 2 and LADCCA GTK Test Client both rely on GTK+ 2.

'--disable-gtk'
: Prevents the configure script from checking for the presence of the GTK+ 1.2 toolkit and disables the building of any code that relies on it. The LADCCA Save Button client relies on GTK+ 1.2.

'--enable-debug'
: Causes the library, server and clients to be built with debugging output. This is not very useful and only recommended for developers working on the LADCCA code itself.

'--with-default-dir=DIR'
: Specifies the default directory, relative to \$HOME under which the server will create new project directories. Without this option, the value defaults to 'audio-projects'.

'--disable-serv-inst'
: The LADCCA server and library look up port addresses for the LADCCA service using standard system calls that refer to the '/etc/services' database. If '/etc/services' does not contain a valid entry, an error will result. By default, the installation routine will install a service entry if one does not exist. This flag disables that action.

3.2.2 Building

To build the package, simply type `make` in the top source directory. This will build the server, the library and the clients that are compatible with the resources found by the configure script.

3.2.3 Installing

To install the package, type `make install` in the top source directory. By default the package installation prefix is '/usr/local' but the '--prefix' option to the configure script will change this. The `ladccad` server and the clients that were built are installed in '*prefix*/bin'. The `libladcca` client library is installed in '*prefix*/lib'. The C header files for the client library are installed under '*prefix*/include'. This manual is installed under '*prefix*/info'.

With `make install`, the package is installed with debugging symbols in the object files. To save space, you can install with `make install-strip` to install the object files without debugging symbols.

4 Server

The LADCCA server is called '`ladccad`'.

5 Server interface

Run the `ladccad_control`. There is a `help` command.

6 Client reference

This chapter provides a programming guide and library reference for programmers of LADCCA clients.

6.1 Operational overview

In this section we give an overview of how the LADCCA system operates, describing the server and client objects and operations that make it work. The `ladccad` server must be running in order for clients to participate in the system; clients cannot interoperate soley between themselves. The server maintains a list of connected clients and a list of projects with which these clients are associated.

The server and clients exchange events and configs over their connections. There is one, and only one, bi-directional connection between a client and the server. The transport for this connection is currently TCP.

An *event* is a very simple object having two relevant properties: a type and an optional arbitrary character string. The type defines what the event means to the recipient, and the string allows additional information to be included with it. For example, if a client wishes the server to save the current project, it sends a `CCA_Save` event to the server. While saving the project, the server may wish to tell a client to save its data in a certain directory. To so, it sends a `CCA_Save_File` event to the client with a string containing the name of a directory into which the client should save its data files.

Clients can save data on the server if they wish. To do this, the client declares that it wants to save data on the server when it initialises the server connection and then later sends one or more *config*s to the server. A config is also a very simple object. It has a client-unique character string key, and a value of arbitrary size and type (well, almost arbitrary; its size must be able to be described by a `uint32_t` integer due to byte-order conversions done when sending data over the network.)

6.1.1 Session example

In this section we will examine a typical session in some detail, describing the server and client operations that take place. In the session, the server is started, a number of clients connect, the session is saved and then restored.

6.1.1.1 Starting up the server

Before all else, the user starts the server. It starts up and begins listening for connections from clients. It doesn't do much else.

6.1.1.2 A client connection

The user then starts a JACK client program. It opens a connection to the server and provides it with all information that the server will need to run the application again. This information includes: the current directory that the user was in when they ran the program, the command line that started the application and the *class* of the client (a character string that the client application provides the initialisation routine that will never change over all initialisations.)

With this information is included a set of flags that describe the client to the server. This particular client saves data to files and wants the server to tell it where to save files when the project is saved, so it has the `CCA_Config_File` flag set.

The client library starts two threads for communication with the server, one for sending data and the other for recieving. It also sends, along with the client supplied data, a number of parameters that were extracted from the client's command line options before it checked them. This optionally includes the name of the project that the client should initially be associated with and a 128-bit, world-unique identifier for this particular client instance (the *LADCCCA ID*.)

Server-side, the server wakes up to the fact that a new connection has arrived and immediately adds it to a list of open connections and then goes back to waiting. When the client sends the requisite information, the server looks at it and decides what to do with the client. This client has not requested a specific project to which it should be connected. However, there are no existing projects so the server creates a new project with the name '`project-1`' in the directory '`/home/`*user*`/audio-projects/project-1`' (assuming the user didn't specify a different default directory when running configure.) It also generates a new LADCCA ID for the client. It then adds the client to the new project and goes back to listening.

The client then connects up to the JACK server and, after having done this, sends a `CCA_Jack_Client_Name` event to the server with the name that it registered to JACK with as the string. This notifies the server that it is a JACK client and needs its JACK port connections saved and restored. The server will now pay attention to any activity regarding the client (ie, port creation and destruction and port connection and disconnection.)

6.1.1.3 Another client

The user then starts a second client that uses the ALSA sequencer interface and wishes to save data on the server. It connects to the server with a different class to the JACK client and with the `CCA_Config_Data_Set` flag set.

The server sees that this client also didn't specify a project, and so adds it to the first available project; the same one as the previous project, '`project-1`'. It also sees that the client wants to store data on the server, and so it creates a directory within the project directory for this data to be stored in and creates a database-style object to manage the client's data.

The client then connects to the ALSA sequencer and sends its client ID to the server in the first character of the string of a `CCA_Alsa_Client_Name` event. The server regards this similarly to the other client's JACK client name.

6.1.1.4 Saving the project

After the user has done some work in the two clients, they want to save their work. They click a button on one of the clients (or something similar) and the client sends a `CCA_Save` event to server. The server recieves this and then iterates through each client in the project and checks its flags. The JACK client saves data by itself (it has the `CCA_Config_File` flag set,) so the server creates a directory under the project directory for it to save in and then sends a `CCA_Save_File` event to the client with a string containing the name of the directory it made. The client recieves the event and saves its data into the specified directory.

Next, the server examines the ALSA client. It wishes to save data on the server, so the server sends a `CCA_Save_Data_Set` to the client. With all of the clients iterated through, it now saves all the information it needs to be able to restore them; their working directory, command line options, etc. In order to do this, it asks the JACK server to find the connections for the JACK client, and asks the ALSA sequencer to find the connections for the ALSA client. It uses the client name and ID that both clients sent to the server after opening their connections to the respective systems. All of this information is stored in a file under the project's directory. When this is done, the server goes back to listening for events and configs.

The client, meanwhile, has recieved the `CCA_Save_Data_Set` event and sends back a number of configs to the server. When it has sent all the data it wishes to be saved, it sends back a `CCA_Save_Data_Set` event. The server passes all of the configs to the object managing the data store for the ALSA client. When the server recieves the `CCA_Save_Data_Set` event from the client, it tells the data store to write the data to disk. The save is now complete.

6.1.1.5 Client resumption

Unfortunately for the user, the ALSA client crashes. The server detects that the client has disconnected, and puts the client on a list of lost clients for the project. The user then starts another copy of the client, which connects to the server in the same way it did before. This time, however, the server checks through the list of lost clients and finds that the class of the new client matches the class of the lost client and so it resumes the lost client using the new one. It gives it the 128-bit ID of the lost client, adds it to the project, and then sends a `CCA_Restore_Data_Set` event to the client. The client then cleans itself up, ready to recieve the data set. The server sends the client the configs, and then another `CCA_Restore_Data_Set` event. The client recieves this data and its state has been restored that of the client that crashed.

The user can stop this behaviour by specifying the '`--ladcca-no-autoresume`' option on the client's command line.

6.1.1.6 Restoring the project

The user has to go off and do other things, and so they close down the clients and the server. Some time later, the user comes back and wants to start working again so first, as always, they start up the server. They then start the `ladcca_control` program. This is a text interface command program for controlling the server. They get a command prompt and into it type *`restore /home/user/audio-projects/project-1`*. The ladcca_control client sends a `CCA_Restore` event to the server with the specified directory as the string. The server opens the file that it saved before, and reads in all the information about the project and its clients. It creates a new project with this information. The clients are created as lost clients, however.

The server then iterates through each client and starts a new copy of it using the information provided when the original client connected. It also adds some command line options that are extracted by the client library. These specify the LADCCA ID of the client, the project name that it should be connecting to and the server's hostname and port. It then goes back to waiting.

The new JACK client then connects to the server as normal. When the server recieves it connection, it checks the client against the project's list of lost clients. This time, however, it has its ID specified, so the server will only resume a client with a matching ID. Lo and behold, such a client exists. The server resumes the old JACK client, telling it to load its state from the files in the project directory that the client previously stored. It does so with a `CCA_Restore_File` event with the string as the directory name. The ALSA client does exactly the same, except having its data restored through `CCA_Restore_Data_Set` as described above.

Only one thing remains for the clients to be fully restored: the JACK and ALSA sequencer connections. This happens when the clients send their `CCA_Jack_Client_Name` and `CCA_Alsa_Client_ID` events. The connections are stored with the LADCCA ID rather than the JACK client name or ALSA client ID. When the client registers its name or ID, the connections are converted from the LADCCA ID to the JACK client name or ALSA client ID, and the connections are restored. It also pays attention to connections to other clients within the same project, converting between JACK client names, ALSA client IDs and LADCCA IDs as appropriate.

6.2 Types and functions

6.2.1 Server interaction

`cca_client_t *` **cca_init** (`cca_args_t *` *args*, `const char *` *client_class*, `int` *client_flags*, `cca_protocol_t` *protocol*) Function

Open a connection to the server. Returns `NULL` on failure.

The *args* argument must be obtained using `cca_extract_args`.

The *client_class* argument must be a string that will never change over invocations of the program. If using GNU automake, the best way to do this is to use the `PACKAGE_NAME` macro that is automatically defined.

The *client_flags* argument should be 0 or bitwise-OR'd values from this list:

`CCA_Config_Data_Set`
: The client wishes to save its data use the LADCCA config system. See [Configs], page 13 and Section 6.3 [Event protocol], page 14.

`CCA_Config_File`
: The client saves its data to a file. See Section 6.3 [Event protocol], page 14.

`CCA_Server_Interface`
: The client is a server interface. See Section 6.3.2 [Server interfaces], page 17.

`CCA_No_Autoresume`
: This flag is set by the `--ladcca-no-autoresume` command line option and should not normally be set by clients themselves.

`CCA_Terminal`
: The client is dependant on being run in a terminal.

The *protocol* argument should be the version of the high-level protocol that the client implements See [Protocol versioning], page 12 for information on how to contruct a `cca_protocol_t` variable.

`cca_args_t *` **cca_extract_args** (`int *` *argc*, `char ***` *argv*) — Function

Extract LADCCA-specific arguments from argc/argv for use in `cca_init`. This should be done before the client checks the arguments, obviously.

`const char *` **cca_get_server_name** (`cca_client_t *` *client*) — Function

Get the hostname of the server.

`unsigned int` **cca_get_pending_event_count** (`cca_client_t *` *client*) — Function

Get the number of pending events.

`cca_event_t *` **cca_get_event** (`cca_client_t *` *client*) — Function

Retrieve an event. The event must be freed using `cca_event_destroy`. Returns `NULL` if there are no events pending.

`unsigned int` **cca_get_pending_config_count** (`cca_client_t *` *client*) — Function

Get the number of pending configs.

`cca_config_t *` **cca_get_config** (`cca_client_t *` *client*) — Function

Retrieve a config. The config must be freed using `cca_config_destroy`. Returns `NULL` if there are no configs pending.

`void` **cca_send_event** (`cca_client_t *` *client*, `cca_event_t *` *event*) — Function

Send an event to the server. The event must be created using `cca_event_new` or `cca_event_new_with_type`. The library takes over ownership of the memory and it should not be freed by the client.

`void` **cca_send_config** (`cca_client_t *` *client*, `cca_config_t *` *config*) — Function

Send some configuration data to the server. The config must be created using `cca_config_new`, `cca_config_new_with_key` or `cca_config_dup`. The library takes over ownership of the memory and it should not be freed by the client.

cca_enabled (*client*) — Macro

Check whether the cca_client_t pointer *client* is not `NULL`, and if it isn't, that the server is still connected.

`int` **cca_server_connected** (`cca_client_t *` *client*) Function

Check whether the server is connected. Returns 1 if the server is still connected or 0 if it isn't

`void` **cca_jack_client_name** (`cca_client_t *` *client*, `const char *` *name*) Function

Tell the server the client's JACK client name. This is a convenience function that just sends a CCA_Jack_Client_Name event to the server. See [Normal CCA_Jack_Client_Name], page 14.

`void` **cca_alsa_client_id** (`cca_client_t *` *client*, `unsigned char` *id*); Function

Tell the server the client's ALSA client ID. This just is a convenience function that just sends a CCA_Alsa_Client_ID event to the server. See [Normal CCA_Alsa_Client_ID], page 15.

6.2.2 Protocol versioning

The event protocol (See Section 6.3 [Event protocol], page 14,) is versioned with a major and minor component. The `cca_protocol_t` type represents a version number in a 32-bit unsigned integer split 16:16. A protocol is comptible with the server's protocol if the major numbers are the same and the minor number is less than, or equal to, the server's minor number (ie, 1.0 is compatible with a server using 1.0, 1.1 is compatible with a server using 1.3, but neither 2.0 or 1.6 are compatible with a server using 1.4. The minor component may be dropped in the future.

CCA_PROTOCOL (*major*, *minor*) Macro

Contruct a protocol version with a major component *major* and a minor component *minor*.

CCA_PROTOCOL_GET_MAJOR (*protocol*) Macro

Obtain the major component of a `cca_protocol_t` protocol version.

CCA_PROTOCOL_GET_MINOR (*protocol*) Macro

Obtain the minor component of a `cca_protocol_t` protocol version.

`const` **char** `* cca_protocol_string` (`cca_protocol_t` *protocol*) Function

Obtain a string representation of the protocol version *protocol*. String representations are of the form "*major.minor*".

6.2.3 Events

`cca_event_t *` **cca_event_new** (*void*) Function

`cca_event_t *` **cca_event_new_with_type** (`enum CCA_Event_Type` *type*) Function void **cca_event_destroy** (cca_event_t * *event*) Function

enum CCA_Event_Type **cca_event_get_type** (const cca_event_t * *event*) Function

const char * **cca_event_get_string** (const cca_event_t * *event*) Function

void **cca_event_set_type** (cca_event_t * *event*, enum CCA_Event_Type *type*) Function

void **cca_event_set_string** (cca_event_t * *event*, const char * *string*); Function

6.2.3.1 Server interface events

All events have a LADCCA ID and project name property. They are only relevant to server interfaces, however, which need to refer to clients other than themselves and to projects (server interfaces are never assigned to a project.)

void **cca_event_get_client_id** (const cca_event_t * *event*, uuid_t *id*) Function

The event's client ID property will be copied into *id*.

const char * **cca_event_get_string** (const cca_event_t * *event*) Function

void **cca_event_set_client_id** (cca_event_t * *event*, enum uuid_t *id*) Function

void **cca_event_set_project** (cca_event_t * *event*, const char * *project_name*); Function

6.2.4 Configs

cca_config_t * **cca_config_new** (*void*) Function

cca_config_t * **cca_config_dup** (const cca_config_t * *config*) Function

cca_config_t * **cca_config_new_with_key** (const char * *key*) Function

void **cca_config_destroy** (cca_config_t * *config*) Function

const char * **cca_config_get_key** (const cca_config_t * *config*) Function

const void * **cca_config_get_value** (const cca_config_t * *config*) Function

size_t **cca_config_get_value_size** (const cca_config_t * *config*) Function

void **cca_config_set_key** (cca_config_t * *config*, const char * *key*) Function

void **cca_config_set_value** (cca_config_t * *config*, const void * *value*, size_t *value_size*) Function

6.2.4.1 Semi-typed configs

With these functions, no type checking is done; you can do `cca_config_get_value_int` on a config that was set with `cca_config_set_value_float`. The integer values are converted to and from network byte order as appropriate.

`uint32_t` **cca_config_get_value_int** (`const cca_config_t *` *config*) Function

`float` **cca_config_get_value_float** (`const cca_config_t *` *config*) Function

`double` **cca_config_get_value_double** (`const cca_config_t *` *config*) Function

`const char *` **cca_config_get_value_string** (`const cca_config_t *` *config*) Function

`void` **cca_config_set_value_int** (`cca_config_t *` *config*, `uint32_t` *value*) Function

`void` **cca_config_set_value_float** (`cca_config_t *` *config*, `float` *value*) Function

`void` **cca_config_set_value_double** (`cca_config_t *` *config*, `double` *value*) Function

`void` **cca_config_set_value_string** (`cca_config_t *` *config*, `const char *` *value*) Function

6.3 Event protocol

This section describes version 2.0 of the event protocol.

6.3.1 Normal clients

This section deals with normal clients (as opposed to Section 6.3.2 [Server interfaces], page 17.)

`CCA_Client_Name`

To server, non-`NULL` *string*

Set the client's user-visible name.

To server, `NULL` *string*

Request the client's user-visible name.

From server

This will only be sent in response to a `CCA_Client_Name` with a `NULL` string. The string will be `NULL` if the client has not set a user-visible name, and the name itself if it has.

`CCA_Jack_Client_Name`

To server, non-`NULL` *string*
: Tell the server what name the client is connected to JACK with. Clients should only ever send one non-`NULL` `CCA_Jack_Client_Name` event. Note that you *must* send this event *after* calling `jack_activate()`; otherwise, the server will not be able to connect the client's ports.

To server, `NULL` *string*
: Request the client name that the server thinks the client is connected to JACK with.

From server
: This will only be sent in response to a `CCA_Jack_Client_Name` with a `NULL` string. The string will be `NULL` if the client has not set a JACK client name, and the client name itself if it has.

`CCA_Alsa_Client_ID`

To communicate ALSA client IDs within events, use the first character of a two character string of the form `{` *id*`, '\0' }` as the event string. A convenience function, `cca_alsa_client_id`, exists to do this for you (see [cca_alsa_client_id], page 12.)

To server, non-`NULL` *string*
: Tell the server what ID the client is connected to ALSA with. Clients should only ever send one non-`NULL` `CCA_Alsa_Client_ID` event.

To server, `NULL` *string*
: Request the client ID that the server thinks the client is connected to ALSA with.

From server
: This will only be sent in response to a `CCA_Alsa_Client_ID` with a `NULL` string. The string will be `NULL` if the client has not set an ALSA client ID, and a string containing the ALSA client ID as described above if it has.

`CCA_Save_File`

From server
: Tell the client to save all its data to files within a specific directory. The event string will never be `NULL` and will contain the name of the directory in which the client should save its data. Clients must always send a `CCA_Save_File` event back to the server when they have finished saving their data. The client should not rely on the directory existing after it has sent its `CCA_Save_File` event back. It is valid behaviour for a client to save no files within the directory. Files should always be overwritten (ie, using the `"w"` flag with `fopen()`,) preferably without user confirmation if you care for their sanity.

From client
: Tell the server that the client has finished saving its data within the directory it was told to. The string is ignored.

`CCA_Restore_File`

From server
: Tell the client to load all its data from files within a specific directory. The event string will never be `NULL` and will contain the name of the directory from which the client should load its data. Clients must always send a `CCA_Restore_File` event back to the server when they have finished restoring their data. The client should not rely on the directory existing after it has sent its `CCA_Restore_File` event back.

From client
: Tell the server that the client has finished restoring its data from within the directory it was told to. The string is ignored.

`CCA_Save_Data_Set`

From server
: Tell the client to send all its configuration data to the server with a number of configs. The client must always send a `CCA_Save_Data_Set` event back to the server when it has finished sending its configs. The event string will always be `NULL`.

From client
: Tell the server that the client has finished sending its configs to the server. The event string is ignored.

`CCA_Restore_Data_Set`

From server
: Tell the client to immediately expect a stream of configs from the server. This event will only be sent if there are one or more configs to be sent. The event string will always be `NULL`. The client must always send a `CCA_Restore_Data_Set` back to the server when it has recieved all of its configs.

From client
: Tell the server that the client has finished recieving its configs from the server. The event string is ignored.

`CCA_Save`

From client
: Tell the server to save the project that the client is attached to.

From server
: Never occurs.

`CCA_Quit`

From client
: Tell the server to close all clients in the project that the client is attached to.

From server

The client should immediately quit without saving. No more events will be sent by the server and the client's connection will be terminated.

6.3.2 Server interfaces

Server interfaces are treated very differently to normal interfaces. Events from and to server interfaces are, for the most part, in order to describe and manipulate existing projects and clients. For this reason, the `cca_event_t` type has `project` and `client_id` properties which facilitate this. See [Server interface events], page 13. The `project` property contains the name of the project.

A server interface should start up with the default assumption that there are no projects. Upon connection, the server will send appropriate events (`CCA_Project_Add`, `CCA_Client_Add`, `CCA_Client_Name`, etc) that describe the current state of the system. From then on, events will be sent to keep the interface up to date with the server's state.

`CCA_Project_Add`

From interface

Restore a project from an existing directory. The event string should contain the directory's name.

`project` Ignored.

`client_id`

Ignored.

From server

A new project has been added. The event string will contain the project's name.

`project` `NULL`

`client_id`

Undefined.

`CCA_Project_Remove`

From interface

Close an open project. All of the project's clients will be told to quit and the project will be removed from the server's project list.

`project` The project to remove.

`client_id`

Ignored.

From server

A project has been removed.

`project` The project that has been removed.

`client_id`

Undefined.

`CCA_Project_Dir`

From interface, non-`NULL` *string*

Move a project to a different directory. The directory name should be contained in the event's string.

`project` The project to move.

`client_id`
Undefined.

From interface, `NULL` *string*

Request a project's directory.

`project` The project whose directory is being requested.

`client_id`
Undefined.

From server

A declaration of the project's directory; either because it has been requested or because the project has been moved. The directory name is contained in the event's string.

`project` The project whose directory is being declared.

`client ID` Undefined.

`CCA_Project_Name`

From interface

Change a project's name. The new project name should be contained in the event's string.

`project` The project name to change.

`client_id`
Undefined.

From server

A project's name has changed. The new project name is contained in the event's string.

`project` The project name that has changed.

`client ID` Undefined.

`CCA_Client_Add`

From interface

Should not occur

From server

A new client has been added.

`project` The project that the new client has been added to.

`client ID` The new client's ID.

`CCA_Client_Name`

From interface, non-`NULL` *string*

Should not occur.

From interface, `NULL` *string*

Request a client's name.

`project` The client's project.

`client ID` The client's ID.

From server

A declaration of a client's name; either because it has been requested or because the client set the name. The name is contained in the event's string.

`project` The client's project.

`client ID` The client's ID.

`CCA_Jack_Client_Name`

From interface, non-`NULL` *string*

Should not occur.

From interface, `NULL` *string*

Request a client's JACK client name.

`project` The client's project.

`client ID` The client's ID.

From server

A declaration of a client's JACK client name; either because it has been requested or because the client set the name. The client name is contained in the event's string.

`project` The client's project.

`client ID` The client's ID.

`CCA_Alsa_Client_ID`

From interface, non-`NULL` *string*

Should not occur.

From interface, `NULL` *string*

Request a client's ALSA client ID.

`project` The client's project.

`client ID` The client's LADCCA ID.

From server

A declaration of a client's ALSA client ID; either because it has been requested or because the client set the ID. The ALSA client ID is contained in the event's string, as desribed in [Normal CCA_Alsa_Client_ID], page 15.

`project` The client's project.

`client ID` The client's LADCCA ID.

`CCA_Percentage`

This event exists to provide user feedback on the status of save operations and perhaps other operations in future. The server will first send a percentage of 0, then successive percentages up to and including 100. When the operation is complete, the server will send a percentage of 0 again.

From interface

Should not occur.

From server

The percentage of completion of the current operation. The percentage is sent as a string, derived from `sprintf`ing an `int`.

`project` The project whose operation is being described.

`client ID` Undefined.

Appendix A Copying restrictions

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.