RhythmCat Reference Manual

COLLABORATORS			
	TITLE : RhythmCat Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 22, 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

[Insert title here]

1.1 plugin

plugin —

Synopsis

```
enum                RCPluginType;
gint                (*module_init)                      ();
void                (*module_exit)                      ();
const gchar *       (*module_get_group_name)            ();
gboolean            rc_plugin_init                      ();
void                rc_plugin_exit                      ();
gboolean            rc_plugin_search_dir                (const gchar *Param1);
const GSList *      rc_plugin_get_list                  ();
void                rc_plugin_list_free                 ();
void                rc_plugin_plugin_free               (RCPluginData *Param1);
void                rc_plugin_module_free               (RCModuleData *Param1);
gboolean            rc_plugin_module_check_running      (const gchar *Param1);
gboolean            rc_plugin_load                      (const gchar *Param1,
                                                         RCPluginData **Param2);
gboolean            rc_plugin_module_load               (const gchar *Param1);
void                rc_plugin_module_close              (const gchar *Param1);
gboolean            rc_plugin_module_configure          (const gchar *Param1);
gboolean            rc_plugin_python_load               (const gchar *Param1);
gboolean            rc_plugin_python_configure          (const gchar *Param1);
```

Description

Details

enum RCPluginType

```
typedef enum RCPluginType
{
    PLUGIN_TYPE_MODULE = 1,
    PLUGIN_TYPE_PYTHON = 2
}RCPluginType;
```

module_init ()

```
gint                (*module_init)                      ();
```

module_exit ()

```
void                (*module_exit)                      ();
```

module_get_group_name ()

```
const gchar *       (*module_get_group_name)            ();
```

rc_plugin_init ()

```
gboolean            rc_plugin_init                      ();
```

rc_plugin_exit ()

```
void                rc_plugin_exit                      ();
```

rc_plugin_search_dir ()

```
gboolean            rc_plugin_search_dir                (const gchar *Param1);
```

rc_plugin_get_list ()

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

rc_plugin_list_free ()

```
void                rc_plugin_list_free                 ();
```

rc_plugin_plugin_free ()

```
void                rc_plugin_plugin_free               (RCPluginData *Param1);
```

rc_plugin_module_free ()

```
void                rc_plugin_module_free               (RCModuleData *Param1);
```

rc_plugin_module_check_running ()

```
gboolean            rc_plugin_module_check_running      (const gchar *Param1);
```

rc_plugin_load ()

```
gboolean            rc_plugin_load                      (const gchar *Param1,
                                                         RCPluginData **Param2);
```

rc_plugin_module_load ()

```
gboolean            rc_plugin_module_load               (const gchar *Param1);
```

rc_plugin_module_close ()

```
void                rc_plugin_module_close              (const gchar *Param1);
```

rc_plugin_module_configure ()

```
gboolean            rc_plugin_module_configure          (const gchar *Param1);
```

rc_plugin_python_load ()

```
gboolean            rc_plugin_python_load               (const gchar *Param1);
```

rc_plugin_python_configure ()

```
gboolean            rc_plugin_python_configure          (const gchar *Param1);
```

1.2 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.

1.3 Lyric

Lyric — Process lyric data.

Synopsis

```
#include <lyric.h>

                    RCLyricData;
gboolean            rc_lrc_read_from_file               (const gchar *filename);
void                rc_lrc_clean_data                   ();
const GList *       rc_lrc_get_lrc_data                 ();
const gchar *       rc_lrc_get_text_data                ();
const RCLyricData * rc_lrc_get_line_by_time             (gint64 time);
```

Description

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

Details

RCLyricData

```
typedef struct {
    guint64 time;
    guint64 length;
    gchar *text;
} RCLyricData;
```

Custom struct type to store the data of lyrics.

guint64 ***time*****;** the start time of the lyric text

guint64 ***length*****;** the time length of the lyric text

gchar ****text*****;** the lyric text

rc_lrc_read_from_file ()

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

rc_lrc_clean_data ()

```
void                rc_lrc_clean_data                   ();
```

Clean the read lyric data from the player.

rc_lrc_get_lrc_data ()

```
const GList *       rc_lrc_get_lrc_data                 ();
```

Return the processed lyric data in the player.

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

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.

1.4 gui_plugin

gui_plugin —

Synopsis

```
void                rc_gui_plugin_window_create         (GtkWidget *Param1,
                                                         gpointer Param2);
```

Description

Details

rc_gui_plugin_window_create ()

```
void                rc_gui_plugin_window_create         (GtkWidget *Param1,
                                                         gpointer Param2);
```

1.5 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.

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

1.6 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.

1.7 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 load 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.

1.8 gui_dialog

gui_dialog —

Synopsis

```
#define             MAX_DIR_DEPTH
void                rc_gui_about_player                 ();
void                rc_gui_show_message_dialog          (GtkMessageType Param1,
                                                         const gchar *Param2,
                                                         const gchar *Param3,
                                                         ...);
void                rc_gui_show_open_dialog             (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_open_music_directory         (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_save_playlist_dialog         (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_load_playlist_dialog         (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_save_all_playlists_dialog    (GtkWidget *Param1,
                                                         gpointer Param2);
```

Description

Details

MAX_DIR_DEPTH

```
#define MAX_DIR_DEPTH 5
```

rc_gui_about_player ()

```
void                rc_gui_about_player                 ();
```

rc_gui_show_message_dialog ()

```
void                rc_gui_show_message_dialog          (GtkMessageType Param1,
                                                         const gchar *Param2,
                                                         const gchar *Param3,
                                                         ...);
```

rc_gui_show_open_dialog ()

```
void                rc_gui_show_open_dialog             (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_open_music_directory ()

```
void                rc_gui_open_music_directory         (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_save_playlist_dialog ()

```
void                rc_gui_save_playlist_dialog         (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_load_playlist_dialog ()

```
void                rc_gui_load_playlist_dialog         (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_save_all_playlists_dialog ()

```
void                rc_gui_save_all_playlists_dialog    (GtkWidget *Param1,
                                                         gpointer Param2);
```

1.9 gui_eq

gui_eq —

Synopsis

```
void                rc_gui_init_eq_data             ();
void                rc_gui_eq_init                  ();
void                rc_gui_create_equalizer         ();
RCGuiEQData *       rc_gui_eq_get_data              ();
```

Description

Details

rc_gui_init_eq_data ()

```
void                rc_gui_init_eq_data             ();
```

rc_gui_eq_init ()

```
void                rc_gui_eq_init                  ();
```

rc_gui_create_equalizer ()

```
void                rc_gui_create_equalizer         ();
```

rc_gui_eq_get_data ()

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

1.10 gui_style

gui_style —

Synopsis

```
void                rc_gui_style_init               ();
void                rc_gui_style_refresh            ();
void                rc_gui_style_set_color_style    (const RCGuiColorStyle *Param1);
const RCGuiColorStyle * rc_gui_style_get_color_style    (gint Param1);
void                rc_gui_style_set_color_style_by_index
                                                    (gint Param1);
```

Description

Details

rc_gui_style_init ()

```
void                rc_gui_style_init                   ();
```

rc_gui_style_refresh ()

```
void                rc_gui_style_refresh                ();
```

rc_gui_style_set_color_style ()

```
void                rc_gui_style_set_color_style        (const RCGuiColorStyle *Param1);
```

rc_gui_style_get_color_style ()

```
const RCGuiColorStyle * rc_gui_style_get_color_style    (gint Param1);
```

rc_gui_style_set_color_style_by_index ()

```
void                rc_gui_style_set_color_style_by_index
                                                        (gint Param1);
```

1.11 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.

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.

1.12 gui_treeview

gui_treeview —

Synopsis

```
void                rc_gui_treeview_init                (();
void                rc_gui_list_tree_reset_list_store   ();
gboolean            rc_gui_list1_popup_menu             (GtkWidget *Param1,
                                                         GdkEventButton *Param2,
                                                         gpointer Param3);
gboolean            rc_gui_list2_popup_menu             (GtkWidget *Param1,
                                                         GdkEventButton *Param2,
                                                         gpointer Param3);
gboolean            rc_gui_list2_button_release_event   (GtkWidget *Param1,
                                                         GdkEventButton *Param2,
                                                         gpointer Param3);
void                rc_gui_list1_row_selected           (GtkTreeView *Param1,
                                                         gpointer Param2);
void                rc_gui_list2_row_activated          (GtkTreeView *Param1,
                                                         GtkTreePath *Param2,
                                                         GtkTreeViewColumn *Param3,
                                                         gpointer Param4);
void                rc_gui_select_list1                 (gint Param1);
void                rc_gui_select_list2                 (gint Param1);
void                rc_gui_list1_new_list               (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_list1_delete_list            (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_list_model_inserted          (GtkTreeModel *Param1,
                                                         GtkTreePath *Param2,
                                                         GtkTreeIter *Param3,
                                                         gpointer Param4);
gint                rc_gui_list1_get_index              (GtkTreeIter *Param1);
gint                rc_gui_list1_get_selected_index     ();
void                rc_gui_list2_dnd_data_received      (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         gint Param3,
                                                         gint Param4,
                                                         GtkSelectionData *Param5,
                                                         guint Param6,
                                                         guint Param7,
                                                         gpointer Param8);
void                rc_gui_list2_dnd_data_get           (GtkWidget *Param1,
```

```
                                                GdkDragContext *Param2,
                                                GtkSelectionData *Param3,
                                                guint Param4,
                                                guint Param5,
                                                gpointer Param6);
void                rc_gui_list2_dnd_motion             (GtkWidget *Param1,
                                                GdkDragContext *Param2,
                                                gint Param3,
                                                gint Param4,
                                                guint Param5,
                                                gpointer Param6);
void                rc_gui_list1_dnd_data_received      (GtkWidget *Param1,
                                                GdkDragContext *Param2,
                                                gint Param3,
                                                gint Param4,
                                                GtkSelectionData *Param5,
                                                guint Param6,
                                                guint Param7,
                                                gpointer Param8);
void                rc_gui_list1_dnd_data_get           (GtkWidget *Param1,
                                                GdkDragContext *Param2,
                                                GtkSelectionData *Param3,
                                                guint Param4,
                                                guint Param5,
                                                gpointer Param6);
void                rc_gui_list2_delete_lists           (GtkWidget *Param1,
                                                gpointer Param2);
void                rc_gui_list2_select_all             (GtkWidget *Param1,
                                                gpointer Param2);
void                rc_gui_list1_edited                 (GtkCellRendererText *Param1,
                                                gchar *Param2,
                                                gchar *Param3,
                                                gpointer Param4);
void                rc_gui_list1_rename_list            (GtkWidget *Param1,
                                                gpointer Param2);
```

Description

Details

rc_gui_treeview_init ()

```
void                rc_gui_treeview_init                ();
```

rc_gui_list_tree_reset_list_store ()

```
void                rc_gui_list_tree_reset_list_store   ();
```

rc_gui_list1_popup_menu ()

```
gboolean            rc_gui_list1_popup_menu             (GtkWidget *Param1,
                                                GdkEventButton *Param2,
                                                gpointer Param3);
```

rc_gui_list2_popup_menu ()

```
gboolean            rc_gui_list2_popup_menu             (GtkWidget *Param1,
                                                         GdkEventButton *Param2,
                                                         gpointer Param3);
```

rc_gui_list2_button_release_event ()

```
gboolean            rc_gui_list2_button_release_event   (GtkWidget *Param1,
                                                         GdkEventButton *Param2,
                                                         gpointer Param3);
```

rc_gui_list1_row_selected ()

```
void                rc_gui_list1_row_selected           (GtkTreeView *Param1,
                                                         gpointer Param2);
```

rc_gui_list2_row_activated ()

```
void                rc_gui_list2_row_activated          (GtkTreeView *Param1,
                                                         GtkTreePath *Param2,
                                                         GtkTreeViewColumn *Param3,
                                                         gpointer Param4);
```

rc_gui_select_list1 ()

```
void                rc_gui_select_list1                 (gint Param1);
```

rc_gui_select_list2 ()

```
void                rc_gui_select_list2                 (gint Param1);
```

rc_gui_list1_new_list ()

```
void                rc_gui_list1_new_list               (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_list1_delete_list ()

```
void                rc_gui_list1_delete_list            (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_list_model_inserted ()

```
void                rc_gui_list_model_inserted          (GtkTreeModel *Param1,
                                                         GtkTreePath *Param2,
                                                         GtkTreeIter *Param3,
                                                         gpointer Param4);
```

rc_gui_list1_get_index ()

```
gint                rc_gui_list1_get_index              (GtkTreeIter *Param1);
```

rc_gui_list1_get_selected_index ()

```
gint                rc_gui_list1_get_selected_index     ();
```

rc_gui_list2_dnd_data_received ()

```
void                rc_gui_list2_dnd_data_received      (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         gint Param3,
                                                         gint Param4,
                                                         GtkSelectionData *Param5,
                                                         guint Param6,
                                                         guint Param7,
                                                         gpointer Param8);
```

rc_gui_list2_dnd_data_get ()

```
void                rc_gui_list2_dnd_data_get           (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         GtkSelectionData *Param3,
                                                         guint Param4,
                                                         guint Param5,
                                                         gpointer Param6);
```

rc_gui_list2_dnd_motion ()

```
void                rc_gui_list2_dnd_motion             (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         gint Param3,
                                                         gint Param4,
                                                         guint Param5,
                                                         gpointer Param6);
```

rc_gui_list1_dnd_data_received ()

```
void                rc_gui_list1_dnd_data_received      (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         gint Param3,
                                                         gint Param4,
                                                         GtkSelectionData *Param5,
                                                         guint Param6,
                                                         guint Param7,
                                                         gpointer Param8);
```

rc_gui_list1_dnd_data_get ()

```
void                rc_gui_list1_dnd_data_get           (GtkWidget *Param1,
                                                         GdkDragContext *Param2,
                                                         GtkSelectionData *Param3,
                                                         guint Param4,
                                                         guint Param5,
                                                         gpointer Param6);
```

rc_gui_list2_delete_lists ()

```
void                rc_gui_list2_delete_lists           (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_list2_select_all ()

```
void                rc_gui_list2_select_all             (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_list1_edited ()

```
void                rc_gui_list1_edited                 (GtkCellRendererText *Param1,
                                                         gchar *Param2,
                                                         gchar *Param3,
                                                         gpointer Param4);
```

rc_gui_list1_rename_list ()

```
void                rc_gui_list1_rename_list            (GtkWidget *Param1,
                                                         gpointer Param2);
```

1.13 GUI

GUI — 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 Param1);
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 struct type to store the UI data. Please do not change the data in this struct.

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.

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 Param1);
```

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.

1.14 Mini Mode GUI

Mini Mode GUI — 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.

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.

***time*:** 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.

1.15 gui_setting

gui_setting —

Synopsis

```
void                rc_gui_create_setting_window        (GtkWidget *Param1,
                                                         gpointer Param2);
void                rc_gui_create_setting_treeview      ();
void                rc_gui_close_setting_window         (GtkButton *Param1,
                                                         gpointer Param2);
void                rc_gui_setting_row_selected         (GtkTreeView *Param1,
                                                         gpointer Param2);
void                rc_gui_setting_apply                (GtkButton *Param1,
                                                         gpointer Param2);
void                rc_gui_setting_confirm              (GtkButton *Param1,
                                                         gpointer Param2);
void                rc_gui_create_setting_general       ();
void                rc_gui_create_setting_playlist      ();
void                rc_gui_create_setting_appearance    ();
```

Description

Details

rc_gui_create_setting_window ()

```
void                rc_gui_create_setting_window        (GtkWidget *Param1,
                                                         gpointer Param2);
```

rc_gui_create_setting_treeview ()

```
void                rc_gui_create_setting_treeview      ();
```

rc_gui_close_setting_window ()

```
void                rc_gui_close_setting_window         (GtkButton *Param1,
                                                         gpointer Param2);
```

rc_gui_setting_row_selected ()

```
void                rc_gui_setting_row_selected         (GtkTreeView *Param1,
                                                         gpointer Param2);
```

rc_gui_setting_apply ()

```
void                rc_gui_setting_apply            (GtkButton *Param1,
                                                     gpointer Param2);
```

rc_gui_setting_confirm ()

```
void                rc_gui_setting_confirm          (GtkButton *Param1,
                                                     gpointer Param2);
```

rc_gui_create_setting_general ()

```
void                rc_gui_create_setting_general   ();
```

rc_gui_create_setting_playlist ()

```
void                rc_gui_create_setting_playlist  ();
```

rc_gui_create_setting_appearance ()

```
void                rc_gui_create_setting_appearance ();
```

1.16 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 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_string_list ()

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

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.

Chapter 2

Object Hierarchy

Chapter 3

Index