Multimedia Playlist Library Reference

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: March 31, 2015

Contents

About This Reference

The *Multimedia Playlist Library Reference* is aimed at developers who want to write applications that use the **libmmplaylist** library to read playlist files and provide users with playlist functions such as track selection and forward or backward movement of the playback position.

This table may help you find what you need in this reference:

To find out about:	Go to:
The purpose and capabilities of **libmmplaylist**	*Multimedia Playlist Library Overview* (p. 9)
The list of Playlist Plugins (PLPs) included with **libmmplaylist**	*Included plugins* (p. 12)
The **libmmplaylist** configuration file, which lists the library paths and configuration settings of supported plugins	*Configuration File* (p. 15)
Using the Multimedia Playlist API to establish playlist sessions, seek to playlist positions, retrieve metadata, and obtain error information	*Multimedia Playlist API* (p. 17)

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	**unsigned short**
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

WARNING: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1 Multimedia Playlist Library Overview

The multimedia playlist library, **libmmplaylist**, reads playlists on media devices and allows clients to seek to and play individual tracks within playlists.

Playlists are track sequences that store track metadata and ordering information. Playlists come in many different formats and typically contain either the URL of a source media stream or a set of URLs or filepaths of individual tracks.

By supporting playlists, your media applications can:

- play multiple tracks in sequence
- navigate between or within tracks
- skip or repeat tracks

Other multimedia services use **libmmplaylist** to support playlists. For instance, the **mm-sync** service uses this library to synchronize playlists and even allows users to define a custom configuration for **libmmplaylist**. Also, the `Playlist engine` plugin in **mm-renderer** uses the library to manage playback when users attach an input whose type is `"playlist"`.

Media applications can use **mm-renderer** features such as repeat and position seek for basic playlist management. However, if you want greater control over playlists, you can write applications that use **libmmplaylist** directly. The library provides functionality to:

- explore playlists from the HMI without having to synchronize them to databases
- parse playlists to locate and copy media files that store either individual tracks or playlist information to persistent storage
- seek to relative or absolute playlist positions
- obtain summary information on a group of related tracks, such as the total runtime for an album

The **libmmplaylist** library provides a common, high-level interface for supporting playback and track seeking in playlists of different formats. With this design, application writers must learn only one set of commands to manage many types of playlists.

Architecture of libmmplaylist

The **libmmplaylist** library uses a plugin architecture in which each plugin can manage a particular playlist format. When a client opens a session on a playlist, the library determines which plugin is most suited to manage the playlist and uses that selected plugin to carry out subsequent playlist operations.

The library is implemented in three layers:

Playlist management

This layer:

- initializes the library by loading the character-converter service and determining the path of the configuration file
- opens playlist sessions and returns session handles
- loads and validates playlist entries and performs character encoding
- reports the number of playlist entries and the position (index) of the currently playing entry
- updates the playlist position after validating the new position requested by the client

Plugin management

This layer:

- reads the configuration file to learn the plugin filenames and configuration settings
- loads, validates, and unloads plugins
- determines which plugins support a given playlist and ranks those plugins
- provides configuration settings to the playlist management layer to help it perform character encoding

Plugins

This layer consists of many playlist plugins that:

- rate themselves on their ability to support a particular playlist
- open and close playlist sessions when requested by the playlist management layer
- provide basic operations for navigating and retrieving information from playlists
- may provide more efficient methods for some navigation and seek operations

The plugin-based architecture makes it easy for future releases of **libmmplaylist** to support additional playlist formats while clients continue to use the same commands to manage playlists.

Plugins

Playlist plugins (PLPs) are **libmmplaylist** components that manage playlists of specific formats. PLPs abstract the parsing of playlists by implementing a standard set of functions that higher layers of **libmmplaylist** can call to navigate and read information from playlists.

When the playlist management layer forwards a user request to open a session on a playlist, the selected plugin invokes the appropriate lower-level service to access and begin parsing the playlist. Depending on the playlist format, this service could be an XML reader, a database engine, or a media streamer. The plugin stores the parser state in an internal structure. This way, the plugin can resume reading the playlist from the same location in the file or database result set that it read up to in the last playlist operation. Common operations such as retrieving the next playlist entry (track) can be implemented more efficiently because the plugin doesn't have to read through the entire playlist each time the track changes.

All PLPs implement functions to:

- open and close playlist sessions
- rewind a playlist to the beginning
- move to the next entry and return its information
- rate themselves on their ability to manage a particular playlist

Some PLPs also implement functions to:

- report the playlist's defined character encoding to the playlist management layer
- provide fast methods for getting the number of playlist entries and for getting and setting the playlist position
- report detailed information on the last error encountered (useful for debugging)

Plugin ratings

When the user opens a session on a playlist, **libmmplaylist** queries all available PLPs for their ratings on managing the specified playlist. The “available” PLPs are those that were successfully loaded during library initialization. Their ratings measure their individual abilities to manage the playlist that the user is opening. PLP ratings range from 0, which means the plugin doesn't support the playlist, to 100, which means the plugin is a perfect choice for managing it.

Typically, a plugin examines the playlist extension and if that extension indicates a format that it supports, the plugin returns its preset, nonzero rating to indicate that it can parse the playlist. If the extension indicates an unsupported format, the plugin returns a rating of 0.

When **libmmplaylist** has obtained all the PLP ratings, it sorts the PLPs from highest- to lowest-rated, discarding references to PLPs with a rating of 0. The library then goes through the sorted list and tries to open a session with each plugin in turn. This way, **libmmplaylist** picks the highest-rated plugin to handle all operations on that same playlist.

The **libmmplaylist** library assigns default ratings for all PLPs but you can overwrite these settings in the *configuration file* (p. 15).

Included plugins

The **libmmplaylist** library is shipped with many plugins capable of parsing various playlist formats:

asx

Files of the following types:

- Microsoft Advanced Streaming Redirector (**.asx**) files
- Windows Media Audio Redirector (**.wax**) files
- Windows Media Video Redirector (**.wvx**) files

b4s

Playlist files for WinAmp versions 3 and later (**.b4s** files)

filelist

String listings of media files, with entries separated by configurable delimiters

iTunes

XML files created by iTunes (**iTunes Library.xml**)

m3u

MP3 playlist (**.m3u**) files

mediafs

Playlist files that can be converted into directories containing references to the real media files (e.g., MP3 files). This plugin supports the **.pla** extension.

pls

Text-based playlist (**.pls**) files

qdb

Result sets from SQL queries made against QDB databases

rmp

RealAudio audio (**.rmp**) files

wpl

Windows Media Player Playlist (**.wpl**) files

xspf

XML Shareable Playlist Format (**.xspf**) files

Playlist sessions

To manage a playlist, a client must open a session on the playlist before it can seek to and play tracks or extract metadata.

To open a playlist session, the client must name the media that stores the playlist and provide the playlist's path. The client may also instruct **libmmplaylist** to convert the playlist entries to a certain character encoding or to validate the entries. This latter task refers to how the library identifies files that correspond to playlist entries; see the *mmplaylist_open()* (p. 33) function for details on specifying entry validation.

When **libmmplaylist** successfully opens a playlist session, the *mmplaylist_open()* call returns a session handle. Clients must provide this handle in subsequent **libmmplaylist** API calls to operate on the same playlist (e.g., to seek to different positions with *mmplaylist_current_pos_get()* (p. 19)).

The session handle contains a reference to the plugin chosen to manage the playlist, which allows the library to reuse that same plugin for all playlist operations. The handle also stores the preferences for encoding and validation as well as a reference (if applicable) to the active character converter used for encoding playlist entries. Storing these preferences in the session handle allows **libmmplaylist** to support multiple playlists concurrently because the preferences of individual sessions are stored separately and hence, they can differ from each other.

The encoding and validation preferences are used in retrieving the next playlist entry when the client calls *mmplaylist_entry_next_get()* (p. 21). Sometimes, **libmmplaylist** can return playlist entries with a different encoding than what's defined in the preferences. For instance, **m3u** playlists are text files that don't have an official character set. These files are often encoded with the ISO-8859-1 character set; however, in many geographic regions, **m3u** files use alternative encodings. Because **m3u** doesn't have a mechanism for communicating the encoding used, the `m3u` plugin might report the encoding as ISO-8859-1 while the actual encoding differs.

When it's finished using a playlist, the client must close the playlist session by calling *mmplaylist_close()* (p. 18).

Concurrent sessions

Clients can open as many concurrent playlist sessions in **libmmplaylist** as they like, including different sessions on the same playlist (by making distinct *mmplayist_open()* calls with the same playlist path). Multiple concurrent sessions allow media applications to support multitasking with playlists. For example, an application can display browsing information for all the playlists on a device while at the same copying or playing tracks from one of those playlists.

CAUTION: Multithreaded clients can open and use common playlist sessions from different threads. However, clients that share session handles between threads must carefully guard against concurrent access to those handles. For example, if two threads call *mmplaylist_entry_next_get()* on the same session, the playlist position gets incremented once for each call, meaning that one thread could alter the playlist state for the other thread.

Obtaining error information

While a session is active, the client can call *mmplaylist_last_error_get()* (p. 31) to obtain the numeric error code of the last error that occurred on that session. We recommend that your client check the return values of all API calls. If any value indicates an error, the client can retrieve the error code and use it to help recover.

Chapter 2
Configuration File

The **libmmplaylist** configuration file lists the playlist plugins (PLPs) that the library can use to manage playlists as well as the configuration settings for those plugins.

The **libmmplaylist** library is shipped with a default configuration file. You can modify this included file or create your own. When calling *mmplaylist_init()* (p. 30) to initialize the library, your client must supply either the full path of another configuration file or a path of NULL to use the default file. In this second case, the library first searches the path given in the MM_PLAYLIST_CONFIG environment variable or if this variable isn't defined, the library searches the hardcoded default path of **/etc/mm/mm-playlist.conf**.

Redefining MM_PLAYLIST_CONFIG lets you use a different configuration file as the default. This is useful when launching applications such as **mm-renderer** that use **libmmplaylist** but don't allow you to set the configuration path.

In any configuration file, each section defines settings for an individual PLP and must begin with a line like this:

`[plugin]`

The PLP settings are listed on the lines that follow, one per line. A `dll` setting is required in every section to name the library file that implements the PLP. A setting is specified by stating a field name, followed by an equal sign (`=`), followed by the field value. For example, the following lines name the library file and assign a rating of 20 for the `iTunes` plugin:

```
[plugin]
dll=mm-plp-itunes.so
rating=20
```

You can also place comments in the configuration file by starting lines with the number sign (`#`).

Default configuration file

The contents of the default configuration file look like this:

```
# libmmplaylist config file

[plugin]
dll=mm-plp-qdb.so

[plugin]
dll=mm-plp-m3u.so

[plugin]
dll=mm-plp-asx.so
```

```
[plugin]
dll=mm-plp-itunes.so

[plugin]
dll=mm-plp-pls.so

[plugin]
dll=mm-plp-rmp.so

[plugin]
dll=mm-plp-wpl.so

[plugin]
dll=mm-plp-xspf.so

[plugin]
dll=mm-plp-b4s.so

[plugin]
dll=mm-plp-mediafsdir.so

[plugin]
dll=mm-plp-filelist.so
# The filelist delimiter can be changed here noting that
# characters starting with '\' will be converted to
# their escaped equivalent if they exist.  If \xZZ is
# used, the ASCII value for the two hexadecimal digits
# following the 'x' character will be used. Because PPS
# treats newlines as a special character, \n will not
# be escaped. Examples:
#
# delimiter=\x1e -- Hex character 0x1e (default).
# delimiter=\t\t -- Two tabs
```

Chapter 3
Multimedia Playlist API

The Multimedia Playlist API exposes the constants, data types (including enumerations), and functions that client applications can use to open playlist sessions, seek to specific track positions, and retrieve playlist metadata.

Before it can access any playlists, the client must first initialize the **libmmplaylist** library by calling *mmplaylist_init()* (p. 30), which loads the playlist plugins (PLPs) listed in the configuration file.

The client can then open sessions on playlists by calling *mmplaylist_open()* (p. 33). In each *mmplaylist_open()* call, the client can specify the character encoding and validation to perform on each entry that's retrieved by an *mmplaylist_entry_next_get()* (p. 21) call.

Media applications can read playlist information for many reasons. For example, an application could extract the track URLs of a playlist's entries and pass these URLs to **mm-renderer** to play the tracks one by one. Or, it could retrieve the number of entries in all the playlists on a mediastore and display this information to the user as a type of playlist inventory.

The *mmplaylist_props_get()* (p. 36) function allows the client to learn which operations (e.g., seeking to a new position, getting the number of playlist entries) have fast implementations in the current session and to then optimize the user experience by avoiding slow operations.

When it's finished using a playlist, the client can close the corresponding playlist session by calling *mmplaylist_close()* (p. 18). When it's finished using **libmmplaylist** altogether (e.g., during shutdown), the client must call *mmplaylist_terminate()* (p. 42) to unload the PLPs used by the library.

mmplaylist_close()

Close a playlist session

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_close( mmplaylist_t *pl )
```

Arguments:

pl

A valid session handle.

Library:

libmmplaylist

Description:

This function closes an active playlist session. If an error occurs, the function returns an error code but the session handle still becomes invalid. The application should not, under any circumstances, use the handle after calling this function.

Returns:

0

Success. This value is equivalent to the mmplaylist_ok error code.

>0

An **mmplaylist_error_type_t** (p. 27) constant indicating the error that occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_current_pos_get()

Get the position of the current playlist entry

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_current_pos_get( mmplaylist_t *pl )
```

Arguments:

pl

A valid session handle.

Library:

libmmplaylist

Description:

This function gets the position of the current playlist entry. When the current position is at the end of the playlist, this function returns -1 and sets an error code of mmplaylist_end_of_playlist.

Returns:

>=0

The position of the current entry.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_current_pos_set()

Jump to a position in the playlist

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_current_pos_set( mmplaylist_t *pl,
                                int new_pos,
                                mmplaylist_seek_offset_t whence )
```

Arguments:

pl

A valid session handle.

new_pos

The new position in the playlist to jump to. Negative numbers are allowed for moving backwards.

whence

The reference point for the new position (see **mmplaylist_seek_offset_t** (p. 38) for the list of acceptable values).

Library:

libmmplaylist

Description:

This function jumps to a position in the playlist. The new position is specified as the index of the entry that you're seeking to. This operation can be very expensive, so the application should first check the value of the *seek_fast* field in the **mmplaylist_props_t** (p. 37) structure to determine whether this operation can be completed quickly.

Returns:

0

Success.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_entry_next_get()

Get the next entry from a playlist

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_entry_next_get( mmplaylist_t *pl,
                               mmplaylist_entry_t **entry )
```

Arguments:

pl

A valid session handle.

entry

A pointer for storing a reference to the entry pointer defined by this function.

Library:

libmmplaylist

Description:

This function gets the next entry from a playlist. The data structure filled in by this function contains the next entry from the playlist file and a flag field that describes the properties of this entry.

When an application requests the next playlist entry, the selected plugin:

1. Retrieves the next entry from the playlist file.
2. Identifies and possibly modifies the character encoding of the entry, based on the preferences given when the playlist session was created. The actual encoding of the playlist entry returned to the caller might differ from what's specified in the preferences (see "*Playlist sessions* (p. 13)" for an explanation).
3. If necessary, attempts to identify the entry's track file by validating the entry with either the specified callback or the *stat()* function (see **mmplaylist_entry_validated_t** (p. 25) for the list of properties that the validation function can set).

This last step repeats until either the callback determines that the entry is valid or the playlist manager has iterated through the character conversion and/or validation routines. At this point, the function returns the raw entry to the caller.

Callers must use *mmplaylist_entry_release()* (p. 23) to free the memory for a playlist entry when they're done with it. They must not free the memory themselves; doing so will result in unpredictable behavior.

Returns:

1

The entry was successfully retrieved.

0

The end of the playlist was reached.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_entry_release()

Release memory for a playlist entry

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_entry_release( mmplaylist_t *pl,
                              mmplaylist_entry_t *entry )
```

Arguments:

pl

A valid session handle.

entry

A pointer to the entry whose memory is being released (freed).

Library:

libmmplaylist

Description:

This function releases the memory for a playlist entry. This memory was allocated in an earlier *mmplaylist_entry_next_get()* (p. 21) call.

Returns:

0

Success.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_entry_t

Playlist entry information

Synopsis:

```
#include <mm/mmplaylist.h>

typedef struct mmplaylist_entry {
    uint32_t props;
    char name[];
} mmplaylist_entry_t;
```

Data:

uint32_t props

The properties of the entry.

char name

The extracted entry. This string is null-terminated.

Library:

libmmplaylist

Description:

The **mmplaylist_entry_t** data structure stores a playlist entry. This structure is filled in and returned to the client by *mmplaylist_entry_next_get()* (p. 21). When finished using this information, the client must free the structure by calling *mmplaylist_entry_release()* (p. 23).

mmplaylist_entry_validated_t

Properties of retrieved playlist entries

Synopsis:

```
#include <mm/mmplaylist.h>

typedef enum {
    MMPLAYLIST_ENTRY_RESOLVED = 0,
    MMPLAYLIST_ENTRY_NOT_LOCATED = 0x01,
    MMPLAYLIST_ENTRY_NOT_CONVERTED = 0x02,
} mmplaylist_entry_validated_t;
```

Data:

MMPLAYLIST_ENTRY_RESOLVED

The resolved entry is valid (no further processing is required).

MMPLAYLIST_ENTRY_NOT_LOCATED

The entry's track file wasn't found.

MMPLAYLIST_ENTRY_NOT_CONVERTED

The entry couldn't be converted to the desired encoding.

Library:

libmmplaylist

mmplaylist_error_info_t

Error information

Synopsis:

```
#include <mm/mmplaylist.h>

typedef struct {
    int error_code;
} mmplaylist_error_t;
```

Data:

int error_code

An **mmplaylist_error_type_t** (p. 27) value identifying the error that occurred.

Library:

libmmplaylist

Description:

The **mmplaylist_error_t** structure stores information on the last error that occurred in a session. This structure is filled in by *mmplaylist_last_error_get()* (p. 31).

mmplaylist_error_type_t

Playlist session errors

Synopsis:

```
#include <mm/mmplaylist.h>

typedef enum {
    mmplaylist_ok = 0,
    mmplaylist_err,
    mmplaylist_no_memory,
    mmplaylist_char_conv_failure,
    mmplaylist_entry_conf_failure,
    mmplaylist_entry_validation_failure,
    mmplaylist_config_error,
    mmplaylist_entry_too_large,
    mmplaylist_read_error,
    mmplaylist_end_of_playlist,
    mmplaylist_position_too_short,
    mmplaylist_position_too_long,
    mmplaylist_file_system_error,
    mmplaylist_invalid_input,
    mmplaylist_session_invalid
} mmplaylist_error_type_t;
```

Data:

mmplaylist_ok

The operation was successful.

mmplaylist_err

An unexpected error occurred (e.g., initialization failed or a plugin couldn't complete the requested operation).

mmplaylist_no_memory

The library couldn't allocate enough memory to store the requested information.

mmplaylist_char_conv_failure

The character conversion failed.

mmplaylist_entry_conf_failure

An error occurred while converting a playlist entry to the requested encoding.

mmplaylist_entry_validation_failure

An error occurred while validating a playlist entry.

mmplaylist_config_error

The library couldn't read the configuration file or a plugin failed to load.

mmplaylist_entry_too_large

The entry was too large for the library to process, so the caller should skip this entry and get the next one.

mmplaylist_read_error

A plugin encountered an error while parsing the next entry.

mmplaylist_end_of_playlist

The end of the playlist was reached.

mmplaylist_position_too_short

The requested new position is before the beginning of the playlist.

mmplaylist_position_too_long

The requested new position is after the end of the playlist.

mmplaylist_file_system_error

An error occurred while parsing the playlist file.

mmplaylist_invalid_input

An invalid argument was given.

mmplaylist_session_invalid

The playlist session is invalid.

Library:

libmmplaylist

Description:

The **mmplaylist_error_type_t** enumerated type defines codes for playlist session errors. These values are stored in the **mmplaylist_error_info_t** (p. 26) structure.

mmplaylist_fmt_list_t

List of supported playlist formats

Synopsis:

```
#include <mm/mmplaylist.h>

typedef struct mmplaylist_fmt_list {
    int num_entries;
    char **playlist_fmtstr;
} mmplaylist_fmt_list_t;
```

Data:

int num_entries

The number of extensions/formats that are supported.

char **playlist_fmtstr

An array of strings containing the supported extensions/formats.

Library:

libmmplaylist

Description:

The **mmplaylist_fmt_list_t** structure stores the list of supported playlist formats. This structure is filled in and returned to the client by *mmplaylist_supported_fmts_get()* (p. 39). When finished using this information, the client must free the structure by calling *mmplaylist_supported_fmts_release()* (p. 40).

mmplaylist_init()

Initialize the library

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_init( const char *config )
```

Arguments:

config

The path of the configuration file to use. Set this parameter to NULL to make the library use the file at the path in the MM_PLAYLIST_CONFIG environment variable or at the default path if that variable isn't defined.

Library:

libmmplaylist

Description:

This function initializes the playlist library by loading the character converter library, parsing the configuration file, and then trying to load each PLP named in the configuration file.

You must call this function once and it must be the first function you call.

Returns:

0

Success. This value is equivalent to the mmplaylist_ok error code.

>0

An **mmplaylist_error_type_t** (p. 27) constant indicating the error that occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_last_error_get()

Get information about the last error

Synopsis:

```
#include <mm/mmplaylist.h>

void mmplaylist_last_error_get( mmplaylist_t *pl,
                                mmplaylist_error_info_t *error_info )
```

Arguments:

pl

A valid session handle.

error_info

A pointer to a structure for storing the error information.

Library:

libmmplaylist

Description:

This function gets information on the last error that occurred for the specified session. The library fills in the **mmplaylist_error_info_t** (p. 26) structure referred to by *error_info*.

Note that reading the last error value clears it as well.

mmplaylist_num_entries_get()

Get the number of entries in a playlist

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_num_entries_get( mmplaylist_t *pl )
```

Arguments:

pl

A valid session handle.

Library:

libmmplaylist

Description:

This function gets the number of entries in a playlist. This operation can be very expensive, so the application should first check the *num_entries_get_fast* field in the **mmplaylist_props_t** (p. 37) structure to determine whether this operation can be completed quickly.

Note that calling this function on an empty playlist returns a value of 0.

Returns:

>=0

The number of playlist entries.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_open()

Open a session on a playlist

Synopsis:

```
#include <mm/mmplaylist.h>

mmplaylist_t* mmplaylist_open(
                    const char *base_name,
                    const char *playlist_name,
                    mmplaylist_validation_mode_t validation_mode,
                    mmplaylist_validation_fn_t *validate_fn,
                    void *cbk_data )
```

Arguments:

base_name

A string containing the base directory of the mediastore on which the playlist is contained.

playlist_name

A string containing the full path to the playlist. This path can be an absolute filepath or a URL.

validation_mode

The method that **libmmplaylist** must use to resolve playlist entries (see *mmplaylist_validation_mode_t* (p. 44) for the list of acceptable values).

validate_fn

The callback function to use for validating playlist entries. This optional parameter can be set to NULL, in which case the library uses *stat()* to validate entries. Also, entry validation is done only when *validation_mode* is set to MMPLAYLIST_ENTRY_VALIDATE.

cbk_data

Data for the callback function. This data gets passed unmodified to the callback function. The data can be NULL.

Library:

libmmplaylist

Description:

This function opens a session on a playlist. The library creates and returns a handle to represent the new playlist session. Internally, the library queries the available plugins to identify which ones support this playlist format. If multiple plugins support the format, the highest-ranked one is selected for use with this session. The plugin selection is stored in the session handle.

Returns:

A pointer to the new playlist handle

Success (i.e., the playlist is supported).

NULL

Failure.

mmplaylist_plps_list()

List available playlist plugins

Synopsis:

```
#include <mm/mmplaylist.h>

ssize_t mmplaylist_plps_list( char *buffer, size_t buf_len )
```

Arguments:

buffer

A pointer to memory for storing a comma-separated list of playlist plugin (PLP) names. When NULL, the function writes no data but returns the buffer size required for holding the PLP names.

buf_len

Length of buffer (can be 0).

Library:

libmmplaylist

Description:

This diagnostic function returns a list of all PLPs that were successfully loaded and initialized.

You must allocate the memory for the buffer that will hold the plugin names. If you need to know how much memory to allocate, call this function with *buffer* set to NULL. The function then returns the number of bytes needed to store the names. You can then allocate this much memory in a buffer and call *mmplaylist_plps_list()* a second time, passing in a pointer to the new buffer to make the library populate it with the list of PLP names.

Returns:

>=0

The buffer length needed to list all available plugins.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_props_get()

Get properties of a playlist session

Synopsis:

```
#include <mm/mmplaylist.h>

mmplaylist_props_t* mmplaylist_props_get( mmplaylist_t *pl )
```

Arguments:

pl

A valid session handle.

Library:

libmmplaylist

Description:

This function gets the properties of a playlist session. The properties are written in a data structure returned to the caller. Applications can then read the properties to learn which operations are fast in the current session and then provide a better user experience by avoiding slow operations.

The *mmplaylist_props_get()* function allocates memory for the structure that it returns. The caller should free this memory using *free()*.

Returns:

A pointer to an mmplaylist_props_t structure

Success.

NULL

Failure (call *mmplaylist_last_error_get()* for details).

mmplaylist_props_t

Playlist session properties

Synopsis:

```
#include <mm/mmplaylist.h>

typedef struct mmplaylist_props {
    int num_entries_get_fast;
    int seek_fast;
    int rewind_fast;
} mmplaylist_props_t;
```

Data:

int num_entries_get_fast

A value of 1 if the number of playlist entries can be retrieved quickly; 0 if it can't be.

int seek_fast

A value of 1 if the playlist allows the client to quickly jump to an arbitrary position; 0 if it doesn't.

int rewind_fast

A value of 1 if the playlist allows the client to quickly rewind to the beginning of the playlist; 0 if it doesn't.

Library:

libmmplaylist

Description:

The **mmplaylist_props_t** structure is used to return the properties of a playlist session to the client application (through *mmplaylist_props_get()* (p. 36)). The operation to retrieve this information is always fast. By reading the properties returned, the client can choose to call certain library functions based on whether the corresponding operation is fast.

mmplaylist_seek_offset_t

Directives for seeking to playlist positions

Synopsis:

```
#include <mm/mmplaylist.h>

typedef enum {
    MMPLAYLIST_SEEK_CUR = 0,
    MMPLAYLIST_SEEK_ABS
} mmplaylist_seek_offset_t;
```

Data:

MMPLAYLIST_SEEK_CUR

A position relative to the current position.

MMPLAYLIST_SEEK_ABS

An absolute position in the playlist.

Library:

libmmplaylist

mmplaylist_supported_fmts_get()

Get the list of supported playlist formats

Synopsis:

```
#include <mm/mmplaylist.h>

mmplaylist_fmt_list_t* mmplaylist_supported_fmts_get( void )
```

Library:

libmmplaylist

Description:

This function gets the list of playlist formats supported by the library. The data structure filled in by this function contains the number of supported playlist types and a pointer to an array of strings containing the file extensions of the supported playlist types.

The library allocates memory for the data structure and returns a pointer to it. Callers must use *mmplaylist_supported_fmts_release()* (p. 40) to release the memory when they're done with it. They must not modify or free the memory themselves; doing so will result in unpredictable behavior.

Returns:

A pointer to an mmplaylist_fmt_list_t structure

Success.

NULL

Failure (call *mmplaylist_last_error_get()* for details).

mmplaylist_supported_fmts_release()

Release memory for the list of supported playlist formats

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_supported_fmts_release(
                         mmplaylist_fmt_list_t *extn_list )
```

Arguments:

extn_list

The list of supported playlist formats.

Library:

libmmplaylist

Description:

This function releases the memory for the list of playlist formats supported by the library. This memory was allocated in an earlier *mmplaylist_supported_fmts_get()* (p. 39) call.

Returns:

0

Success.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_t

Data type for playlist session handle

Synopsis:

```
#include <mm/mmplaylist.h>

typedef struct mmplaylist mmplaylist_t;
```

Library:

libmmplaylist

Description:

The **mmplaylist_t** structure is a private data type representing a playlist session handle.

mmplaylist_terminate()

Terminate the library

Synopsis:

```
#include <mm/mmplaylist.h>

int mmplaylist_terminate( void )
```

Library:

libmmplaylist

Description:

This function terminates the playlist library from use by unloading all the PLPs. You must call this function once and it must be the last function you call.

If an error occurs, the function returns an error code but the library still becomes unusable. The application should not, under any circumstances, call any library function (not even *mmplaylist_last_error_get()* (p. 31)) after calling this function.

Returns:

0

Success.

-1

An error occurred (call *mmplaylist_last_error_get()* for details).

mmplaylist_validation_fn_t

Prototype for the validation function

Synopsis:

```
#include <mm/mmplaylist.h>

typedef mmplaylist_entry_validated_t
            mmplaylist_validation_fn_t( char *filename, void *cbk_data );
```

Library:

libmmplaylist

Description:

The **mmplaylist_validation_fn_t** data type specifies the prototype for the validation function. See the *mmplaylist_entry_next_get()* (p. 21) description for an explanation of playlist entry validation.

The function takes two parameters:

- *filename*, which contains the name being proposed as resolved
- *cbk_data*, which stores a pointer to data for the callback (validation) function

The function returns an **mmplaylist_entry_validated_t** (p. 25) constant indicating the properties of the retrieved entry.

mmplaylist_validation_mode_t

Directives for validating playlist entries

Synopsis:

```
#include <mm/mmplaylist.h>

typedef enum {
    MMPLAYLIST_ENTRY_RAW_ENC = 0,
    MMPLAYLIST_ENTRY_UTF8_ENC,
    MMPLAYLIST_ENTRY_VALIDATE,
    MMPLAYLIST_ENTRY_UTF8_VALIDATE
} mmplaylist_validation_mode_t;
```

Data:

MMPLAYLIST_ENTRY_RAW_ENC

Return raw playlist entries to the caller.

MMPLAYLIST_ENTRY_UTF8_ENC

Return playlist entries converted to UTF-8 to the caller.

MMPLAYLIST_ENTRY_VALIDATE

Use a helper function to validate the raw entry before returning it to the caller.

The library will validate the entry as an absolute path, not just the entry read from the playlist.

MMPLAYLIST_ENTRY_UTF8_VALIDATE

Use a helper function to validate a UTF-8 encoded entry before returning it to the caller.

The library will pass only the encoded entry to the validation callback; it won't try to make the path absolute.

Library:

libmmplaylist

Description:

The **mmplaylist_validation_mode_t** enumerated type defines the types of validation that can be performed on playlist entries. This list might be extended at a future date.

Index