RhythmCat Music Player Development Reference Manual

COLLABORATORS

	TITLE : RhythmCat Music Player Development Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 2, 2011	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Part I

RhythmCat Music Player Reference

Chapter 1

Player Backend

1.1 Player

Player — Manage the player information, and initialize the player.

Synopsis

```
#include <player.h>

#define             GETTEXT_PACKAGE
void                rc_player_init                      (int *argc,
                                                         char **argv[]);
void                rc_player_main                      ();
void                rc_player_exit                      ();
const gchar *       rc_player_get_program_name          ();
const gchar *const * rc_player_get_authors              ();
const gchar *const * rc_player_get_documenters          ();
const gchar *const * rc_player_get_artists              ();
const gchar *       rc_player_get_build_date            ();
const gchar *       rc_player_get_version               ();
gboolean            rc_player_get_stable_flag           ();
const gchar *       rc_player_get_conf_dir              ();
const gchar *       rc_player_get_data_dir              ();
const gchar *       rc_player_get_home_dir              ();
const gchar *       rc_player_get_locale                ();
gboolean            rc_player_check_supported_format    (const gchar *filename);
```

Description

Manage the player information, and initialize the player.

Details

GETTEXT_PACKAGE

```
#define GETTEXT_PACKAGE "RhythmCat"
```

rc_player_init ()

```
void                 rc_player_init                      (int *argc,
                                                          char **argv[]);
```

Initialize the player.

***argc* :** address of the argc parameter of your main() function

***argv* :** address of the argv parameter of main()

rc_player_main ()

```
void                 rc_player_main                      ();
```

Runs the main loop until rc_player_exit() is called.

rc_player_exit ()

```
void                 rc_player_exit                      ();
```

Exit the player.

rc_player_get_program_name ()

```
const gchar *        rc_player_get_program_name          ();
```

Return the name of the program.

***Returns* :** The program name of the player, cannot be changed.

rc_player_get_authors ()

```
const gchar *const * rc_player_get_authors               ();
```

Return the author information.

***Returns* :** The string array of author information, cannot be changed.

rc_player_get_documenters ()

```
const gchar *const * rc_player_get_documenters           ();
```

Return the documenter information.

***Returns* :** The string array of documenter information, cannot be changed.

rc_player_get_artists ()

```
const gchar *const * rc_player_get_artists               ();
```

Return the artist information.

***Returns* :** The string array of artist information, cannot be changed.

rc_player_get_build_date ()

```
const gchar *       rc_player_get_build_date            ();
```

Return the build date.

***Returns* :** The build date, cannot be changed.

rc_player_get_version ()

```
const gchar *       rc_player_get_version               ();
```

Return the version information.

***Returns* :** The version information, cannot be changed.

rc_player_get_stable_flag ()

```
gboolean            rc_player_get_stable_flag           ();
```

Return the stable flag.

***Returns* :** Whether the player is in a stable version.

rc_player_get_conf_dir ()

```
const gchar *       rc_player_get_conf_dir              ();
```

Return the configure directory (~/.RhythmCat).

***Returns* :** The path of the configure directory.

rc_player_get_data_dir ()

```
const gchar *       rc_player_get_data_dir              ();
```

Return the program data directory.

***Returns* :** The program data directory.

rc_player_get_home_dir ()

```
const gchar *       rc_player_get_home_dir              ();
```

Return the user home directory.

***Returns* :** The user home directory.

rc_player_get_locale ()

```
const gchar *       rc_player_get_locale                ();
```

Return the locale information (e.g en_US).

***Returns* :** The locale information.

rc_player_check_supported_format ()

```
gboolean            rc_player_check_supported_format    (const gchar *filename);
```

Check whether the given file is supported by the player.

***`filename`* :** the filename to check

***Returns* :** Whether the file is supported.

1.2 Core

Core — The core of the player.

Synopsis

```
#include <core.h>

                    RCCoreData;
void                rc_core_init                        ();
void                rc_core_exit                        ();
RCCoreData *        rc_core_get_data                    ();
void                rc_core_set_uri                     (const gchar *uri);
gchar *             rc_core_get_uri                     ();
gboolean            rc_core_play                        ();
gboolean            rc_core_pause                       ();
gboolean            rc_core_stop                        ();
gboolean            rc_core_set_volume                  (gdouble volume);
gboolean            rc_core_set_play_position           (gint64 time);
gboolean            rc_core_set_play_position_by_persent
                                                        (gdouble persent);
gint64              rc_core_get_play_position           ();
gint64              rc_core_get_music_length            ();
gdouble             rc_core_get_volume                  ();
void                rc_core_set_eq_effect               (gdouble *fq);
GstState            rc_core_get_play_state              ();
```

Description

The core part of the player, it uses Gstreamer as backend to play audio files.

Details

RCCoreData

```
typedef struct {
    GstElement *playbin;
    GstElement *audio_sink;
    GstElement *eq_plugin;
    GstElement *vol_plugin;
    guint ver_major;
    guint ver_minor;
    guint ver_micro;
    guint ver_nano;
} RCCoreData;
```

The data of the core.

GstElement ****playbin***; the playbin element

GstElement ****audio_sink***; the audio sink element

GstElement ****eq_plugin***; the equalizer element

GstElement ****vol_plugin***; the volume control element

guint ***ver_major***; the major version number of Gstreamer

guint ***ver_minor***; the minor version number of Gstreamer

guint ***ver_micro***; the micro version number of Gstreamer

guint ***ver_nano***; the nano version number of Gstreamer

rc_core_init ()

```
void                    rc_core_init                        ();
```

Initialize the core of the player. Can be used only once.

rc_core_exit ()

```
void                    rc_core_exit                        ();
```

Free the core when exits.

rc_core_get_data ()

```
RCCoreData *            rc_core_get_data                    ();
```

Return the pointer of the core.

Returns **:** The pointer to the data structure of the core.

rc_core_set_uri ()

```
void                rc_core_set_uri                     (const gchar *uri);
```

Set the URI to play.

uri **:** the URI to play

rc_core_get_uri ()

```
gchar *             rc_core_get_uri                     ();
```

Return the URI the core opened.

Returns **:** The URI the core opened, free after usage.

rc_core_play ()

```
gboolean            rc_core_play                        ();
```

Set the state of the core to playing.

Returns **:** Whether the state is set to playing successfully.

rc_core_pause ()

```
gboolean            rc_core_pause                       ();
```

Set the core to pause state.

Returns **:** Whether the state is set to paused successfully.

rc_core_stop ()

```
gboolean            rc_core_stop                        ();
```

Set the core to stop state.

rc_core_set_volume ()

```
gboolean            rc_core_set_volume                  (gdouble volume);
```

Set the volume of player.

volume **:** the volume of the player, it should be between 0.0 and 100.0.

rc_core_set_play_position ()

```
gboolean            rc_core_set_play_position           (gint64 time);
```

Set the position to go to (in nanosecond). Notice that this function can only be used when the state of the player is playing or paused.

time **:** the position to go to

Returns **:** Whether the time is valid.

rc_core_set_play_position_by_persent ()

```
gboolean            rc_core_set_play_position_by_persent
                                                    (gdouble persent);
```

Set the position to to go to in persent (0.0 - 1.0).

`persent` **:** the position (in persent, from 0.0 to 1.0) to go to

Returns **:** Whether the persent is valid.

rc_core_get_play_position ()

```
gint64              rc_core_get_play_position           ();
```

Return the playing position (in nanosecond).

Returns **:** The playing position (in nanosecond).

rc_core_get_music_length ()

```
gint64              rc_core_get_music_length            ();
```

Return the time length of the playing music (in nanosecond).

Returns **:** The time length of the playing music (in nanosecond).

rc_core_get_volume ()

```
gdouble             rc_core_get_volume                  ();
```

Return the volume of the player.

Returns **:** The volume of the player.

rc_core_set_eq_effect ()

```
void                rc_core_set_eq_effect               (gdouble *fq);
```

Set the EQ effect of the player.

`fq` **:** an array (10 elements) of the gain for the frequency bands

rc_core_get_play_state ()

```
GstState            rc_core_get_play_state              ();
```

Return the state of the core.

Returns **:** The state of the core.

1.3 Tag

Tag — Process the tags of the music.

Synopsis

```
#include <tag.h>

                          RCMusicMetaData;
RCMusicMetaData *         rc_tag_read_metadata              (const gchar *uri);
void                      rc_tag_free                       (RCMusicMetaData *mmd);
void                      rc_tag_set_playing_metadata       (const RCMusicMetaData *mmd);
const RCMusicMetaData *   rc_tag_get_playing_metadata       ();
gchar *                   rc_tag_get_name_from_fpath        (const gchar *filename);
gchar *                   rc_tag_find_file                  (const gchar *dirname,
                                                             const gchar *str,
                                                             const gchar *extname);
```

Description

Process the tags of the music, like metadata, etc.

Details

RCMusicMetaData

```
typedef struct {
    gint64 length;
    gchar *uri;
    guint tracknum;
    guint bitrate;
    gint samplerate;
    gint channels;
    gint eos;
    gint list2_index;
    gchar *title;
    gchar *artist;
    gchar *album;
    gchar *comment;
    gchar *file_type;
    GstBuffer *image;
    gboolean audio_flag;
    gboolean video_flag;
    GtkTreeRowReference *reference;
    GtkListStore *store;
    gpointer user_data;
} RCMusicMetaData;
```

Custom struct type to store the music metadata.

gint64 ***length***; the length of the music

gchar ****uri***; the URI of the music

guint ***tracknum***; the track number of the music

guint ***bitrate*****;** the bitrate of the music

gint ***samplerate*****;** the sample rate of the music

gint ***channels*****;** the channel number of the music

gint ***eos*****;** the EOS signal

gint ***list2_index*****;** the insert index in list2, only used in insert operation

gchar ****title*****;** the title text of the music

gchar ****artist*****;** the artist text of the music

gchar ****album*****;** the album text of the music

gchar ****comment*****;** the comment text of the music

gchar ****file_type*****;** the file type of the music

GstBuffer ****image*****;** the GstBuffer which contains the cover image

gboolean ***audio_flag*****;** whether this file has audio

gboolean ***video_flag*****;** whether this file has video

GtkTreeRowReference ****reference*****;** the GtkTreeRowReference, used in list2 refresh operation

GtkListStore ****store*****;** the GtkListStore, used in list2 refresh operation

gpointer ***user_data*****;** the user data

rc_tag_read_metadata ()

```
RCMusicMetaData *   rc_tag_read_metadata                (const gchar *uri);
```

Read tag (metadata) from given URI.

uri **:** the URI of the music file

Returns **:** The Metadata of the music, NULL if the file is not a music file, free after usage.

rc_tag_free ()

```
void                rc_tag_free                         (RCMusicMetaData *mmd);
```

Free the memory allocated for metadata struct (RCMusicMetaData).

mmd **:** the metadata

rc_tag_set_playing_metadata ()

```
void                rc_tag_set_playing_metadata         (const RCMusicMetaData *mmd);
```

Set playing metadata. Please do not use this function in plugins.

mmd **:** the metadata

rc_tag_get_playing_metadata ()

```
const RCMusicMetaData * rc_tag_get_playing_metadata     ();
```

Return the metadata which the player is playing.

Returns **:** The metadata which the player is playing.

rc_tag_get_name_from_fpath ()

```
gchar *              rc_tag_get_name_from_fpath          (const gchar *filename);
```

Return the base-name without extension from a full path or file name.

`filename` **:** the full path or file name

Returns **:** The base-name without extension.

rc_tag_find_file ()

```
gchar *              rc_tag_find_file                    (const gchar *dirname,
                                                          const gchar *str,
                                                          const gchar *extname);
```

Find a file in the directory by extension name and prefix string.

`dirname` **:** the directory name

`str` **:** the prefix string of the file name

`extname` **:** the extenstion name of the file

Returns **:** The file name which is found in the directory, NULL if not found, free after usage.

1.4 Player Object

Player Object — The player object of the player.

Synopsis

```
#include <player_object.h>

#define             RC_PLAYER_TYPE
                    RCPlayer;
                    RCPlayerClass;
RCPlayer *          rc_player_new                       ();
gboolean            rc_player_object_init               ();
GObject *           rc_player_object_get                ();
void                rc_player_object_signal_emit_simple (const char *name);
gulong              rc_player_object_signal_connect_simple
                                                        (const char *name,
                                                         GCallback callback);
void                rc_player_object_signal_disconnect  (gulong id);
```

Object Hierarchy

```
  GObject
   +----RCPlayer
```

Signals

```
  "lyric-found"                                       : Run First
  "lyric-not-found"                                   : Run First
  "player-continue"                                   : Run First
  "player-pause"                                      : Run First
  "player-play"                                       : Run First
  "player-stop"                                       : Run First
```

Description

The player object of the player, used in player signal processing.

Details

RC_PLAYER_TYPE

```
#define RC_PLAYER_TYPE (rc_player_get_type())
```

RCPlayer

```
typedef struct _RCPlayer RCPlayer;
```

RCPlayerClass

```
typedef struct {
    GObjectClass parent_class;
    void (*player_play)();
    void (*player_stop)();
    void (*player_pause)();
    void (*player_continue)();
    void (*lyric_found)();
    void (*lyric_not_found)();
} RCPlayerClass;
```

Provide signal process functions in the class.

GObjectClass ***parent_class***; parent class, not used

player_play () the function to call when the player starts playing

player_stop () the function to call when the player stopped

player_pause () the function to call when the player paused

player_continue () the function to call when the player continues playing

lyric_found () the function to call when the lyric data is found

lyric_not_found () the function to call when the lyric data is not found

rc_player_new ()

```
RCPlayer *          rc_player_new                       ();
```

Return a new RCPlayer object.

***Returns* :** A new RCPlayer object.

rc_player_object_init ()

```
gboolean            rc_player_object_init               ();
```

Make a default RCPlayer object for the player. Can be used only once.

***Returns* :** Whether the object is made.

rc_player_object_get ()

```
GObject *           rc_player_object_get                ();
```

Return the default RCPlayer object.

***Returns* :** The default RCPlayer object.

rc_player_object_signal_emit_simple ()

```
void                rc_player_object_signal_emit_simple (const char *name);
```

Emit a signal by the given name.

***`name`* :** the name of the signal

rc_player_object_signal_connect_simple ()

```
gulong              rc_player_object_signal_connect_simple
                                                        (const char *name,
                                                         GCallback callback);
```

Connect the GCallback function to the given signal for the default RCPlayer object.

***`name`* :** the name of the signal

***`callback`* :** the the GCallback to connect

***Returns* :** The handler ID.

rc_player_object_signal_disconnect ()

```
void                rc_player_object_signal_disconnect  (gulong id);
```

Disconnects thg handler from the default RCPlayer object, so it will not be called during any future or currently ongoing emissions of the signal it has been connected to. The `id` becomes invalid and may be reused.

***`id`* :** the handler ID

Signal Details

The "`lyric-found`" signal

```
void                user_function                      (RCPlayer *arg0,
                                                        gpointer  user_data)      : Run  ↩
                                                            First
```

Emitted when the lyric is found at the moment the player starts playing.

`user_data` **:** user data set when the signal handler was connected.

The "`lyric-not-found`" signal

```
void                user_function                      (RCPlayer *arg0,
                                                        gpointer  user_data)      : Run  ↩
                                                            First
```

Emitted when the lyric is not found at the moment the player starts playing.

`user_data` **:** user data set when the signal handler was connected.

The "`player-continue`" signal

```
void                user_function                      (RCPlayer *arg0,
                                                        gpointer  user_data)      : Run  ↩
                                                            First
```

Emitted after the player paused.

`user_data` **:** user data set when the signal handler was connected.

The "`player-pause`" signal

```
void                user_function                      (RCPlayer *arg0,
                                                        gpointer  user_data)      : Run  ↩
                                                            First
```

Emitted after the player paused.

`user_data` **:** user data set when the signal handler was connected.

The "`player-play`" signal

```
void                user_function                      (RCPlayer *arg0,
                                                        gpointer  user_data)      : Run  ↩
                                                            First
```

Emitted after the player starts playing.

`user_data` **:** user data set when the signal handler was connected.

The "player-stop" signal

```
void                  user_function                      (RCPlayer *arg0,
                                                          gpointer  user_data)       : Run  ↩
                                                             First
```

Emitted after the player stopped.

user_data **:** user data set when the signal handler was connected.

1.5 Lyric

Lyric — Process lyric data.

Synopsis

```
#include <lyric.h>

                    RCLyricData;
void                rc_lrc_init                         ();
void                rc_lrc_exit                         ();
gboolean            rc_lrc_read_from_file               (const gchar *filename);
void                rc_lrc_clean_data                   ();
const RCLyricData ** rc_lrc_get_lrc_data                ();
gsize               rc_lrc_get_lrc_length               ();
const gchar *       rc_lrc_get_text_data                ();
const RCLyricData * rc_lrc_get_line_by_time             (gint64 time);
const RCLyricData * rc_lrc_get_line_now                 ();
```

Description

Process lyric texts, like reading lyric from LRC file, etc.

Details

RCLyricData

```
typedef struct {
    gint64 time;
    gint64 length;
    guint index;
    gchar *text;
} RCLyricData;
```

Custom struct type to store the data of lyrics.

gint64 *time*; the start time of the lyric text

gint64 *length*; the time length of the lyric text

guint *index*; the line index number of the lyric text

gchar *_text_; the lyric text

rc_lrc_init ()

```
void                    rc_lrc_init                         ();
```

Initialize the lyric watch timer.

rc_lrc_exit ()

```
void                    rc_lrc_exit                         ();
```

Remove the lyric watch timer.

rc_lrc_read_from_file ()

```
gboolean                rc_lrc_read_from_file               (const gchar *filename);
```

Read lyric data from given file.

filename **:** the lyric file

Returns **:** Whether the file is read successfully.

rc_lrc_clean_data ()

```
void                    rc_lrc_clean_data                   ();
```

Clean the read lyric data from the player.

rc_lrc_get_lrc_data ()

```
const RCLyricData ** rc_lrc_get_lrc_data                    ();
```

Return the processed lyric array in the player.

Returns **:** The processed lyric array in the player, the data is stored in an array, NULL if there is no lyric data.

rc_lrc_get_lrc_length ()

```
gsize                   rc_lrc_get_lrc_length               ();
```

Return the length of the lyric array.

Returns **:** The length of the lyric array.

rc_lrc_get_text_data ()

```
const gchar *           rc_lrc_get_text_data                ();
```

Return the original lyric text in the player.

Returns **:** The original lyric text in the player, NULL if there is no lyric text.

rc_lrc_get_line_by_time ()

```
const RCLyricData * rc_lrc_get_line_by_time            (gint64 time);
```

Return the lyric line data by given time.

time **:** the time in nanosecond

Returns **:** The lyric line data by given time, NULL if not found.

rc_lrc_get_line_now ()

```
const RCLyricData * rc_lrc_get_line_now                ();
```

Return the lyric line data while the player is playing.

Returns **:** The lyric line data the player is playing.

1.6 Asynchronous Message Queue

Asynchronous Message Queue — Asynchronous message queue process among threads.

Synopsis

```
#include <msg.h>

void                (*RCMsgAsyncQueueWatchFunc)        (gpointer item,
                                                        gpointer data);
enum                RCMsgType;
                    RCMsgData;
guint               rc_msg_async_queue_watch_new       (GAsyncQueue *queue,
                                                        gint priority,
                                                        RCMsgAsyncQueueWatchFunc callback
                                                        gpointer data,
                                                        GDestroyNotify notify,
                                                        GMainContext *context);
void                rc_msg_init                        ();
void                rc_msg_push                        (RCMsgType type,
                                                        gpointer data);
```

Description

Process asynchronous message queue between GUI Thread (main thread) and other threads.

Details

RCMsgAsyncQueueWatchFunc ()

```
void                (*RCMsgAsyncQueueWatchFunc)        (gpointer item,
                                                        gpointer data);
```

The watch function type which executes when the message async queue changed.

item **:** queue item

data **:** user data

enum RCMsgType

```
typedef enum RCMsgType {
    MSG_TYPE_EMPTY = 0,
    MSG_TYPE_TEST = 1,
    MSG_TYPE_PL_INSERT = 2,
    MSG_TYPE_PL_REFRESH = 3,
    MSG_TYPE_PL_REMOVE = 4
}RCMsgType;
```

Types of the message.

`MSG_TYPE_EMPTY` empty message

`MSG_TYPE_TEST` test message

`MSG_TYPE_PL_INSERT` playlist insertion message

`MSG_TYPE_PL_REFRESH` playlist refresh message

`MSG_TYPE_PL_REMOVE` playlist remove message

RCMsgData

```
typedef struct {
    RCMsgType type;
    gpointer data;
} RCMsgData;
```

Custom struct to store message data.

RCMsgType *`type`*; message type

gpointer *`data`*; message data

rc_msg_async_queue_watch_new ()

```
guint                rc_msg_async_queue_watch_new        (GAsyncQueue *queue,
                                                          gint priority,
                                                          RCMsgAsyncQueueWatchFunc callback,
                                                          gpointer data,
                                                          GDestroyNotify notify,
                                                          GMainContext *context);
```

Add new watch to the given GAsyncQueue.

***`queue`* :** the GAsyncQueue to watch

***`priority`* :** the priority

***`callback`* :** the callback function to execute when the queue changed

***`data`* :** user data

***`notify`* :** a function to call when data is no longer in use, or NULL

***`context`* :** a GMainContext (if NULL, the default context will be used)

***Returns* :** The GSource ID of the new watch.

rc_msg_init ()

```
void                  rc_msg_init                       ();
```

Initialize the default asynchronous message queue for the player. Can be used only once.

rc_msg_push ()

```
void                  rc_msg_push                       (RCMsgType type,
                                                         gpointer data);
```

Add new message to the default asynchronous message queue.

type **:** the message type

data **:** the message data

1.7 Playlist

Playlist — Manage the playlists.

Synopsis

```
#include <playlist.h>

enum                RCPlist1Column;
enum                RCPlist2Column;
gboolean            rc_plist_init                    ();
void                rc_plist_exit                    ();
gboolean            rc_plist_insert_list             (const gchar *listname,
                                                      gint index);
gboolean            rc_plist_insert_music            (const gchar *uri,
                                                      gint list1_index,
                                                      gint list2_index);
void                rc_plist_list2_insert_item       (const gchar *uri,
                                                      const gchar *title,
                                                      const gchar *artist,
                                                      const gchar *album,
                                                      gint64 length,
                                                      gint trackno,
                                                      GtkListStore *store,
                                                      gint list2_index);
void                rc_plist_list2_refresh_item      (const gchar *uri,
                                                      const gchar *title,
                                                      const gchar *artist,
                                                      const gchar *album,
                                                      gint64 length,
                                                      gint trackno,
                                                      GtkTreeRowReference *reference);
void                rc_plist_list2_remove_item       (GtkTreeRowReference *reference);
gboolean            rc_plist_remove_list             (gint index);
gchar *             rc_plist_get_list1_name          (gint index);
gint                rc_plist_get_list1_length        ();
```

```
void                 rc_plist_set_list1_name            (gint index,
                                                         const gchar *name);
gint                 rc_plist_get_list2_length          (gint index);
gboolean             rc_plist_play_by_index             (gint list_index,
                                                         gint music_index);
gboolean             rc_plist_play_get_index            (gint *index1,
                                                         gint *index2);
void                 rc_plist_stop                      ();
gboolean             rc_plist_play_prev                 ();
gboolean             rc_plist_play_next                 (gboolean flag);
void                 rc_plist_set_play_mode             (gint repeat,
                                                         gint random);
void                 rc_plist_get_play_mode             (gint *repeat,
                                                         gint *random);
gboolean             rc_plist_load_playlist_setting     ();
gboolean             rc_plist_save_playlist_setting     ();
void                 rc_plist_build_default_list        ();
void                 rc_plist_plist_move2               (gint list_index,
                                                         GtkTreePath **from_paths,
                                                         gint f_length,
                                                         gint to_list_index);
void                 rc_plist_save_playlist             (const gchar *s_filename,
                                                         gint index);
void                 rc_plist_load_playlist             (const gchar *s_filename,
                                                         gint index);
GtkListStore *       rc_plist_get_list_store            (gint index);
GtkListStore *       rc_plist_get_list_head             ();
gboolean             rc_plist_list2_refresh             (gint list1_index);
gint                 rc_plist_import_job_get_length     ();
void                 rc_plist_import_job_cancel         ();
void                 rc_plist_load_argument             (char *argv[]);
gboolean             rc_plist_load_uri_from_remote      (const gchar *uri);
```

Description

Manage the playlists in the player and control all operations on playlists.

Details

enum RCPlist1Column

```
typedef enum RCPlist1Column {
    PLIST1_STATE = 0,
    PLIST1_NAME = 1,
    PLIST1_STORE = 2,
    PLIST1_LAST = 3
}RCPlist1Column;
```

The enum type to show the columns in ListStore1.

`PLIST1_STATE` the state image stock

`PLIST1_NAME` the list name

`PLIST1_STORE` the list store of list2

`PLIST1_LAST` the column number of list1

enum RCPlist2Column

```
typedef enum RCPlist2Column {
    PLIST2_URI = 0,
    PLIST2_STATE = 1,
    PLIST2_TITLE = 2,
    PLIST2_ARTIST = 3,
    PLIST2_ALBUM = 4,
    PLIST2_LENGTH = 5,
    PLIST2_TRACKNO = 6,
    PLIST2_LAST = 7
}RCPlist2Column;
```

The enum type to show the columns in ListStore2.

`PLIST2_URI` the URI of the music

`PLIST2_STATE` the state of the music

`PLIST2_TITLE` the title of the music

`PLIST2_ARTIST` the artist of the music

`PLIST2_ALBUM` the album of the music

`PLIST2_LENGTH` the time length of the music

`PLIST2_TRACKNO` the track number of the music

`PLIST2_LAST` the column number of list2

rc_plist_init ()

```
gboolean            rc_plist_init                         ();
```

Initialize playlists. Can be used only once.

***Returns* :** Whether the initiation succeeds.

rc_plist_exit ()

```
void                rc_plist_exit                         ();
```

Free all playlists data.

rc_plist_insert_list ()

```
gboolean            rc_plist_insert_list                  (const gchar *listname,
                                                           gint index);
```

Insert a new playlist into the playlist.

***`listname`* :** the name of the new playlist

***`index`* :** the position to insert, set to -1 to append to the last

***Returns* :** Whether the insertion succeeds.

rc_plist_insert_music ()

```
gboolean            rc_plist_insert_music               (const gchar *uri,
                                                         gint list1_index,
                                                         gint list2_index);
```

Import music by given URI to the playlist.

uri **:** the URI of the music

list1_index **:** the index of the playlists to insert to

list2_index **:** the position where the music insert to, -1 to append to the last

Returns **:** Whether the insertion succeeds.

rc_plist_list2_insert_item ()

```
void                rc_plist_list2_insert_item          (const gchar *uri,
                                                         const gchar *title,
                                                         const gchar *artist,
                                                         const gchar *album,
                                                         gint64 length,
                                                         gint trackno,
                                                         GtkListStore *store,
                                                         gint list2_index);
```

Insert music to list2 by metadata. WARNING: This function is only used in insertion job, if you really want to insert a music, please use rc_plist_insert_music().

uri **:** the URI of the item

title **:** the title of the item

artist **:** the artist of the item

album **:** the album of the item

length **:** the time length of the item

trackno **:** the track number of the item

store **:** the GtkListStore to insert to

list2_index **:** the position to insert to

rc_plist_list2_refresh_item ()

```
void                rc_plist_list2_refresh_item         (const gchar *uri,
                                                         const gchar *title,
                                                         const gchar *artist,
                                                         const gchar *album,
                                                         gint64 length,
                                                         gint trackno,
                                                         GtkTreeRowReference *reference);
```

Refresh the item in list2 by metadata. WARNING: This function is only used in refresh job, if you really want to refresh the playlist, please use rc_plist_list2_refresh().

`uri` **:** the URI of the item

`title` **:** the title of the item

`artist` **:** the artist of the item

`album` **:** the album of the item

`length` **:** the time length of the item

`trackno` **:** the track number of the item

`reference` **:** the path reference of the item

rc_plist_list2_remove_item ()

```
void                rc_plist_list2_remove_item          (GtkTreeRowReference *reference);
```

Remove invalid item in list2. WARNING: This function is only used to remove invalid music item.

`reference` **:** the path reference of the item

rc_plist_remove_list ()

```
gboolean            rc_plist_remove_list                (gint index);
```

Remove a playlist by given index.

`index` **:** the index of the playlist to remove

Returns **:** Whether the operation succeeds.

rc_plist_get_list1_name ()

```
gchar *             rc_plist_get_list1_name             (gint index);
```

Return the name of the list.

`index` **:** the index number of the playlist

Returns **:** The name of the list, NULL if not found, free after usage.

rc_plist_get_list1_length ()

```
gint                rc_plist_get_list1_length           ();
```

Return the length of playlists.

Returns **:** The length of playlists.

rc_plist_set_list1_name ()

```
void                rc_plist_set_list1_name             (gint index,
                                                         const gchar *name);
```

Rename an exist playlist.

`index` **:** the index of the playlist to rename

`name` **:** the new name

rc_plist_get_list2_length ()

```
gint                    rc_plist_get_list2_length          (gint index);
```

Return the music number in the playlist.

index **:** the index of the playlist

Returns **:** The music number in the playlist.

rc_plist_play_by_index ()

```
gboolean                rc_plist_play_by_index             (gint list_index,
                                                            gint music_index);
```

Play music by given list1 index and list2 index.

list_index **:** the index of the playlist

music_index **:** the index of the music in the playlist

Returns **:** Whether the operation succeeds.

rc_plist_play_get_index ()

```
gboolean                rc_plist_play_get_index            (gint *index1,
                                                            gint *index2);
```

Get the playlist index number and the music index number of playing music.

index1 **:** the index of the playlist

index2 **:** the index of the music in the playlist

Returns **:** Whether the indexes are set.

rc_plist_stop ()

```
void                    rc_plist_stop                      ();
```

Clean the references data when the player stops.

rc_plist_play_prev ()

```
gboolean                rc_plist_play_prev                 ();
```

Play the previous music in the playlist.

Returns **:** Whether the operation succeeds.

rc_plist_play_next ()

```
gboolean            rc_plist_play_next                  (gboolean flag);
```

Play the next music in the playlist.

flag **:** whether the operation is produced by player, if it is TRUE, the operation will effect by playing mode (repeat mode and random mode)

Returns **:** Whether the operation succeeds.

rc_plist_set_play_mode ()

```
void                rc_plist_set_play_mode              (gint repeat,
                                                         gint random);
```

Set the play mode of the player.

repeat **:** the repeat mode

random **:** the random mode

rc_plist_get_play_mode ()

```
void                rc_plist_get_play_mode              (gint *repeat,
                                                         gint *random);
```

Get the play mode of the player.

repeat **:** return the repeat mode, can be NULL

random **:** return the random mdoe, can be NULL

rc_plist_load_playlist_setting ()

```
gboolean            rc_plist_load_playlist_setting      ();
```

Load playlists from file.

Returns **:** Whether the operation succeeds.

rc_plist_save_playlist_setting ()

```
gboolean            rc_plist_save_playlist_setting      ();
```

Save the playlist settings to file.

Returns **:** Whether the operation succeeds.

rc_plist_build_default_list ()

```
void                rc_plist_build_default_list         ();
```

Make a default playlist.

rc_plist_plist_move2 ()

```
void                rc_plist_plist_move2                (gint list_index,
                                                         GtkTreePath **from_paths,
                                                         gint f_length,
                                                         gint to_list_index);
```

Move item(s) in the playlist to another playlist.

list_index **:** the index of the source playlist

from_paths **:** the GtkTreePaths to move

f_length **:** the length of the the GtkTreePaths

to_list_index **:** the index of the target playlist

rc_plist_save_playlist ()

```
void                rc_plist_save_playlist              (const gchar *s_filename,
                                                         gint index);
```

Save the playlist to a file.

s_filename **:** the path of the playlist file

index **:** the index of the playlist to save

rc_plist_load_playlist ()

```
void                rc_plist_load_playlist              (const gchar *s_filename,
                                                         gint index);
```

Load a playlist from a playlist file.

s_filename **:** the path of the playlist file

index **:** the index of the playlist to load

rc_plist_get_list_store ()

```
GtkListStore *      rc_plist_get_list_store             (gint index);
```

Return the GtkListStore of the playlist by given index.

index **:** the index of the playlist

Returns **:** The GtkListStore of the playlist.

rc_plist_get_list_head ()

```
GtkListStore *      rc_plist_get_list_head              ();
```

Return the GtkListStore of the playlists.

Returns **:** The GtkListStore of the playlists.

rc_plist_list2_refresh ()

```
gboolean            rc_plist_list2_refresh             (gint list1_index);
```

Refresh the music information in the playlist.

list1_index **:** the index of the playlist to refresh

Returns **:** Whether the operation succeeds.

rc_plist_import_job_get_length ()

```
gint                rc_plist_import_job_get_length     ();
```

Return the remaining job length in the job queue.

Returns **:** The remaining job length in the job queue.

rc_plist_import_job_cancel ()

```
void                rc_plist_import_job_cancel         ();
```

Cancel all remaining jobs in the job queue.

rc_plist_load_argument ()

```
void                rc_plist_load_argument             (char *argv[]);
```

Import music from the given argument list.

argv **:** the argument list

rc_plist_load_uri_from_remote ()

```
gboolean            rc_plist_load_uri_from_remote      (const gchar *uri);
```

Import music from remote (e.g. D-Bus) by given URI.

uri **:** the URI of the music to import

Returns **:** Whether the operation succeeds.

1.8 Settings

Settings — Manage the settings of the player.

Synopsis

```
#include <settings.h>

void                rc_set_init                     ();
void                rc_set_exit                     ();
gchar *             rc_set_get_string               (const gchar *group_name,
                                                     const gchar *key,
                                                     GError **error);
gint                rc_set_get_integer              (const gchar *group_name,
                                                     const gchar *key,
                                                     GError **error);
gdouble             rc_set_get_double               (const gchar *group_name,
                                                     const gchar *key,
                                                     GError **error);
gboolean            rc_set_get_boolean              (const gchar *group_name,
                                                     const gchar *key,
                                                     GError **error);
gchar **            rc_set_get_string_list          (const gchar *group_name,
                                                     const gchar *key,
                                                     gsize *length,
                                                     GError **error);
gboolean *          rc_set_get_boolean_list         (const gchar *group_name,
                                                     const gchar *key,
                                                     gsize *length,
                                                     GError **error);
gint *              rc_set_get_integer_list         (const gchar *group_name,
                                                     const gchar *key,
                                                     gsize *length,
                                                     GError **error);
gdouble *           rc_set_get_double_list          (const gchar *group_name,
                                                     const gchar *key,
                                                     gsize *length,
                                                     GError **error);
void                rc_set_set_string               (const gchar *group_name,
                                                     const gchar *key,
                                                     const gchar *string);
void                rc_set_set_boolean              (const gchar *group_name,
                                                     const gchar *key,
                                                     gboolean value);
void                rc_set_set_integer              (const gchar *group_name,
                                                     const gchar *key,
                                                     gint value);
void                rc_set_set_double               (const gchar *group_name,
                                                     const gchar *key,
                                                     gdouble value);
void                rc_set_set_string_list          (const gchar *group_name,
                                                     const gchar *key,
                                                     const gchar * const list[],
                                                     gsize length);
void                rc_set_set_boolean_list         (const gchar *group_name,
                                                     const gchar *key,
                                                     gboolean list[],
                                                     gsize length);
void                rc_set_set_integer_list         (const gchar *group_name,
                                                     const gchar *key,
                                                     gint list[],
                                                               gsize length);
void                 rc_set_set_double_list                   (const gchar *group_name,
                                                               const gchar *key,
                                                               gdouble list[],
                                                               gsize length);
gboolean             rc_set_load_setting                      (const gchar *filename);
void                 rc_set_save_setting                      (const gchar *filename);
GKeyFile *           rc_set_get_plugin_configure              ();
```

Description

Manage the settings of player. Store settings in an ini-like configuration file.

Details

rc_set_init ()

```
void                 rc_set_init                              ();
```

Initialize and load the settings of the player.

rc_set_exit ()

```
void                 rc_set_exit                              ();
```

Free the settings when exits.

rc_set_get_string ()

```
gchar *              rc_set_get_string                        (const gchar *group_name,
                                                               const gchar *key,
                                                               GError **error);
```

Returns the string value associated with key under group_name.

group_name **:** a group name

key **:** a key

error **:** return location for a GError, or NULL

Returns **:** A newly allocated string or NULL if the specified key cannot be found.

rc_set_get_integer ()

```
gint                 rc_set_get_integer                       (const gchar *group_name,
                                                               const gchar *key,
                                                               GError **error);
```

Returns the value associated with key under group_name as an integer.

group_name **:** a group name

key **:** a key

error **:** return location for a GError, or NULL

Returns **:** The value associated with the key as an integer, or 0 if the key was not found or could not be parsed.

rc_set_get_double ()

```
gdouble             rc_set_get_double                   (const gchar *group_name,
                                                         const gchar *key,
                                                         GError **error);
```

Returns the value associated with key under group_name as a double. If group_name is NULL, the start_group is used.

group_name **:** a group name

key **:** a key

error **:** return location for a GError, or NULL

Returns **:** The value associated with the key as a double, or 0.0 if the key was not found or could not be parsed.

rc_set_get_boolean ()

```
gboolean            rc_set_get_boolean                  (const gchar *group_name,
                                                         const gchar *key,
                                                         GError **error);
```

Returns the value associated with key under group_name as a boolean.

group_name **:** a group name

key **:** a key

error **:** return location for a GError, or NULL

Returns **:** The value associated with the key as a boolean, or FALSE if the key was not found or could not be parsed.

rc_set_get_string_list ()

```
gchar **            rc_set_get_string_list              (const gchar *group_name,
                                                         const gchar *key,
                                                         gsize *length,
                                                         GError **error);
```

Returns the values associated with key under group_name.

group_name **:** a group name

key **:** a key

length **:** return location for the number of returned strings, or NULL

error **:** return location for a GError, or NULL

Returns **:** A NULL-terminated string array or NULL if the specified key cannot be found. The array should be freed with g_strfreev().

rc_set_get_boolean_list ()

```
gboolean *          rc_set_get_boolean_list             (const gchar *group_name,
                                                         const gchar *key,
                                                         gsize *length,
                                                         GError **error);
```

Returns the values associated with key under group_name as booleans.

***group_name* :** a group name

***key* :** a key

***length* :** the number of booleans returned

***error* :** return location for a GError, or NULL

***Returns* :** The values associated with the key as a list of booleans, or NULL if the key was not found or could not be parsed.

rc_set_get_integer_list ()

```
gint *              rc_set_get_integer_list             (const gchar *group_name,
                                                         const gchar *key,
                                                         gsize *length,
                                                         GError **error);
```

Returns the values associated with key under group_name as integers.

***group_name* :** a group name

***key* :** a key

***length* :** the number of integers returned

***error* :** return location for a GError, or NULL

***Returns* :** The values associated with the key as a list of integers, or NULL if the key was not found or could not be parsed.

rc_set_get_double_list ()

```
gdouble *           rc_set_get_double_list              (const gchar *group_name,
                                                         const gchar *key,
                                                         gsize *length,
                                                         GError **error);
```

Returns the values associated with key under group_name as doubles.

***group_name* :** a group name

***key* :** a key

***length* :** the number of doubles returned

***error* :** return location for a GError, or NULL

***Returns* :** The values associated with the key as a list of doubles, or NULL if the key was not found or could not be parsed.

rc_set_set_string ()

```
void                rc_set_set_string                   (const gchar *group_name,
                                                         const gchar *key,
                                                         const gchar *string);
```

Associates a new string value with key under group_name. If key cannot be found then it is created. If group_name cannot be found then it is created.

group_name **:** a group name

key **:** a key

string **:** a string

rc_set_set_boolean ()

```
void                rc_set_set_boolean                  (const gchar *group_name,
                                                         const gchar *key,
                                                         gboolean value);
```

Associates a new boolean value with key under group_name. If key cannot be found then it is created. If group_name cannot be found then it is created.

group_name **:** a group name

key **:** a key

value **:** TRUE or FALSE

rc_set_set_integer ()

```
void                rc_set_set_integer                  (const gchar *group_name,
                                                         const gchar *key,
                                                         gint value);
```

Associates a new integer value with key under group_name. If key cannot be found then it is created. If group_name cannot be found then it is created.

group_name **:** a group name

key **:** a key

value **:** an integer value

rc_set_set_double ()

```
void                rc_set_set_double                   (const gchar *group_name,
                                                         const gchar *key,
                                                         gdouble value);
```

Associates a new double value with key under group_name. If key cannot be found then it is created. If group_name cannot be found then it is created.

group_name **:** a group name

key **:** a key

value **:** an double value

rc_set_set_string_list ()

```
void                rc_set_set_string_list              (const gchar *group_name,
                                                         const gchar *key,
                                                         const gchar * const list[],
                                                         gsize length);
```

Associates a list of string values for key under group_name. If key cannot be found then it is created. If group_name cannot be found then it is created.

group_name **:** a group name

key **:** a key

list **:** an array of string values

length **:** number of string values in list

rc_set_set_boolean_list ()

```
void                rc_set_set_boolean_list             (const gchar *group_name,
                                                         const gchar *key,
                                                         gboolean list[],
                                                         gsize length);
```

Associates a list of boolean values with key under group_name. If key cannot be found then it is created. If group_name is NULL, the start_group is used.

group_name **:** a group name

key **:** a key

list **:** an array of boolean values

length **:** number of string values in list

rc_set_set_integer_list ()

```
void                rc_set_set_integer_list             (const gchar *group_name,
                                                         const gchar *key,
                                                         gint list[],
                                                         gsize length);
```

Associates a list of integer values with key under group_name. If key cannot be found then it is created. If group_name is NULL, the start_group is used.

group_name **:** a group name

key **:** a key

list **:** an array of integer values

length **:** number of integer values in list

rc_set_set_double_list ()

```
void                rc_set_set_double_list              (const gchar *group_name,
                                                         const gchar *key,
                                                         gdouble list[],
                                                         gsize length);
```

Associates a list of double values with key under group_name. If key cannot be found then it is created. If group_name is NULL, the start_group is used.

group_name **:** a group name

key **:** a key

list **:** an array of double values

length **:** number of double values in list

rc_set_load_setting ()

```
gboolean            rc_set_load_setting                 (const gchar *filename);
```

Read configuration from given file.

filename **:** the path of configuration file

Returns **:** Whether the configuration file is read.

rc_set_save_setting ()

```
void                rc_set_save_setting                 (const gchar *filename);
```

Save configuration data to given file.

filename **:** the path of configuration file

rc_set_get_plugin_configure ()

```
GKeyFile *          rc_set_get_plugin_configure         ();
```

Return the GKeyFile of plugin configuration.

Returns **:** The GKeyFile of plugin configuration.

1.9 Plugin Support

Plugin Support — Plugin support of the player.

Synopsis

```
#include <plugin.h>

enum                RCPluginType;
                    RCPluginConfData;
                    RCPluginModuleData;
void                rc_plugin_init                      ();
void                rc_plugin_exit                      ();
gboolean            rc_plugin_search_dir                (const gchar *dirname);
const GSList *      rc_plugin_get_list                  ();
void                rc_plugin_list_free                 ();
void                rc_plugin_conf_free                 (RCPluginConfData *plugin_data);
RCPluginConfData *  rc_plugin_conf_load                 (const gchar *filename);
gboolean            rc_plugin_load                      (RCPluginType type,
                                                         const gchar *filename);
gboolean            rc_plugin_configure                 (RCPluginType type,
                                                         const gchar *filename);
void                rc_plugin_close                     (RCPluginType type,
                                                         const gchar *filename);
gboolean            rc_plugin_check_running             (RCPluginType type,
                                                         const gchar *path);
```

Description

Plugin support of the player. It supports module type (usually it is a dynamic link library) plugin.

Details

enum RCPluginType

```
typedef enum RCPluginType {
    PLUGIN_TYPE_MODULE = 1,
}RCPluginType;
```

The enum type to show the type of the plugin.

PLUGIN_TYPE_MODULE the plugin is a module

RCPluginConfData

```
typedef struct {
    gchar *path;
    gchar *name;
    gchar *desc;
    gchar *author;
    gchar *version;
    gchar *website;
    RCPluginType type;
} RCPluginConfData;
```

The plugin configuration data structure.

gchar ***path**; the path of the plugin file

gchar *_name_; the name of the plugin

gchar *_desc_; the description of the plugin

gchar *_author_; the author of the plugin

gchar *_version_; the version of the plugin

gchar *_website_; the website of the plugin

RCPluginType _type_; the type of the plugin

RCPluginModuleData

```
typedef struct {
    gchar *group_name;
    gboolean resident;
} RCPluginModuleData;
```

The data structure of module.

gchar *_group_name_; the group name used in plugin configure file

gboolean _resident_; whether the plugin can be removed while the player is running

rc_plugin_init ()

```
void                    rc_plugin_init                      ();
```

Initialize the plugin support of the player. Can be used only once.

rc_plugin_exit ()

```
void                    rc_plugin_exit                      ();
```

Free and close all plugins when exits.

rc_plugin_search_dir ()

```
gboolean                rc_plugin_search_dir                (const gchar *dirname);
```

Search all plugins in a directory. And save them in a list.

***dirname* :** the path of the directory

***Returns* :** Whether the directory has plugin.

rc_plugin_get_list ()

```
const GSList *          rc_plugin_get_list                  ();
```

Return all plugins in the plugin list found before.

***Returns* :** The plugin list found before.

rc_plugin_list_free ()

```
void                rc_plugin_list_free                 ();
```

Free and clean the plugin list.

rc_plugin_conf_free ()

```
void                rc_plugin_conf_free                 (RCPluginConfData *plugin_data);
```

Free the plugin configure data.

***plugin_data* :** the plugin configure data

rc_plugin_conf_load ()

```
RCPluginConfData *  rc_plugin_conf_load                 (const gchar *filename);
```

Open a configuration file.

***filename* :** the configuration file to open

***Returns* :** The configuration data read from the file, NULL if error happens.

rc_plugin_load ()

```
gboolean            rc_plugin_load                      (RCPluginType type,
                                                         const gchar *filename);
```

Open and run the plugin.

***type* :** the type of the plugin

***filename* :** the path of the plugin

***Returns* :** Whether the operation succeeds.

rc_plugin_configure ()

```
gboolean            rc_plugin_configure                 (RCPluginType type,
                                                         const gchar *filename);
```

Open and configure the plugin.

***type* :** the type of the plugin

***filename* :** the path of the plugin

***Returns* :** Whether the operation succeeds.

rc_plugin_close ()

```
void                rc_plugin_close                     (RCPluginType type,
                                                         const gchar *filename);
```

Close the plugin.

`type` **:** the type of the plugin

`filename` **:** the path of the plugin

rc_plugin_check_running ()

```
gboolean            rc_plugin_check_running             (RCPluginType type,
                                                         const gchar *path);
```

Check if the plugin is running.

`type` **:** the type of the plugin

`path` **:** the path of the plugin

Returns **:** Whether the plugin is running.

1.10 Debug

Debug — Debug and print debug information.

Synopsis

```
#include <debug.h>

#define             DEBUG_MODE
gboolean            rc_debug_get_flag                   ();
void                rc_debug_set_mode                   (gboolean mode);
gint                rc_debug_print                      (const gchar *format,
                                                         ...);
gint                rc_debug_perror                     (const gchar *format,
                                                         ...);
```

Description

Debug and print information of the working status of the player.

Details

DEBUG_MODE

```
#define DEBUG_MODE FALSE
```

rc_debug_get_flag ()

```
gboolean            rc_debug_get_flag                   ();
```

Return the debug flag.

***Returns* :** Whether the debug flag is enabled.

rc_debug_set_mode ()

```
void                rc_debug_set_mode                   (gboolean mode);
```

Set the debug mode.

***mode* :** the debug flag, set to TRUE to enable debug mode

rc_debug_print ()

```
gint                rc_debug_print                      (const gchar *format,
                                                         ...);
```

Print debug message when debug mode is enabled.

***format* :** a standard printf() format string

... : the arguments to insert in the output

***Returns* :** the number of bytes printed.

rc_debug_perror ()

```
gint                rc_debug_perror                     (const gchar *format,
                                                         ...);
```

Print error message on standard error (stderr).

***format* :** a standard printf() format string

... : the arguments to insert in the output

***Returns* :** the number of bytes printed.

Chapter 2

Player UI

2.1 Main UI

Main UI — The main UI of the player.

Synopsis

```
#include <gui.h>

                    RCGuiData;
RCGuiData *         rc_gui_get_data                    ();
gboolean            rc_gui_init                        ();
void                rc_gui_quit_player                 ();
void                rc_gui_music_info_set_data         (const gchar *title,
                                                        const gpointer data);
void                rc_gui_time_label_set_text         (gint64 time);
void                rc_gui_set_play_button_state       (gboolean state);
void                rc_gui_seek_scaler_disable         ();
void                rc_gui_seek_scaler_enable          ();
void                rc_gui_set_volume                  (gdouble volume);
void                rc_gui_set_player_mode             ();
gboolean            rc_gui_set_cover_image_by_file     (const gchar *filename);
gboolean            rc_gui_set_cover_image_by_buf      (const GstBuffer *buf);
void                rc_gui_status_task_set             (guint type,
                                                        guint len);
void                rc_gui_status_progress_set_progress ();
guint               rc_gui_view_add_page               (const gchar *name,
                                                        const gchar *title,
                                                        GtkWidget *widget);
gboolean            rc_gui_view_remove_page            (guint id);
```

Description

Show the main UI of the player.

Details

RCGuiData

```
typedef struct {
    GtkUIManager *main_ui;
    GtkActionGroup *main_action_group;
    GtkWidget *main_window;
    GtkWidget *eq_vbox;
    GtkWidget *plist_notebook;
    GtkWidget *title_label, *artist_label, *album_label;
    GtkWidget *time_label, *length_label, *info_label;
    GtkWidget *album_image, *album_eventbox, *album_frame;
    GtkWidget *control_images[4], *control_buttons[4];
    GtkWidget *volume_button;
    GtkWidget *time_scroll_bar;
    GtkWidget *lrc_label, *lrc_viewport;
    GtkWidget *list1_tree_view, *list2_tree_view;
    GtkWidget *list1_scr_window, *list2_scr_window;
    GtkWidget *status_hbox, *status_progress, *status_label;
    GtkWidget *status_cancel_button;
    GtkWidget *list_hpaned;
    GtkTreeModel *list1_tree_model, *list2_tree_model;
    GtkTreeSelection *list1_selection, *list2_selection;
    GtkCellRenderer *renderer_text[5];
    GtkCellRenderer *renderer_pixbuf[2];
    GtkAdjustment *lrc_vport_adj;
    guint main_window_width;
    guint main_window_height;
    guint status_task_length;
    gboolean update_seek_scale_flag;
    guint time_info_refresh_timeout;
    GtkTreeRowReference *list1_selected_reference;
    GdkPixbuf *no_cover_image;
    GdkPixbuf *icon_image;
    GtkStatusIcon *tray_icon;
} RCGuiData;
```

Custom structure type to store the UI data. Please do not change the data in this structure.

GtkUIManager ****main_ui***; the GtkUIManager which manages the menus

GtkActionGroup ****main_action_group***; the action groups

GtkWidget ****main_window***; the main window

GtkWidget ****eq_vbox***; the GtkBox which stores equalizer widgets

GtkWidget ****plist_notebook***; the notebook which stores playlist widgets

GtkWidget ****title_label***; show the title text on the player window

GtkWidget ****artist_label***; show the artist text on the player window

GtkWidget ****album_label***; show the album text on the player window

GtkWidget ****time_label***; show the time text on the player window

GtkWidget ****length_label***; show the time length text on the player window

GtkWidget ****info_label***; show the music information text on the player window

GtkWidget ****album_image***; show album image on the player window

GtkWidget ****album_eventbox***; process the events on album image

GtkWidget ****album_frame***; the frame of album image widget

GtkWidget *__control_images__[4]; the image widgets of control buttons

GtkWidget *__control_buttons__[4]; the button widgets of control buttons

GtkWidget *__volume_button__; the volume control button

GtkWidget *__time_scroll_bar__; the time scaler bar

GtkWidget *__lrc_label__; show lyric text on the player window

GtkWidget *__lrc_viewport__; the viewport which makes lyric text widget scrollable

GtkWidget *__list1_tree_view__; the list view of list1

GtkWidget *__list2_tree_view__; the list view of list2

GtkWidget *__list1_scr_window__; add scrollbars on list1

GtkWidget *__list2_scr_window__; add scrollbars on list2

GtkWidget *__status_hbox__; the GtkBox which stores status widgets

GtkWidget *__status_progress__; show progress of status on the player window

GtkWidget *__status_label__; show status text on the player window

GtkWidget *__status_cancel_button__; the cancel button to cancel all working tasks

GtkWidget *__list_hpaned__; the widget with two adjustable panes

GtkTreeModel *__list1_tree_model__; the GtkTreeModel of list1

GtkTreeModel *__list2_tree_model__; the GtkTreeModel of list2

GtkTreeSelection *__list1_selection__; the GtkTreeSelection of list1

GtkTreeSelection *__list2_selection__; the GtkTreeSelection of list2

GtkCellRenderer *__renderer_text__[5]; the text renderers of list1 & list2

GtkCellRenderer *__renderer_pixbuf__[2]; the image renderers of list1 & list2

GtkAdjustment *__lrc_vport_adj__; the GtkAdjustment object of lyric viewport

guint __main_window_width__; the default width of main window

guint __main_window_height__; the default height of main window

guint __status_task_length__; the length of working task

gboolean __update_seek_scale_flag__; whether the time scaler can be updated

guint __time_info_refresh_timeout__; the ID of time information update timer

GtkTreeRowReference *__list1_selected_reference__; the GtkTreeRowReference of selected item in list1

GdkPixbuf *__no_cover_image__; the default image of cover image

GdkPixbuf *__icon_image__; the icon of the player

GtkStatusIcon *__tray_icon__; the icon shows on the system tray

rc_gui_get_data ()

```
RCGuiData *         rc_gui_get_data                     ();
```

Return the data of main UI structure.

***Returns* :** The data of main UI structure.

rc_gui_init ()

```
gboolean            rc_gui_init                        ();
```

Initialize the main window of the player. Can be used only once.

***Returns* :** Whether the initiation succeeds.

rc_gui_quit_player ()

```
void                rc_gui_quit_player                 ();
```

Quit the player.

rc_gui_music_info_set_data ()

```
void                rc_gui_music_info_set_data         (const gchar *title,
                                                        const gpointer data);
```

Set the data in the information labels.

***`title`* :** the title to set

***`data`* :** the metadata, the type should be RCMetaData (defined in tag.h)

rc_gui_time_label_set_text ()

```
void                rc_gui_time_label_set_text         (gint64 time);
```

Set time label of the player.

***`time`* :** the time to set, in nanosecond.

rc_gui_set_play_button_state ()

```
void                rc_gui_set_play_button_state       (gboolean state);
```

Set play button state.

***`state`* :** the state of the play button, if it's TRUE, the image of the button is pause icon, else the image is play icon.

rc_gui_seek_scaler_disable ()

```
void                rc_gui_seek_scaler_disable         ();
```

Disable the scaler bar and the time control menus.

rc_gui_seek_scaler_enable ()

```
void                rc_gui_seek_scaler_enable          ();
```

Enable the scaler bar and the time control menus.

rc_gui_set_volume ()

```
void                rc_gui_set_volume                   (gdouble volume);
```

Set the volume bar value.

***volume* :** the volume to set, the value should be between 0.0 and 100.0

rc_gui_set_player_mode ()

```
void                rc_gui_set_player_mode              ();
```

Set the player repeat mode and random mode (GUI Only). Only used when startup.

rc_gui_set_cover_image_by_file ()

```
gboolean            rc_gui_set_cover_image_by_file      (const gchar *filename);
```

Set the image of cover.

***filename* :** the path of the cover image file

***Returns* :** Whether the image is set.

rc_gui_set_cover_image_by_buf ()

```
gboolean            rc_gui_set_cover_image_by_buf       (const GstBuffer *buf);
```

Set the image of cover by GstBuffer.

***buf* :** the GstBuffer which contains the cover image

***Returns* :** Whether the image is set.

rc_gui_status_task_set ()

```
void                rc_gui_status_task_set              (guint type,
                                                         guint len);
```

Set the type and the length of tasks.

***type* :** the task type: 1=Import, 2=Refresh others=None

***len* :** the length of the task

rc_gui_status_progress_set_progress ()

```
void                rc_gui_status_progress_set_progress ();
```

Set the remaining tasks for status progressbar. This function is usually used to refresh the work status.

rc_gui_view_add_page ()

```
guint               rc_gui_view_add_page                (const gchar *name,
                                                         const gchar *title,
                                                         GtkWidget *widget);
```

Add new view page and menu to player.

name **:** the name of the menu to add

title **:** the string which shows on the menu

widget **:** the widget to add to the page

Returns **:** The unique ID of the added page.

rc_gui_view_remove_page ()

```
gboolean            rc_gui_view_remove_page             (guint id);
```

Remove a view page from player.

id **:** the unique ID of the page to remove

Returns **:** Whether this operation is succeeded.

2.2 Dialogs

Dialogs — Dialogs in the player.

Synopsis

```
#include <gui.h>

#define             MAX_DIR_DEPTH
void                rc_gui_about_player                 ();
void                rc_gui_show_message_dialog          (GtkMessageType type,
                                                         const gchar *title,
                                                         const gchar *format,
                                                         ...);
void                rc_gui_show_open_dialog             ();
void                rc_gui_open_music_directory         ();
void                rc_gui_save_playlist_dialog         ();
void                rc_gui_load_playlist_dialog         ();
void                rc_gui_save_all_playlists_dialog    ();
```

Description

Show dialogs in the player.

Details

MAX_DIR_DEPTH

```
#define MAX_DIR_DEPTH 5
```

Maximum search depth while searching the music files in a directory.

rc_gui_about_player ()

```
void                rc_gui_about_player                 ();
```

Show the about information of this player.

rc_gui_show_message_dialog ()

```
void                rc_gui_show_message_dialog          (GtkMessageType type,
                                                         const gchar *title,
                                                         const gchar *format,
                                                         ...);
```

Show message dialog in the player.

`type` **:** type of message

`title` **:** title of the message

`format` **:** printf()-style format string, or NULL, allow-none

`...` **:** arguments for *`format`*

rc_gui_show_open_dialog ()

```
void                rc_gui_show_open_dialog             ();
```

Show a music import dialog for importing music files.

rc_gui_open_music_directory ()

```
void                rc_gui_open_music_directory         ();
```

Show a music import dialog for importing all music files in a directory.

rc_gui_save_playlist_dialog ()

```
void                rc_gui_save_playlist_dialog         ();
```

Show a playlist export dialog for exporting the selected playlist to a playlist file (M3U Format).

rc_gui_load_playlist_dialog ()

```
void                rc_gui_load_playlist_dialog         ();
```

Show a playlist import dialog for importing all music files in the playlist file.

rc_gui_save_all_playlists_dialog ()

```
void                  rc_gui_save_all_playlists_dialog    ();
```

Show a playlist export dialog for exporting all playlists in the player to playlist files, then putting these files into the given directory.

2.3 Equalizer UI

Equalizer UI — The equalizer of the player.

Synopsis

```
#include <gui_eq.h>

                      RCGuiEQData;
void                  rc_gui_eq_data_init                 ();
void                  rc_gui_eq_init                      ();
RCGuiEQData *         rc_gui_eq_get_data                  ();
```

Description

Show the equalizer of the player.

Details

RCGuiEQData

```
typedef struct {
    GtkWidget *eq_combobox;
    GtkWidget *eq_scales[10];
    GtkWidget *eq_labels[10];
    GtkWidget *db_labels[3];
    GtkWidget *save_button;
    GtkWidget *import_button;
} RCGuiEQData;
```

Custom structure type to store the equalizer UI data. Please do not change the data in this structure.

GtkWidget *`eq_combobox`; the combo-box

GtkWidget *`eq_scales`[10]; scalers

GtkWidget *`eq_labels`[10]; labels to show the frequencies

GtkWidget *`db_labels`[3]; labels to show the decibels

GtkWidget *`save_button`; the button to save the equalizer style

GtkWidget *`import_button`; the button to import the equalizer style

rc_gui_eq_data_init ()

```
void                  rc_gui_eq_data_init                 ();
```

Initialize the equalizer data. Can be used only once.

rc_gui_eq_init ()

```
void                rc_gui_eq_init                      ();
```

Initialize the equalizer UI. Can be used only once.

rc_gui_eq_get_data ()

```
RCGuiEQData *       rc_gui_eq_get_data                  ();
```

Return the UI Data of the equalizer.

***Returns* :** the UI data of the equalizer.

2.4 UI Styles and Themes

UI Styles and Themes — Set the styles and themes of the player window.

Synopsis

```
#include <gui_style.h>

                    RCGuiColorStyle;
void                rc_gui_style_init                   ();
void                rc_gui_style_refresh                ();
void                rc_gui_style_set_color_style        (const RCGuiColorStyle *style_data
const RCGuiColorStyle * rc_gui_style_get_color_style    (gint index);
void                rc_gui_style_set_color_style_by_index
                                                        (gint index);
```

Description

Set the styles and themes of the player window.

Details

RCGuiColorStyle

```
typedef struct {
    gchar name[32];
    GdkColor label_font_color;
    GdkColor time_font_color;
    GdkColor title_font_color;
    GdkColor artist_font_color;
    GdkColor album_font_color;
    GdkColor info_font_color;
    GdkColor length_font_color;
    GdkColor lyric_font_color;
    GdkColor window_bg_color;
    GdkColor time_scalerbar_handle_normal_color;
    GdkColor time_scalerbar_handle_prelight_color;
    GdkColor time_scalerbar_handle_selected_color;
    GdkColor button_bg_color;
    GdkColor button_prelight_color;
    GdkColor button_active_color;
    GdkColor listview_base_normal_color;
    GdkColor listview_base_selected_color;
    GdkColor listview_base_active_color;
    GdkColor listview_font_normal_color;
    GdkColor listview_font_selected_color;
    GdkColor listview_font_active_color;
    GdkColor listview_scrbar_color;
} RCGuiColorStyle;
```

The color style data structure.

gchar *name*[32]; the name of the color style

GdkColor *label_font_color*; the font color of the labels

GdkColor *time_font_color*; the font color of the time label

GdkColor *title_font_color*; the font color of the title label

GdkColor *artist_font_color*; the font color of the artist label

GdkColor *album_font_color*; the font color of the album label

GdkColor *info_font_color*; the font color of the information label

GdkColor *length_font_color*; the font color of the time length label

GdkColor *lyric_font_color*; the font color of the lyric label

GdkColor *window_bg_color*; the background color of the player window

GdkColor *time_scalerbar_handle_normal_color*; the color of the handle of the time scalerbar

GdkColor *time_scalerbar_handle_prelight_color*; the color of the handle of the time scalerbar when the mouse cursor is on the handle

GdkColor *time_scalerbar_handle_selected_color*; the color of the handle of the time scalerbar when the handle is clicked

GdkColor *button_bg_color*; the background color of the buttons

GdkColor *button_prelight_color*; the color of the buttons when the mouse cursor is on them

GdkColor *button_active_color*; the color of the buttons when they are clicked

GdkColor *listview_base_normal_color*; the background color of the playlist views

GdkColor *listview_base_selected_color*; the color of the cursors in the playlist views

GdkColor *listview_base_active_color*; the color of the cursors in the playlist views when the view is focused

GdkColor *listview_font_normal_color*; the color of the texts in the playlist views

GdkColor *listview_font_selected_color*; the color of the texts in the playlist views when the item is selected

GdkColor *listview_font_active_color*; the color of the texts in the playlist views when the item is selected and the view is focused

GdkColor *listview_scrbar_color*; the color of the scroll bars in the playlist views

rc_gui_style_init ()

```
void                rc_gui_style_init                   ();
```

Initialize the theme style of the player window. Can be used only once.

rc_gui_style_refresh ()

```
void                rc_gui_style_refresh                ();
```

Apply the style configuration in the player settings.

rc_gui_style_set_color_style ()

```
void                rc_gui_style_set_color_style        (const RCGuiColorStyle *style_data) ←
    ;
```

Set the color style of the player by given data.

style_data **:** the color style data to set

rc_gui_style_get_color_style ()

```
const RCGuiColorStyle * rc_gui_style_get_color_style    (gint index);
```

Return the embeded color style in the player by given index.

index **:** the index number of the color style

Returns **:** The color style data.

rc_gui_style_set_color_style_by_index ()

```
void                rc_gui_style_set_color_style_by_index
                                                        (gint index);
```

Set the color style of the player to the embeded color style in the player by given index.

index **:** the index number of the color style

2.5 Playlist Views UI

Playlist Views UI — The playlist views of the player.

Synopsis

```
#include <gui_treeview.h>

void                rc_gui_treeview_init                 ();
void                rc_gui_list_tree_reset_list_store   ();
void                rc_gui_select_list1                  (gint list_index);
void                rc_gui_select_list2                  (gint list_index);
void                rc_gui_list1_new_list                ();
void                rc_gui_list1_delete_list             ();
gint                rc_gui_list1_get_selected_index      ();
void                rc_gui_list2_delete_lists            ();
void                rc_gui_list2_select_all              ();
void                rc_gui_list1_rename_list             ();
```

Description

Show the playlist views in the player.

Details

rc_gui_treeview_init ()

```
void                rc_gui_treeview_init                ();
```

Initialize the tree views in the main window. Can be used only once.

rc_gui_list_tree_reset_list_store ()

```
void                rc_gui_list_tree_reset_list_store   ();
```

Reset the playlist views.

rc_gui_select_list1 ()

```
void                rc_gui_select_list1                 (gint list_index);
```

Make the cursor select one item in the playlist by index.

***list_index* :** the index of the item

rc_gui_select_list2 ()

```
void                rc_gui_select_list2                 (gint list_index);
```

Make the cursor select one music in the playlist.

***list_index* :** the index of the item

rc_gui_list1_new_list ()

```
void                    rc_gui_list1_new_list               ();
```

Create a new list with the name the user inputs.

rc_gui_list1_delete_list ()

```
void                    rc_gui_list1_delete_list            ();
```

Delete the playlist the user selected.

rc_gui_list1_get_selected_index ()

```
gint                    rc_gui_list1_get_selected_index     ();
```

Return the index of the selected playlist.

***Returns* :** The index of the selected playlist.

rc_gui_list2_delete_lists ()

```
void                    rc_gui_list2_delete_lists           ();
```

Delete the selected item(s) in the playlist.

rc_gui_list2_select_all ()

```
void                    rc_gui_list2_select_all             ();
```

Select all items in the playlist.

rc_gui_list1_rename_list ()

```
void                    rc_gui_list1_rename_list            ();
```

Rename a list (make the name of the selected playlist editable).

2.6 Mini Mode UI

Mini Mode UI — The mini mode UI of the player.

Synopsis

```
#include <gui_mini.h>

void                rc_gui_mini_init                    ();
RCGuiMiniData *     rc_gui_mini_get_data                ();
void                rc_gui_mini_set_info_text           (const gchar *text);
void                rc_gui_mini_set_lyric_text          (const gchar *text);
void                rc_gui_mini_info_text_move          ();
void                rc_gui_mini_set_lyric_persent       (gdouble persent);
void                rc_gui_mini_set_play_state          (gboolean state);
void                rc_gui_mini_set_time_text           (gint64 pos);
void                rc_gui_mini_window_hide             ();
void                rc_gui_mini_window_show             ();
void                rc_gui_mini_mini_mode_clicked       ();
void                rc_gui_mini_normal_mode_clicked     ();
```

Description

Show the mini mode of the player.

Details

rc_gui_mini_init ()

```
void                rc_gui_mini_init                    ();
```

Initialize the mini mode window of the player. Can be used only once.

rc_gui_mini_get_data ()

```
RCGuiMiniData *     rc_gui_mini_get_data                ();
```

Return the data of mini mode UI structure.

***Returns* :** The data of mini mode UI structure.

rc_gui_mini_set_info_text ()

```
void                rc_gui_mini_set_info_text           (const gchar *text);
```

Set the text of the information label.

***`text`* :** the text which shows on the information label

rc_gui_mini_set_lyric_text ()

```
void                rc_gui_mini_set_lyric_text          (const gchar *text);
```

Set the lyric text of the lyric label.

***`text`* :** the lyric text which shows on the lyric label

rc_gui_mini_info_text_move ()

```
void                rc_gui_mini_info_text_move          ();
```

Make the view of the information label move if the text inside is too loog.

rc_gui_mini_set_lyric_persent ()

```
void                rc_gui_mini_set_lyric_persent       (gdouble persent);
```

Make the view of the lyric label move by given persent if the lyric text is too loog.

***persent* :** the persent position of the lyric text

rc_gui_mini_set_play_state ()

```
void                rc_gui_mini_set_play_state          (gboolean state);
```

Set play button state.

***state* :** the state of the play button, if it’s TRUE, the image of the button is pause icon, else the image is play icon.

rc_gui_mini_set_time_text ()

```
void                rc_gui_mini_set_time_text           (gint64 pos);
```

Set time label.

***pos* :** the time to set, in nanosecond.

rc_gui_mini_window_hide ()

```
void                rc_gui_mini_window_hide             ();
```

Hide the mini mode window.

rc_gui_mini_window_show ()

```
void                rc_gui_mini_window_show             ();
```

Show the mini mode window.

rc_gui_mini_mini_mode_clicked ()

```
void                rc_gui_mini_mini_mode_clicked       ();
```

Enable mini mode.

rc_gui_mini_normal_mode_clicked ()

```
void                rc_gui_mini_normal_mode_clicked     ();
```

Return to normal mode.

2.7 Settings UI

Settings UI — The settings window of the player.

Synopsis

```
#include <gui_setting.h>

void                rc_gui_create_setting_window        ();
```

Description

Show the settings window of the player.

Details

rc_gui_create_setting_window ()

```
void                rc_gui_create_setting_window        ();
```

Show a setting window.

2.8 Plugin Configuration UI

Plugin Configuration UI — The plugin configuration window of the player.

Synopsis

```
#include <gui_plugin.h>

void                rc_gui_plugin_window_create         ();
```

Description

Show the plugin configuration window of the player.

Details

rc_gui_plugin_window_create ()

```
void                rc_gui_plugin_window_create         ();
```

Show the plugin configuration window.

Part II

Plugin Programming Manual

This part describes how to write a plugin for RhythmCat Music Player.

Chapter 3

Plugin Programming Manual

RhythmCat Music Player uses plugin support system to expand its functions and features. This manual can help you implement a plugin for the player.

Introduction

The plugin used in the player now is module, it is usually a shared-object (on Linux) or a dynamic linked library (on Windows). When the player loads the plugin, it will load it into memory, then execute the initialize function in the module. GLib provides the feature "Dynamic Loading of Modules", which is called "GModule" in the library. This player uses this feature to provide plugin support.

Preparation

Before you prepare to write the plugin, you should install development environment. GLib 2.0, GTK+ 2.0 and Gstreamer 0.10 and their development packages, and the other libraries you needed in your plugin are necessary to install.

Implement a simple plugin

The plugin needs the libraries to work, so you should include the header files. You can include them by the codes below:

```
#include <glib.h>
#include <gst/gst.h>
#include <gtk/gtk.h>
#include "plugin.h"
```

The interfaces/functions needed by the player in your plugin are g_module_check_init(), g_module_unload(), rc_plugin_module_init(), rc_plugin_module_exit(), rc_plugin_module_data(), and rc_plugin_module_configure() is optional. Once the plugin is loaded, g_module_check_init() will be called automatically, you can write the codes you want to initialize the configuration data in the function. If the plugin is about to exit, g_module_unload() will be called automatically, you can write the codes in the function to free all memory you have allocated. When the player is about to enable the plugin, rc_plugin_module_init() will be called, you can finish the function to implement your plugin. When the player tries to disable the plugin, rc_plugin_module_exit() will be called, you can finish the function to exit from your plugin. The player also needs some information about the plugin, it will call rc_plugin_module_data() to get the data, you should finish it by returning the information of your plugin. The definition of these functions are:

```
/* Necessary */
const gchar *g_module_check_init(GModule *module);
void g_module_unload(GModule *module);
gint rc_plugin_module_init();
void rc_plugin_module_exit();
const RCPluginModuleData *rc_plugin_module_data();
/* Optional */
void rc_plugin_module_configure();
```

The definition of type RCPluginModuleData is in header file "plugin.h":

```
typedef struct RCPluginModuleData {
    gchar *group_name;
    gboolean resident;
}RCPluginModuleData;
```

The element group_name is used in the configuration file, the player needs it the get the configuration data of the plugin by the group name, it should be unique, or the configuration data may conflict with other plugins. The element resident decides whether the plugin can be removed while the player is running, set it to TRUE to prevent the player from removing it.

If the plugin needs configuration, the function rc_plugin_module_configure() should be implemented. It will be called when the user tries to configure the plugin. You can show a configure dialog to make the user configure the features in the plugin. If you need to load/save the configuration data, you should use the function rc_set_get_plugin_configure() (in header file "settings.h"), it will return the GKeyFile pointer of the plugin configuration data. You can read/write your configuration by using "Key-value file parser" in GLib. Notice that you should only read/write the configuration which is related to your plugin (by the group name).

Here is an example plugin implemented below:

Example 3.1 The example plugin

```
#include <glib.h>
#include <glib/gprintf.h>
#include <gst/gst.h>
#include <gtk/gtk.h>
#include "plugin.h"

static RCPluginModuleData plugin_module_data =
{
    "ExamplePlugin", /* group_name */
    FALSE /* resident */
};

static GKeyFile *keyfile = NULL;

const gchar *g_module_check_init(GModule *module)
{
    g_printf("ExamplePlugin: Plugin loaded successfully!\n");
    keyfile = rc_set_get_plugin_configure();

    /* Change configuration data like below: */
    /* Save configuration string "HelloData" to key "TestString". */
    g_key_file_set_string(keyfile, plugin_module_data.group_name,
        "TestString", "HelloData");
    /* If you want to set other data types, please see the GKeyFile section
       in GLib. */

    /* Implement more initialize functions here. */

    return NULL;
    /* If there is no error, return NULL, else return the error string. */
}

void g_module_unload(GModule *module)
{
    g_printf("ExamplePlugin: Plugin unloaded!\n");
    /* Do some cleaning work here. */
}

gint rc_plugin_module_init()
{
    g_printf("ExamplePlugin: Plugin is running!\n");

    /* Get configuration data like below: */
    /* Load configration string from key "TestString". */
    gchar *string = g_key_file_get_string(keyfile,
        plugin_module_data.group_name, "TestString", NULL);
    g_printf("TestString=%s\n", string); /* Show the string. */
    g_free(string); /* Remember to free pointer after usage. */

    /* Implement plugin features here. */

    return 0;
    /* If there is no error, return 0, else return the error code. */
}

void rc_plugin_module_exit()
{
    g_printf("ExamplePlugin: Plugin is not running now!\n");
    /* Disable all features of the plugin, and do some cleanning work. */
}

void rc_plugin_module_configure()
{
    /* Show configure dialog if necessary. */
}

const RCPluginModuleData *rc_plugin_module_data()
{
```

How to compile the plugin

The plugin is a module, so you have to compile it into a library. On Linux, it is shared-object(.so). It is recommended to write a Makefile to compile the plugin, or use autoconf/automake.

Here is the Makefile of the example plugin mentioned above (the source file name is example.c).

```
CC=gcc
PLUGIN_NAME=example.so
INCS=
SRCS=example.c
OBJS=${SRCS:.c=.o}

LIBS=glib-2.0 gtk+-2.0 gstreamer-0.10

CFLAGS=‘pkg-config --cflags ${LIBS}‘ -Wall -O2 -fPIC -I../../src/
LDFLAGS=‘pkg-config --libs ${LIBS}‘ -Wall -O2 -shared -fPIC

all: ${PLUGIN_NAME}

${PLUGIN_NAME}:${OBJS}
  ${CC} -o ${PLUGIN_NAME} ${OBJS} ${LDFLAGS}

${OBJS}:${INCS}

.c.o:
  ${CC} -c $< ${CFLAGS}

clean:
  rm -f *.o ${PLUGIN_NAME}

rebuild: clean all
```

This Makefile will produce shared-object example.so when it is executed.

Write a plugin description file

Except the library file, the player also needs the description file to get necessary information about the plugin. You need to write a description file to make your plugin usable by the player.

Here is the description file of the example plugin mentioned above.

Example 3.2 The description file of the example plugin

```
[RC Plugin] # The group name must be "RC Plugin".
Type=Module # Must be "Module" now.
File=example.so # The filename of the plugin library file.
Name=Example Plugin # The name of the plugin
Description=This is an example plugin. # The description of the plugin.
Authors=SuperCat <supercatexpert@gmail.com> # The author information.
Version=0.1 # The version
Website=http://code.google.com/p/rhythmcat # The website.
```

Make your custom plugin work

Now you may have implemented the plugin, then you should put the files into the right directory. The player needs two files to make the plugin work, one is the library file, the other is the description file, and put them into a director. The directory which contains the two files should be named with the filename of the description file (without extension name), and then put it into the plugin directory. There are two places where you can put the plugin directory, one is under $APP_DIR/plugins/ ($APP_DIR is the directory where the program data is), the other is under ~/.RhythmCat/Plugins/. The plugin directory may be descripted like this:

```
ExamplePlugin
----ExamplePlugin.conf # The description file.
----example.so # The plugin library file.
```

Then put directory "ExamplePlugin" into $APP_DIR/plugins/ or ~/.RhythmCat/Plugins/

Catch signals in the player

Sometimes we need signals to know the working state of the player. When the player starts to play, or being stopped, it will emit signals. The way to catch these signals is to connect the signals to the callback functions you implemented. If you want to use signals, you should include header file `"player_object.h"` first, then you can connect the signals to your callback functions by using rc_player_object_signal_connect_simple(). The signal names are provided in object RCPlayer. This function will return the signal ID, and when you uninitialize your plugin, please remember remove the signal first, by using function rc_player_object_signal_disconnect().

About this manual

This manual helps you to make a usable plugin for RhythmCat Music Player. If you have any questions or problems, please leave issues on the project homepage, or send an e-mail to me. I also have implemented some plugins for the player, you can view the source codes of them for reference.

Chapter 4

Object Hierarchy

```
    GObject
        RCPlayer
```

Chapter 5

Index