Metadata Provider 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: Monday, February 24, 2014

Table of Contents

About This Reference

The *Metadata Provider Library Reference* is aimed at developers who want to write applications that use the `libmd` library to extract metadata from media files on attached devices. This metadata lets applications display track information and artwork so users can quickly browse device filesystems and search media libraries.

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

To find out about:	Go to:
The purpose and capabilities of `libmd`	*Metadata Provider Overview* (p. 9)
The list of included Metadata Providers (MDPs)	*Included MDPs* (p. 13)
The `libmd` configuration file, which lists the plugins and their preferential order	*Configuration file* (p. 18)
Using the Metadata Provider Library API to manage metadata-extraction sessions and retrieve metadata from media files	*Metadata Provider API* (p. 21)

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`
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.

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

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 use a forward slash (/) as a delimiter in all 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
Metadata Provider Overview

The metadata provider library, libmd, extracts metadata from media files on attached devices to provide client applications with up-to-date information on the media content available for browsing or playback.

Metadata is information that describes media files. This information can include details such as the artist name, album title, or year of creation (for a track), as well as playback details such as track runtime or picture dimensions. Clients use metadata to:

- display details of the currently playing track to users
- provide the names and other file information of selected tracks to media browsers
- display album artwork to users to enhance their media experience
- generate cover flows so users can visually browse albums and tracks

Clients make requests of `libmd` to extract specific sets of metadata fields from individual media files. With this design, clients can retrieve the exact metadata they need at precise times so they can optimize performance and the user experience. For example, suppose the user selects a track in their media browser. The application that provides media information to the browser can extract the track's creation information fields (which are small and fast to retrieve) but not its embedded artwork images (which can be large and slow to process). This strategy increases the browser's responsiveness.

Consider a cover flow application that allows users to visually browse their song collection and begin playback by selecting an album image. The application can extract the cover art when generating the flow of albums and then extract the artist name, year of release, and other album information when the user selects an album image. This “load-on-demand” philosophy supports a good user experience by ensuring the exact information—whether images or text—becomes available as soon as the user needs it.

The multimedia synchronizer service, `mm-sync`, uses `libmd` to extract media metadata so it can upload that metadata to QDB databases. But updating databases requires indexing most or all files on a mediastore, so you might not want to rely on `mm-sync` for obtaining metadata and instead use `libmd`. Retrieving metadata through `libmd` lets you prioritize metadata extraction by reading metadata from one file or a select group of files.

Architecture of libmd

The `libmd` library uses a plugin architecture in which independent plugins support different sets of metadata fields. When a client requests metadata, the library extracts it using one or more plugins and then returns the set of filled-in metadata fields to the client.

The library is implemented in three layers:

Data processing

This layer:

- stores and updates the plugin ratings for metadata fields from specific media file types
- collates (i.e., combines and orders) the metadata field values returned from plugins

Plugin management

This layer:

- parses the configuration file to learn which library files implement the plugins and to read the preference order for various file types
- loads, initializes, and unloads plugins

Plugins

This layer consists of many Metadata Provider (MDP) plugins, each of which:

- manages communication sessions for responding to metadata requests and for reporting errors
- rates itself on its ability to retrieve the requested metadata fields
- retrieves metadata by extracting media information from the named item (media file)

This design lets `libmd` offer a common, high-level interface for extracting metadata from many file types on many device types. Clients need to name only a media file and the metadata fields they want and `libmd` then invokes the necessary MDP plugins to read the metadata and returns the extracted metadata fields to the client.

Each MDP fills in as many fields as it can. The order that `libmd` uses to invoke the MDPs depends on the plugin preferences stated in the *configuration file* (p. 18). The preferential order for plugins can vary from one file type to another.

The file types and their associated URL prefixes supported by `libmd` are:

File type	URL prefix
CDDA track	`cdda:`
POSIX file	`file:`
iPod media file	`ipod:`
HTTP stream	`http:`
RTSP stream	`rtsp:`
File on an MTP device	`mtp:`

If no URL prefix is given, the POSIX file type is assumed (e.g., a URL of `/fs/usb0/one.mp3` is equivalent to `file:/fs/usb0/one.mp3`).

MDPs hide the details of the media interface used for reading metadata so clients can extract it through different network protocols from a variety of hardware. Clients can read metadata from the following device types:

- USB sticks, SD cards, or any storage devices with block filesystems
- iPods
- audio CDs
- MTP devices
- media streams from external sources (e.g., HTTP servers)

The plugin-based architecture makes it possible for future releases of `libmd` to support additional file and device types. The `libmd` library could add new MDPs to provide more sources of metadata while clients continue to use the same commands to extract it.

Metadata providers

Metdata providers (MDPs) are `libmd` plugins that do the actual metadata extraction from media files. MDPs tell the data-processing layer of `libmd` which metadata fields (types) they can extract. When requested, MDPs read as many of the metadata fields listed by the client as possible from the specified media item.

MDP ratings

To handle a client request for metadata, `libmd` queries all loaded MDPs for their ratings on the metadata fields listed in the request. Each MDP keeps an internal map of the fields it can extract from media files. This map contains the field names (i.e., metadata types) and other information such as which collation method to use for handling multiple values for a given field. MDPs consult this map to generate lists of field-specific Boolean ratings (`1` means the plugin can extract the field, `0` means it can't) and then return these ratings to the data-processing layer.

When selecting an MDP plugin as the metadata source, `libmd` considers only MDPs that gave a rating of `1` for at least one metadata field, which means they can extract some or all of the requested information. To pick an MDP within the set of MDPs rated `1` for some field, `libmd` examines the plugin order for the file type of the media item named in the request. This plugin order is read from the configuration file during initialization (see *Configuration file* (p. 18)).

Metadata extraction

When `libmd` asks an MDP to retrieve metadata, the selected MDP parses the request data to obtain either a fully qualified path to the item (media file) or some other information referencing the item (e.g., a track's unique ID (UID) on iPods). Next, the MDP uses POSIX system calls or system libraries to browse the device's directories and files and to read its file information to generate metadata. For instance, the CDDA plugin calls *devctl()* to issue device commands to CDs mounted in the local filesystem. These commands include reading the CD-Text data, which contains album metadata.

The MDP stores the information read from the device in metadata strings and returns these strings along with the number of metadata types (fields) for which metadata was found to the data-processing layer of `libmd`. If the number of types found is less than the number requested by the client, `libmd` picks another MDP to get metadata for the remaining fields. The `libmd` library continues invoking MDPs until all requested metadata fields have been filled in or until it exhausts all MDPs.

Included MDPs

Different MDPs support different file types and metadata fields. When requesting metadata fields, you must state the metadata categories and the attributes (which map to individual fields); see *mmmd_get()* (p. 25) for more details. The `libmd` library combines a category (the prefix) with each of its listed attributes (the suffixes) to form the full names of the metadata fields.

The MDPs shipped with `libmd` and the file types and metadata fields (expressed as category and attribute combinations) that each MDP supports is as follows:

MDP	Files	Metadata categories	Attributes
CDDA	CD audio tracks	`md_title`	`album`, `artist`, `genre`, `name`, `composer`, `track`, `bitrate`, `samplerate`, `duration`, `format`
MMF	MMF files accessible from either a network source (e.g., an HTTP server) or a POSIX device	`md_title`	`name`, `artist`, `album`, `albumartist`, `composer`, `genre`, `comment`, `duration`, `track`, `disc`, `year`, `seekable`, `pausable`, `samplerate`, `bitrate`, `protected`, `mediatype` (see *Footnote.*), `width`, `height`, `art`, `compilation`, `rating`
		`md_video`	`width`, `height`, `pixel_width`, `pixel_height`, `frame`, `fourcc`
		`md_audio`	`fourcc`
		`md_artwork`	`image`, `description`, `type`, `mimetype`, `count`, `size`
Exif	POSIX files on mass storage devices (e.g., USB sticks)	`md_title`	`width`, `height`, `date_time_original`, `shutter_speed`, `fnumber`, `iso_speed_ratings`, `focal_length`, `orientation`, `description`, `latitude`, `longitude`, `keywords`
Extart	External artwork such as cover images for albums and thumbnail graphics for tracks	`md_title`	`art`
		`md_artwork`	`image`, `count`, `size`, `urls`
MediaFS	Files on MTP devices	`md_title`	`name`, `artist`, `album`, `composer`, `genre`, `year`, `duration`, `comment`, `protected`, `track`, `art`
		`md_artwork`	`image`, `width`, `height`, `size`, `mimetype`, `count`
iPod	iPod media files	`md_title`	`art`

[1] The value that `libmd` returns for the `md_title_mediatype` field is in decimal but should be converted to hexadecimal for readability. For the mapping of hexadecimal values to media types, see the *MediaFormat_t* data structure description in the *Addon Interface Library Reference*.

MDP	Files	Metadata categories	Attributes
		`md_artwork`	`image`, `mimetype`, `width`, `height`, `size`, `count`

Metadata-extraction sessions

To extract metadata with `libmd`, a client must establish a communication session with the library before it can issue commands to read metadata from media files stored on an attached device.

To establish a communication session (or *metadata-extraction session*) with `libmd`, the client must name a mediastore (device) to extract the metadata from. If desired, the client can then set session parameters to influence the behavior of MDPs. These parameters can be set only once, so the client should set them just after opening a session but before extracting any metadata.

The client can use any open session to send requests to `libmd` to extract metadata from individual items (media files). In each metadata request, the client must supply the item's path or some device-specific information identifying the item (e.g., a UID on iPods) and must list the desired metadata fields. The client can also request a maximum number of “matches” (i.e., responses returned by different plugins) for metadata fields. Retrieving multiple values for metadata fields lets a client pick the set of values that provide the user with the most complete and accurate media information possible.

Concurrent sessions

Clients can open and extract metadata from as many concurrent `libmd` sessions as they like. This design lets applications display media information for multiple devices to users. We recommend a limit of one session per mediastore to avoid redundant reads of metadata from the same files.

Obtaining error information

While a session is active, the client can obtain information about the last error that occurred for that session by calling *mmmd_error_info()* (p. 24). This function returns error information, including the numeric error code, a string summarizing the error, and an error message. We recommend that your client code check the return values of all API calls. If any value indicates an error, the client can retrieve the error information and use it to help recover.

Chapter 2
Configuring Metadata Providers

You can configure metadata providers (MDPs) in two ways: in the configuration file to define initial settings and through the libmd API to define settings for individual metadata-extraction sessions.

During startup, `libmd` reads its configuration file and loads each listed MDP. After an MDP loads successfully, `libmd` initializes it with any settings listed in the configuration file. These settings apply to the MDP throughout the client application's lifetime.

When `libmd` has finished its setup, your client can establish metadata-extraction sessions and assign parameters to those sessions to influence how MDPs retrieve metadata.

To assign parameters to active sessions (*dynamic parameters*), the client must call *mmmd_session_params_set()* (p. 31) while providing the session handle and the list of parameters. Parameters defined in this manner apply only to the session referred to in the API call. Once set, they can't be changed or unset.

Currently, only the MMF MDP examines dynamic parameters, which it uses to configure the streamers for reading files from HTTP servers. Whether this plugin is used in metadata extraction depends on the type of the media item being read and the plugin preferences stated in the configuration file. When `libmd` uses other MDPs to read metadata, dynamic parameters have no effect. You should therefore set these parameters only when you plan to extract metadata from HTTP servers.

The MDP settings recognized by `libmd` when parsing the configuration file (*static parameters*) differ from those you can assign to an active metadata-extraction session. See the default configuration file for the supported static parameters. For information on the dynamic parameters recognized by MMF, see *mmmd_session_params_set()* (p. 31).

Configuration file

The `libmd` configuration file lists the preferential plugin order, the library files implementing the plugins, and other configuration settings.

The `libmd` library is shipped with a default configuration file (`/etc/mm/mm-md.conf`). You can modify this included file or create your own. Before doing anything else with `libmd`, your client must call *mmmd_init()* (p. 28) and supply either the full path to your own configuration file or a path of NULL to use the default configuration file. Your client must do this exactly once to initialize the library.

Any section that defines settings for an individual plugin (or MDP) must begin with a line of the form:

```
[plugin]
```

The settings are listed on the lines that follow, one per line. The syntax for an MDP setting consists of a field name, followed by the equal sign (=), followed by the field value. For example, the following line enables lazy load filters for MMF:

```
lazyloadfilters=1
```

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

A `dll` setting is required in every plugin section. This setting names the library file implementing the MDP plugin. To support a good user experience, your configuration file should define at least the minimum set of MDPs capable of extracting all the metadata fields needed by your client applications. Most likely, you'll have to define more than one plugin section because most MDPs don't support every metadata field.

The section defining the preferential plugin order begins with a line of the form:

```
[typeratings]
```

The lines that follow list the MDP preferences for specific file types. Each line contains a URL prefix that represents a file type, followed by the MDPs to use for metadata extraction, from most to least preferred. Suppose you want to inform `libmd` of your plugin preferences for POSIX files, whose URLs have either a `file` prefix or no prefix at all. If you want to use the `MMF` MDP first and then the `Exif` MDP if some metadata fields can't be retrieved by the `MMF` MDP, enter the following line:

```
file=mmf,exif
```

Default configuration file

The contents of the default configuration file look like this:

```
# libmd config file
```

```
[plugin]
dll=mm-mdp-mmf.so
lazyloadfilters=1

[plugin]
dll=mm-mdp-exif.so

[plugin]
dll=mm-mdp-cdda.so

#[plugin]
#dll=mm-mdp-ipod.so

#[plugin]
#dll=mm-mdp-extart.so
#ignore_case=true
#max_search=100
#max_cache_entries=0

# All regular expressions following the first
# instance must have a unique suffix appended
# to them (e.g., regex, regex1, regex2).
#regex=album\.jp[e]?g
#regex1=folder\.jp[e]?g

#[plugin]
#dll=mm-mdp-mediafs.so

[typeratings]
file=mmf,exif
http=mmf
cdda=cdda
rtsp=mmf
#ipod=ipod
#mtp=mediafs
```

Chapter 3
Metadata Provider API

The Metadata Provider API exposes the constants, data types (including enumerations), and functions that your client applications can use to initialize the libmd library, create metadata-extraction sessions, and submit metadata retrieval requests.

The first action any client must perform with `libmd` is to initialize the library by calling *mmmd_init()* (p. 28) while supplying the path of the configuration file, which lists the metadata providers (MDPs) to load.

Before it can extract any metadata, your client must establish a *metadata-extraction session* with `libmd` by calling *mmmd_session_open()* (p. 30) while providing the name of the mediastore (device) to read metadata from.

The client can then request specific metadata fields from specific items (media files) by calling *mmmd_get()* (p. 25). The client can ask for a maximum number of *matches* (i.e., responses from different MDPs). Retrieving multiple matches lets the client pick the set of metadata values that provide the most complete and accurate media information possible.

When it's finished retrieving metadata, the client can close the corresponding session by calling *mmmd_session_close()* (p. 30). When it's finished using `libmd` altogether (e.g., during shutdown), the client must call *mmmd_terminate()* (p. 32) to clean up the resources used by the library.

md.h

Defines data types for session handles and error information as well as functions for managing metadata-extraction sessions, getting metadata from media items, and retrieving diagnostic and error information.

Constants in md.h

Preprocessor macro definitions in `md.h`.

Defines:

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

```
#define MD_COVERART_BYREF "BYREF"
```

Keyword for returning cover art by reference (MDPs won't write cover art to a file if requested). Example: `md_artwork::image?file=BYREF`.

Library:

`libmd`

Data types in `md.h`

Data types defined in `md.h`.

mmmd_error_info_t

Information on the last session error.

Synopsis:

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

typedef struct mmmd_error_info {
    mmmd_errcode_t code;
    int64_t extended_code;
    char extended_type[16];
    char extended_msg[256];
} mmmd_error_info_t;
```

Data:

mmmd_errcode_t code

The numeric error code.

int64_t extended_code

The numeric extended error code.

char extended_type

A string containing the extended error type.

char extended_msg

An extended error message.

Library:

`libmd`

Description:

The `mmmd_error_info_t` structure describes errors that occurred during a metadata-extraction session.

This structure may change between `libmd` releases.

mmmd_flags_t

Flags for controlling library logs.

Synopsis:

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

typedef enum {
    MMMD_FLAG_EMIT_TIMING_LOGS = 0x01
} mmmd_flags_t;
```

Data:

`MMMD_FLAG_EMIT_TIMING_LOGS`

Tells the library to emit timing logs.

Library:

`libmd`

Description:

The `mmmd_flags_t` enumeration defines constants for controlling logs for the library.

mmmd_hdl_t

Session handle type.

Synopsis:

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

typedef struct mmmd_hdl mmmd_hdl_t;
```

Library:

`libmd`

Description:

The `mmmd_hdl_t` structure is a private data type representing a session handle.

Functions in `md.h`

Functions defined in `md.h`.

mmmd_error_info()

Get information on the last error in a session.

Synopsis:

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

const mmmd_error_info_t* mmmd_error_info( mmmd_hdl_t *hdl )
```

Arguments:

hdl

The handle of the session whose error information is being retrieved

Library:

`libmd`

Description:

Get information that describes the last error that occurred in a metadata-extraction session.

Returns:

A pointer to the error information structure

mmmd_flags_set()

Set control logs for the library.

Synopsis:

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

mmmd_flags_t mmmd_flags_set( mmmd_flags_t new_flags )
```

Arguments:

new_flags

The new flags to set

Library:

`libmd`

Description:

Set control logs for the library, based on the new flags setting.

Returns:

The old flags setting

mmmd_get()

Get the specified metadata fields from the specified item.

Synopsis:

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

int mmmd_get( mmmd_hdl_t *hdl,
              const char *item,
              const char *types,
              const char *source,
              uint32_t count,
              char **md )
```

Arguments:

hdl

The handle of the session associated with the mediastore where metadata is being read

item

A URL or an absolute path to the item containing the metadata

types

A string storing the requested metadata types (fields) as a series of *group-attributes* listings. Here, *group* refers to the metadata category (e.g., `title`) while *attributes* refers to the list of requested attributes (e.g., `artist`, `album`).

Each group and its list of attributes must be separated by the `::` delimiter and the individual attributes must be separated by commas. Also, each group-attributes listing must be followed by a line-break character, as shown in the following example:

```
md_title::name,artist,album\nmd_video::width,height
```

This syntactic grouping of metadata types makes it easy to request multiple related fields.

source

A string specifying the metadata source (i.e., the MDP to use). Currently, this feature isn't supported so this argument must be NULL to indicate that all sources can be used.

count

The number of desired matches (i.e., responses from MDPs).

If *source* is NULL and *count* is `0`, all responses are collated to return the highest-rated response (see the *Description* (p. 27) subsection for an explanation).

If *source* is NULL and *count* is nonzero, the number of responses returned is less than or equal to *count*, starting with the highest-rated response. No collation is performed.

md

A pointer to a string pointer that references the buffer storing the response. The string pointer is set by the function. Examples of the formatting and typical contents of the response buffer are given in the *Description* (p. 27) subsection.

Library:

`libmd`

Description:

Get the specified metadata fields from the specified item. The *types* string must state the requested fields as group-attributes listings, as explained in the *Arguments* (p. 25) subsection.

Because different MDPs support different fields, `libmd` uses as many MDPs as necessary to extract metadata for all the fields listed in *types*. The order that `libmd` uses to invoke the MDPs is the plugin preference order for the file type indicated by the URL or path in *item*. The preference order is stated in the configuration file.

For the lists of fields supported by different MDPs, see *Included MDPs* (p. 13).

Retrieving multiple responses

Setting *count* to a value greater than `0` allows you to retrieve multiple matches (responses) for metadata fields. Your client code can then choose the set of responses that provides the user with the most accurate and complete metadata possible. The number of responses returned is less than *count* if the number of MDPs supporting any of the requested fields is also less than *count*. A nonzero value for this argument simply limits the number of responses that can be returned.

Suppose a client sets *count* to `3` and requests the `md_title_artist` and `md_title_orientation` fields from a POSIX file while the MDP preference order for POSIX files is `mmf, mediafs, exif`. The `MMF` and `MediaFS` MDPs support the first field but not the second; the `Exif` MDP supports the second field but not the first. The `libmd` library stores a pointer in *md* that references the following string:

```
md_src_name::mmf\nmd_src_rating::0\nmd_title_artist::some_artist\n
md_src_name::mediafs\nmd_src_rating::1\nmd_title_artist::some_artist
\nmd_src_name::exif\nmd_src_rating::2\nmd_orientation::landscape\0
```

The name and rating of the MDP that produced the metadata are placed in front of every metadata field. Ratings are offsets in the zero-based list of preferred MDPs, so `0` indicates the first plugin listed, `1` indicates the second listed, and so on.

Retrieving the highest-rated responses

Setting *count* to `0` makes `libmd` collate the responses from many MDPs into one result set to produce the highest-rated response, which is the set of metadata field values obtained from the MDPs listed earliest in the plugin preference order.

Suppose a client sets *count* to `0` and requests the `md_title_width`, `md_title_height`, and `md_title_orientation` fields from a POSIX file while the MDP preference order is the same as listed above. The `MMF` and `MediaFS` MDPs support

the first two fields but not the last; only the `Exif` MDP supports the last field. The `libmd` library sets *md* to reference the following string:

```
md_title_width::response_from_MMF\nmd_title_height::response_from_
MMF\nmd_title_orientation::response_from_Exif
```

Because `MMF` is rated ahead of `MediaFS`, this first MDP's values for `md_title_width` and `md_title_height` are returned. Neither `MMF` nor `MediaFS` supports `md_title_orientation`, so the value from `Exif` for this last field is returned.

Returning metadata memory

The metadata pointer (*md*) should be deallocated using *free()* when the metadata is no longer needed. The `libmd` library sets this pointer to a valid value (i.e., non-NULL) only if the return value is greater than `0`, meaning metadata was found.

Returns:

-1 on error

0 on failure to get metadata (but no errors occurred)

>0 on success (indicating the number of responses)

mmmd_init()

Initialize the library.

Synopsis:

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

int mmmd_init( const char *config )
```

Arguments:

config

The path to the configuration of the library (may be NULL)

Library:

`libmd`

Description:

Initialize the library. You must call this function before any other `libmd` function to initialize the library before using it. This function loads any metadata providers (MDPs) listed in the configuration file into the library. The default path for the configuration file is `/etc/mm/mm-md.conf`.

The plugin entries in the configuration file must contain `dll` attributes that provide filenames that match the plugin names. All other plugin attributes are ignored by the library and may be used by the plugins during metadata extraction.

Returns:

0 on success, -1 on error (*errno* is set)

mmmd_mdps_list()

Get a list of all loaded MDPs.

Synopsis:

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

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

Arguments:

buffer

A pointer to a string for storing the comma-separated list of MDP names (may be NULL)

buf_len

The length of *buffer* (may be 0)

Library:

`libmd`

Description:

Get a list of all MDPs successfully loaded and initialized. This function helps diagnose problems with library initialization.

To obtain the buffer length needed to store the list of all loaded MDPs, call this function with *buffer* set to NULL. Use the return value of this first function call to allocate sufficient buffer memory, then call this function a second time, passing in the updated *buffer* pointer to fill in the list of loaded MDPs.

Returns:

On success, a value greater than or equal to 0 that indicates either the buffer length needed for storing the MDPs list or the amount of data (in bytes) written to the buffer. On error, -1 is returned.

mmmd_session_close()

Close a metadata-extraction session.

Synopsis:

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

int mmmd_session_close( mmmd_hdl_t *hdl )
```

Arguments:

hdl

The handle of the session to close

Library:

`libmd`

Description:

Close a metadata-extraction session.

Returns:

0 on success, -1 on error

mmmd_session_open()

Open a metadata-extraction session.

Synopsis:

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

mmmd_hdl_t* mmmd_session_open( const char *mediastore,
                               uint32_t flags )
```

Arguments:

mediastore

The URL or mountpoint of the mediastore to associate with the session. The syntax of this argument depends on the mediastore type. For example, to read metadata from a USB stick, set this parameter to `/fs/usb0/` (or something similar). To read metadata from files stored in the root directory of your local filesystem, set the parameter to `/`.

flags

Must be 0; reserved for future use

Library:

`libmd`

Description:

Open a metadata-extraction session. The session is associated with the media device named in *mediastore*, meaning that you can use it to read metadata from items stored on that device.

Returns:

A non-NULL session handle on success, NULL on failure (*errno* is set)

mmmd_session_params_set()

Set parameters for a metadata-extraction session.

Synopsis:

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

int mmmd_session_params_set( mmmd_hdl_t *hdl,
                             const strm_dict_t *dict )
```

Arguments:

hdl

The handle of the session whose parameters are being set

dict

A dictionary of key-value pairs representing the parameters. For information on creating dictionaries and storing key-value pairs, see the Dictionary Object API section in the *Multimedia Renderer Developer's Guide*. For information on which keys (i.e., parameter names) are supported by the function, see the *Description*.

Library:

`libmd`

Description:

Set parameters for a metadata-extraction session. Once these parameters are set, they can't be unset or changed. Furthermore, they apply only to MDPs that haven't been used in the current session, so you should call this function just after calling *mmmd_session_open()* (p. 30) but before calling *mmmd_get()* (p. 25).

Currently, only the MMF MDP uses session parameters, which it passes to the Addon Interfaces Library (`libaoi`) when configuring streamers for reading files from HTTP servers. When `libmd` uses other MDPs to read metadata, session parameters defined through this API call have no effect. You should therefore set session parameters only if you want to read metadata from HTTP servers.

The session parameters that you can apply to MMF are the same as the HTTP-related options that you can define as context, input, or track parameters in the Multimedia Renderer Client API.

Returns:

0 on success, -1 on error

mmmd_terminate()

Terminate the library.

Synopsis:

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

int mmmd_terminate( void )
```

Arguments:

(None)

Library:

`libmd`

Description:

Terminate the library from use. You must call this function once, and it must be the last function you call. This function unloads all MDPs from the library.

Returns:

0 on success, -1 on error

md_errors.h

Defines an enumerated type for error codes and a function for obtaining a string from an error code.

Data types in md_errors.h

Data types defined in md_errors.h.

mmmd_errcode

Error codes.

Synopsis:

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

typedef enum mmmd_errcode {
      MMMD_ERR_NONE = 0,
      MMMD_ERR_OTHER,
      MMMD_ERR_NO_MDPS,
      MMMD_ERR_NOT_SUPPORTED,
      MMMD_ERR_MALFORMED_REQUEST,
      MMMD_ERR_NO_PARSERS,
      MMMD_ERR_CALLDEPTH_EXCEEDED,
      MMMD_ERR_NO_MEMORY,
      MMMD_ERR_CANT_OPEN_FILE,
      MMMD_ERR_CANT_READ_FILE,
      MMMD_ERR_CANT_RECONFIGURE,
} mmmd_errcode_t;
```

Data:

MMMD_ERR_NONE

No error occurred.

MMMD_ERR_OTHER

An error not listed here occurred.

MMMD_ERR_NO_MDPS

No metadata plugins are loaded.

MMMD_ERR_NOT_SUPPORTED

The request isn't supported.

MMMD_ERR_MALFORMED_REQUEST

The request isn't properly formed.

MMMD_ERR_NO_PARSERS

No parsers were found for the request.

MMMD_ERR_CALLDEPTH_EXCEEDED

The derived metadata call depth was exceeded (presently not applicable).

MMMD_ERR_NO_MEMORY

No memory is available.

MMMD_ERR_CANT_OPEN_FILE

The file couldn't be opened.

MMMD_ERR_CANT_READ_FILE

The file couldn't be read.

MMMD_ERR_CANT_RECONFIGURE

The configuration was already set (presently not applicable).

Library:

`libmd`

Description:

The enumerated error codes.

Functions in `md_errors.h`

Functions defined in `md_errors.h`.

mmmd_error_str()

Get an English phrase describing the specified error code.

Synopsis:

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

const char* mmmd_error_str( mmmd_errcode_t errcode )
```

Arguments:

errcode

The code of the error you want a descriptive phrase for

Library:

libmd

Description:

This function returns a simple description of the specified error code.

Returns:

A pointer to a string containing the error phrase (always non-NULL)

Index