QNX® Aviage Multimedia Suite 1.2.0

MME Configuration Guide

For QNX® Neutrino® 6.4.x

Published under license by:

QNX Software Systems International Corporation
175 Terence Matthews Crescent
Kanata, Ontario
K2M 1W8
Canada
Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: `info@qnx.com`
Web: `http://www.qnx.com/`

Electronic edition published March 13, 2009.

Contents

List of Figures

About This Guide

The *MME Configuration Guide* accompanies the QNX Aviage multimedia suite, release 1.2.0. It describes how to configre the MME and its components for optimal performance on your target system. It is intended for application developers who use the suite's MultiMedia Engine (MME) to develop multimedia applications.

The table below may help you find what you need in this book:

For information about:	See:
How the MME is configured.	MME Configuration Basics
Recommendations for configuring and troubleshooting the MME.	MME Configuration Tips and Troubleshooting
How to configure QDB to back up MME databases.	Backing up and Restoring MME Databases
Configuring how the MME manages its databases.	Configuring Database Behavior
Configuring how the MME handles device support.	Configuring Device Support
Configuring how the MME manages mediastore synchronizations.	Configuring Synchronization
Configuring how the MME manages metadata support.	Configuring Metadata Support
Configuring playback behavior.	Configuring Playback
Configuring how the MME copies and rips media.	Configurating Media Copying and Ripping
Configuring MME components and other QNX Neutrino components affecting the MME's behavior.	Configuring Other Components
Configuring Internationalization.	Configuring Internationalization
A list of the binary files required.	Appendix: Binary File Dependencies

Other MME documentation available to application developers includes:

Book	Description
Introduction to the MME	MME Architecture, Quickstart Guide, and FAQs.
MME Developer's Guide	How to use the MME to program client applications.

continued...

Book	Description
MME API Library Reference	MME API functions, data structures, enumerated types, and events.
MME Utilities	Utilities used by the MME.
MME Technotes	MME technical notes.
QDB Developer's Guide	QDB database engine programming guide and API library reference.

Note that the MME is a component of the QNX Aviage multimedia core package, which is available in the QNX Aviage multimedia suite of products. The MME is the main component of this core package. It is used for configuration and control of your multimedia applications.

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	**something you type**
Keyboard keys	Enter
Program output	`login:`
Programming constants	NULL
Programming data types	**unsigned short**
Programming literals	**0xFF**, **"message string"**
Variable names	*stdin*

continued. . .

Reference	Example
User-interface components	**Cancel**

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 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 options

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

Chapter 1

MME Configuration Basics

In this chapter. . .

This chapter explains the basics of MME configuration. It includes:

- About configuring the MME
- Configuring memory usage
- Global settings

When you are configuring your MME system, you should make sure that you install all the components required to support the media formats you will be processing. For information about formats supported by third party components and the required MME components for these, see the appendix Binary File Dependencies.

About configuring the MME

The MME offers a wide range of configuration options that you can use to configure the behavior of:

- the MME
- MME components
- plugins
- the QDB

The MME uses two configuration files:

`config.h`	This file contains the configuration values that are loaded into the MME at compile time.
`mme.conf`	This file contains the settings that the MME loads at startup. These values can be modified by the end user and take effect when the MME is restarted.

Compiled configuration

The file **`config.h`** is located in the source tree at **`/lib/public/mme/config.h`**. Changes you may make to **`config.h`** have no effect on the MME. The file lists the values for the MME's CONFIG_* constants, and is included for information only. The MME uses the values in this file if any one of the following conditions apply:

- The MME cannot find the user-accessible configuration file **`mme.conf`**.
- No value for the configuration setting is defined in **`mme.conf`** (e.g. the relevant element is commented out).
- The MME is unable to correctly interpret an element in **`mme.conf`**.

MME command-line options

You can use command-line options to configure the MME at startup. For details, see **mme** in the chapter MME Utilities Reference.

MME user managed configuration

You can change the MME configuration by changing the settings in the user-managed configuration file, **mme.conf**. You can edit this XML file to change user-managed MME settings.

The MME does not include a defined schema for **mme.conf**, but the file must be a well-formed and correct XML file. The main document element in **mme.conf** must be called **<Configuration>**. This element contains hierarchically arranged configuration elements. At the highest level are global configuration settings, such as the initial language (locale) setting. The file also includes elements defining configuration of synchronization, copying, playback, device control, etc.

Location of the configuration file mme.conf

The configuration file **mme.conf** is located at **etc/** (or the equivalent in the staging directory). This directory also includes the sample MCD configuration files **mcd.conf** and **mcd.mnt**.

Default configuration values

The default **mme.conf** file is shipped with all its elements except the base element **<Configuration>** commented out so that, by default, the MME uses no values in this file and uses default settings.

How the MME determines the source of the configuration

The MME uses the following logic to determine how to configure itself on startup:

1. If you specify the **-c** command line option, the MME attempts to use the specified configuration file. If the file does not exist, execution aborts. Note that the specified file may be a relative path or an absolute path.
2. If you don't use the **-c** command line option, the MME looks for the file **/etc/mme.conf**. If the file exists, the MME uses values from any readable elements in that file, and use defaults for all other settings.
3. If you don't use the **-c** command line option and the default file **/etc/mme.conf** does not exist, the MME runs from the defaults as specified in **config.h**.

In summary, if a file is specified, it must exist. If nothing is specified, MME looks for the default configuration file **/etc/mme/mme.conf**. If that file does not exist, the MME uses internal defaults.

Using the configuration file `mme.conf`

You do not need to set values for all elements in **`mme.conf`**. The MME uses only those values for the elements it finds in **`mme.conf`**. If it does not find an element or is unable to read an element in **`mme.conf`**, the MME uses the defaults set at compile time.

This behavior means that:

- Only those values that need to be different from the default values need to be specified in the XML configuration file.
- The values in the XML configuration file are provided as examples to assist users who want changes to the configuration. They are not required by the MME.

Changing the contents of **`mme.conf`** does not automatically change the MME configuration settings. To change the MME configuration, you can either modify the default **`mme.conf`** file, or create and modify a new **`mme.conf`** file and use the **`-c`** option at startup to point the MME to this new file.

Modifying the default `mme.conf` file

To change a configuration value in the configuration file **`mme.conf`**:

1. If necessary, remove the comments around the element and all elements in its path. For example, for the element **`<dvdvideo>`**, which is inside **`<dvdifo>`**, you must make sure that no elements in the path **`<Configuration>`**/**`<Database>`**/**`<Synchronization>`**/**`<MDS>`**/**`<dvdifo>`** are commented so that the MME can find your configured element **`<dvdvideo>`**.
2. Change the configuration settings of the element, as required.
3. Save **`mme.conf`**.
4. Restart the MME.

Configuration element attributes

Some elements in **`mme.conf`** have attributes. The MME accepts the following values for these attributes:

- **`on`**, **`true`**, **`enabled`** and **`yes`** for positive boolean values
- **`off`**, **`false`**, **`disabled`** and **`no`** for negative boolean values

Using a copy of the `mme.conf` file

If you use a copy of the **`mme.conf`** file:

1. Make a copy of the **`mme.conf`** file and save it in a directory of your choosing.
2. Change the configuration settings in the new **`mme.conf`** file.
3. Restart the MME and use the **`-c`** command-line option to tell it to use the modified **`mme.conf`** file.

Configuration elements

All MME configuration elements from **mme.conf** are under the base configuration element Configuration. Unless otherwise stated, a value of 0 (zero) means "Use the default".

In the tables listing configuration elements, element attributes are shown in italics: *attribute*. Elements that require more detailed explanation than can be provided in a table are treated at the end of each section.

Using functions to configure behavior

Many MME functions that configure behavior for specific operations or permissions, such as seek or playback characteristics, device and disk region, or parental control. For more information, see the descriptions for the API functions in the *MME API Library reference*>.

Configuring memory usage

For optimal performance, you should give careful consideration to how you configure memory usage for your MME environement. This section provides some basic information Configuring memory usage for:

- **qdb**
- **io-fs**
- **io-media**

qdb memory usage

The memory used by **qdb** can be influenced by defining caching **PRAGMA**s in the various MME schema files. The **PRAGMA**s control the amount of memory used for caching by SQLite3 library.

The default value for these **PRAGMA**s is 500. Because the MME has many connections to **qdb** opened at any given time, reducing the default caching values can reduce **qdb** memory usage by several megabytes.

Applying the following to the **mme.sql**, **mme_temp.sql** and **mme_library.sql** schemas will limit the number of 1.5K memory pages used for caching on each database connection:

```
PRAGMA cache_size=1;
PRAGMA default_cache_size=1;
```

Reducing **qdb** caching values can affect the performance of queries to the database, and therefore overall MME performance. You should measure performance with different cache values to find the optimal values for your environment.

See also the chapter **PRAGMA** in the QDB Developer's Guide.

io-fs memory usage

A wide range of caching methods can be used for **io-fs**. These methods can be configured by command-line options when starting **io-fs**. When using the **io-fs** as a driver for iPod and PFS devices rather than as a general purpose filesystem, caching can be limited to reduce memory requirements. For example, the following options have been used with success to reduce memory usage with minimal impact on performance:

```
# io-fs-media -cpages=16,bundles=1,wad=0 -ddevice
```

To completely disable caches to **tmpfs**, use the options below.

```
# io-fs-media -cpages=4,bundles=0,wads=0,throngs=0 -ddevice
```

See also **io-fs-media**, **iofs-ipod.so** and **iofs-pfs.so**.

io-media memory usage

The memory used by **io-media** depends on:

- the filters making up the graph implemented by **io-media**
- the size of the queue buffers
- the depth of the queues

A queue consists of multiple buffers of 32K multiplied by the MRA size. The MRA size is dependant on the specific characteristics of the media format (sample size, stereo/mono, etc.). The number of buffers on the queue is configurable via the **io-media.cfg** file.

io-media uses two queues:

- *queue1* — buffers data in the native format of the audio file being read before the data is passed to the audio decoder. By default, *queue1* has 49 buffers.
- *queue2* — buffers decoded (pcm) audio data before the data is passed on to the audio hardware. By default, **queue2** has 8 buffers.

You can use the **io-media** **mmf_graphbuilder** **queue*** options to reduce the size of the queue buffers and the depth of the queues used by **io-media**.

In general, reducing the queue sizes increases the risk of introducing "audio drops" during playback due to CPU load and media access latencies. Whenever you make changes to queue sizes, you should test your environment thoroughly for playback integrity.

See also **io-media-generic**.

Global settings

This section lists global configuration elements.

<ControlContext> element

The **<ControlContext>** element contains global configuration elements. Its *PathPrefix* attribute sets the location for control contexts.

By default, the mountpath to the resource manager API for the MME is **/dev/mme**, and all MME control contexts are created under this path.

You can use the **<ControlContext>** element's optional *PathPrefix* attribute to configure an alternate prefix for the control context path. For example:

```
<ControlContext PathPrefix="/dev/mme_rearseat">
```

The table below lists global configuration elements contained under the **<ControlContext>** element.

Element	Default	Description
<Maximum>	10	Maximum number of control contexts.
<Unblock>	**false**	MME API unblocking capability. See "Enabling the unblock capability" below.

Enabling the unblock capability

Use the **<Unblock>** configuration element to enable the MME function *mme_set_api_timeout()*:

```
<Unblock enabled="true"/>
```

By default, **<Unblock>** is set to **false**, disabling the MME unblocking capability.

Effects of setting the `<Unblock>` element to `true`

Enabling the MME unblock capabilities affects MME behavior as follows:

- It enables the function *mme_set_api_timeout()* used to set timeouts for blocked API calls, and renders meaningful the information delivered by *mme_get_api_timeout_remaining()*.
- A thread blocked on the MME may return from the MME with an EINTR error if:
  - it receives a signal
  - it had set up a kernel timeout
- If the MME fails with an EINTR error, the MME connection handle will be busy for some time, and calls to the MME will return with an EBUSY error.

`<Locale>` element

The **`<Locale>`** element set the default language code. Its default is **`en`** (for English). For more information, see the chapter Configuring Internationalization in this guide.

Global settings constants

The table below lists the defaults for global MME constants.

Constant	Value	Description
CONFIG_DEF_MAX_CONTROLCONTEXTS	10	Maximum control contexts.

Chapter 2

MME Configuration Tips and Troubleshooting

In this chapter. . .

This chapter introduces some configuration techniques that you can use to tune MME performance for the environment in which it will be deployed, and some useful troubleshooting information. It contains the following topics:

- Performance tuning
- Troubleshooting the configuration

Performance tuning

This section describes some configuration techniques that you can use to tune MME performance for the environment in which it will be deployed. It includes:

- Key configuration elements
- Using indexes to optimize synchronization performance
- Adjusting cache size when using USB storage

Key configuration elements

When using the MME in embedded systems with limited resources, imposing limits on specific operations can imnprove system performance and the overall end-user experience.

The MME configuration file provides elements that control set limits to everything from buffer sizes to database usage. Careful configuration of these elements can help optimize the MME's overall resource usage and speed.

This section describes some key configuration elements and what they do. For a complete list of the MME configuration elements and descriptions of how to use them, see the other chapters in this guide. For more information about configuring **`io-media`** see the **`io-media`** chapter in the *MME Utilities reference*; for more information about configuring the QDB, see the chapter Backing up and Restoring MME Databases in this guide, and the *QDB Developer's Guide*.

Synchronization limits

This section describes key synchronization configuration elements.

Folder limits

The greater the number of items an MME sycnhronization examines in a folder, the greater the memory resources it requires for the synchronization. The **`<MaxFolderItems>`** element limits the the number of items the MME examines in any one folder during synchronization.

Folder depth limits

Limiting the depth MME synchronization will go on mediastore's file system may be required to maintain optimal performance. The maximum depth of recursion can be set with the **`<MaxRecursionDepth>`** configuration element. When the MME synchronization process reaches the set folder depth limit, it ends the recursion of the folder chain, returns to the last common folder level, and continues with the next item at that level.

Playlist limits

Playlists created by end users can be extremely large and processing them can strain limited sytem resources. The **`<MaxLines>`** configuration element in the **`<PLSS>`** can be used to set a limit to the number of lines in a playlist the MME will process.

Synchronization buffer limits

MME synchronization performance can be affected by the number of synchronization buffers available to the MME while adding media data to its database. Generally, the more buffers available, the faster the synchronization, but the greater the memory resources required. The **`<MaxSyncBuffers>`** configuration element can be used to increase or decrease the number of synchronization buffers available to the MME.

Database usage limits

No embedded system has an infinite amount of space to store media data collected during synchronizations, and in a system with limited resources, maintaining data for rarely used media can reduce performance, and prevent synchronization and playback of new media. The **`<MaxDatabaseSize>`** configuration element can be used to limit the size of the MME database. When the database reaches or exceeds the set limit, the MME automatically begins pruning old, unused data from the database.

Using indexes to optimize synchronization performance

The MME includes a default set of indexes that has been carefully chosen to maximize MME performance for synchronization, playback and track session handling. These default indexes are in the **`mme.sql`** and **`mme_library.sql`** schema files delivered with the MME.

The effects of indexing on performance depends on a number of factors, including the system platform and the specific contents of mediastores; in particular, for synchronizations, the number and size of folders and files.

Judiciously used, indexes can improve performance. However, adding indexes increases the database size and can, therefore, produce the inverse effect and *degrade* performance:

- Indexes slow down table insertions, and increasing the number of indexes may significantly increase synchronization times.
- A large number of indexes generally *increases* lookup times.

Recommendations

It is expected that you will add indexes to your projects to obtain the best possible performance for each project. To ensure optimal synchronization performance in your environment, we suggest that you consider the recommendations below:

- After installing the **`mme.sql`** and **`mme_library.sql`** schema files, test synchronization performance with these files.
- Review the indexes in your MME implementation and experiment with adding and removing indexes, measuring the effect on synchronization performance. Use the SQLite **`ANALYZE`** command to gather statistics about your indexes as you develop and test them.
- Use indexes sparingly and carefully; whenever possible craft your queries to reuse indexes instead of creating indexes for each query statement.
- If your client application will make queries such as **`SELECT a,b,c WHERE a=something AND b=something_else ORDER BY c`**, and other queries such as **`SELECT a,b,c WHERE a=something AND b=something_else`**, consider creating indexes for *a*, *b* and *c* and test with these, but do not create a second set of indexes for *a* and *b*.

For information about how to add indexes, see "Adding and removing indexes" in the chapter Configuring Database Behavior.

Adjusting cache size when using USB storage

If your environment hosts media files on USB storage devices, you should ensure that your configuration allows sufficient RAM for read-ahead processing of large files, such as MP3 files. You can change the configuration by adjusting the **`cache`** and **`vnode`** values that **`devb-umass`** passes to **`io-blk.so`** with the **`blk`** option.

A reasonable starting configuration for the **`blk`** option is: **`cache=512k,vnode=256`**. You should, however, establish benchmarks for key activities in your environment, then adjust these values for optimal performance.

For more information about the utilities, see **`devb-umass`** and **`io-blk.so`** in the *Neutrino Utilities Reference*.

Troubleshooting the configuration

This section includes:

- MME configuration
- **`io-media`** configuration
- QDB configuration

To get more or less information about your implementation you can adjust verbosity and debug levels. For more information, see:

- **`io-media -D`** and **`-q`** options
- **`mme -v`** and **`-o`** options
- **`qdb -V`** and **`-v`** options
- *mme_set_debug()* — set MME debug settings
- *mme_sync_set_debug()* — set the verbosity level for synchronization operations
- *mme_get_logging()* — get the log levels for specified logging modules
- *mme_set_logging()* — set the log levels for specified logging modules

If you have trouble with your MME installation or with playing media, try the following:

- Check what's running on your target:

  ```
  # pidin arg
  ```

- Start your applications in verbose mode to capture "system-log" output and commands that failed. To start **`io-media`** in verbose mode:

  ```
  # io-media-generic -DD
  ```

  To start **`mme`** in verbose mode:

  ```
  # mme-generic -vvv
  ```

You can use **`sloginfo`** to view messages from the system log.

MME configuration

In all cases where the MME uses assignments from **`mme.conf`**, it makes log entries of the type WARN, followed by the **`mme.conf`** assignment used and its value. If the assignment comes from the user-managed configuration file **`mme.conf`**, the MME makes a log entry of type INFO.

This behavior makes it easier to see what was not taken from the configuration file, which is good because it is relatively easy to make errors in configuration value specification, and little information is available when using a badly formed XML document.

Common errors are:

- A configuration file was specified but was not found or was not a valid XML file. In this case, the MME initialization will have aborted. Correct the file path and/or the XML file structure.

- A configuration file was found (either user-specified or default), but the MME was unable to interpret a value in the file. In this case, the MME will have completed initialization using the default configuration value. Correct the XML for the value.

If the MME is not behaving as expected after a re-boot, you should check the logs to see the source of the assignments. The log type: INFO or WARN, indicates the source of the assignment and the first place to check for an error. If the MME used a default time setting when you expected it to use your configuration, check that:

- the MME has the correct path to the configuration file **mme.conf** (set by the *c* command-line option at startup).
- the element you configured in the **mme.conf** file the MME is using is not commented out and is correctly formed.

Using `mmecli` to view log levels

The MME's command-line interface (**mmecli**) can be used to set and view the MME logging levels. To view current logging levels, use **mmecli get_logging**. For example:

```
# mmecli get_logging
# (rc=0,errno=0) mme@0:0;mdi@0:0;sync@0:0;mdp@0:0.  Execution Time=0.055
```

The above result shows four logging modules: "mme", "mdi", "sync", and "mdp" with all their logging verbosity levels and flags set to 0:0 (verbosity=0 and flags=0).

To set new logging verbosity levels, use **mmecli set_logging**. For example:

```
# mmecli set_logging sync 8 0
# (rc=0,errno=0) logging set to 8 0.  Execution Time=0.001
```

The above example shows how to set the synchronization logging module verbosity to 8. A **get_logging** command would show the new synchronization verbosity level:

```
(rc=0,errno=0) mme@0:0;mdi@0:0;sync@8:0;mdp@0:0.  Execution Time=0.001
```

`io-media` configuration

Use the command-line option **-C** to see the contents of the configuration file for **io-media**:

```
# io-media -C
```

For more information about configuring **io-media**, see **io-media-generic** in the *MME Utilities Reference*.

Log messages implemented for `mpega_parser`

The MP3 parser filter **`mpega_parser`** logs error messages to record conditions such as the ID3Tag in MP3 files exceeding the maximum size handled by the filter, and thus requiring pre-parsing of the tag. For example:

```
Jun 20 14:36:18 5 20 1 io-media-generic/aoi: mpega_parser: ID3TagV2.3 too big (320940bytes) pre-parsing ...
Jun 20 14:36:18 3 20 1 io-media-generic/aoi: PreParseID3Tag() offset 10 truncating at offset on invalid ...
Jun 20 14:36:18 3 20 1 io-media-generic/aoi: mpega_parser couldn't preparse the ID3TagV2.3 Looking for ...
Jun 20 14:36:18 3 20 1 io-media-generic/aoi: ID3Tag V2.3 size 10bytes max = 65536 checking ID3V1 failed ...
```

To have **`mpega_parser`** log these types of messages, start **`io-media`** with at least one level of debug verbosity (at least one **`-D`**).

In a production environment, *always* run the MME, **`io-media`**, and other components and modules at zero verbosity.

QDB configuration

For complete information about configuring the QDB, see the *QDB Developer's Guide*.

QDB and DMS database file attach orders

Different database file attach orders for QDB and DMS result in different locking orders, which cause database deadlocks.

To prevent database deadlocks caused by different database file attach orders, ensure that your projects lock databases in the same order as they are attached:

1. **`mme`** (master)
2. **`mme_temp`**
3. **`mme_custom`**
4. **`mme_library`**

If you don't have an **`mme_custom`** table, use this order:

1. **`mme`** (master)
2. **`mme_temp`**
3. **`mme_library`**

CAUTION: Locking your database files in any other order causes database deadlocks.

Using the client to verify attach order

Before attaching database files in DMS, you can have the client ask QDB the attached order for the files. Below is an example of how to ask QDB the attach order of database files, and the result:

```
qdbc 'pragma database_list;'
Rows: 5  Cols: 3
Names: +seq+name+file+
00000: |0|main|/fs/tmpfs/mme.db|
00001: |1|temp||
00002: |2|mme_temp|/fs/tmpfs/mme_temp|
00003: |3|mme_custom|/fs/tmpfs/mme_custom.db|
00004: |4|mme_library|/fs/tmpfs/mme_library.db|
```

If a file doesn't have a filename (row 1), then don't attach it.

Separating deadlock issues from performance issues

During the development phase of your project you should configure your systems to ensure that you are able to correctly separate performance problems from deadlock problems, and understand and solve each problem accordingly:

- Run your systems with infinite timeouts to ensure that a deadlock is not confused with a performance issue, and is always correctly identified and addressed.
- Enable profiling for queries that take longer than a specified time to execute (for example, 200 milliseconds). If a query takes longer than the specified amount of time, log it as a performance warning, and address the performance issue.

You can change your system configuration when you prepare your system for the production environment.

QDB/MME lockup problem and solution

Testing of the MME has revealed that increasing the MME's maximum number of synchronization threads (**`<MaxThreads>`** in the MME configuration file) without also increasing the maximum number of threads available to the QDB can result in the QDB and MME locking up.

If multiple MME synchronization threads that need to lock the database exhaust the number of available threads, the QDB can find itself without the thread it requires to unlock the database, thus causing a deadlock with the MME waiting for the QDB to unlock the database while the QDB is waiting for the MME to release the thread it needs in order to unlock the database.

Deadlocks like the one described above have been observed in scenarios where the MME has attempted to synchronize multiple and large mediastores at the same time; for example, a USB stick with eight partitions.

Solution

Always ensure that the QDB has a threadpool with more threads than will be required by the MME synchronization threads *plus* any other processes that may lock the MME database. The characteristics of the QDB threadpool can be set with the **`-o thread*`** options; for more information, see the *QDB Developer's Guide* chapter Starting QDB.

Chapter 3

Backing up and Restoring MME Databases

In this chapter. . .

This chapter explains how to configure QDB to back up and restore MME databases, and how to back up and restore MME databases. It includes:

- Configuring the QDB for MME database backups
- Backing up MME databases
- Restoring MME databases
- The QDB configuration file **qdb.cfg**

Configuring the QDB for MME database backups

A database backup is a snapshot of a database, usually RAM-based, that is copied to persistent storage, such as a hard drive or a flash disk. You can use the back up to restore the database after a power cycle, or if the database becomes corrupt.

Even if you run MME databases directly from a hard drive or NAND, you may still want to perform backup and restore operations.

The MME databases are:

- **mme** — configuration information, and dynamic library information
- **mme_library** — library tables
- **mme_temp** — dynamic information that is usually stored in a RAM filesystem

In order to ensure that your MME database backups are usable, you must configure the QDB correctly, and start it in the correct mode:

- Ensure that the configuration block entries in your QDB configuration file: **qdb.cfg** are in exactly the same order as the entries in example **qdb.cfg** file delivered with the MME. See "The QDB configuration file **qdb.cfg**" below.

  The order of the configuration blocks is critical. If the order of these blocks is incorrect, some of the MME database files will have a size of 0 on the filesystem and contain no schema.

- In the QDB configuration file, for each MME database file specify the backup directories for the persistent storage by setting the **Backup Dir** configuration options. Specify more than one directory for each MME database file in order to ensure that there is always one valid backup available, even while a backup is underway. For example, for the **mme_library** file:

  ```
  [mme_library]
  Filename        =  /db/mme_library
  Base Schema     =  /db/mme_library.sql
  Backup Dir     =  /bks1
  Backup Dir     =  /bks2
  Compression     =  bzip
  ```

- Do *not* change the **Backup Attached** options in the configuration file. Leave them at their default settings.

- Start the QDB with:
  - the **-c** specified so that it reads its settings from its configuration file
  - with recovery mode configured to **set**: **-R set** on the command line. This configuration instructs QDB to back up and restore files together rather than separately one at a time, and ensures that all the database files remain consistent with each other over time.

For more information about the QDB, see the *QDB Developer's Guide*.

Backing up MME databases

To back up the MME database files:

1. Shut down the MME by calling *mme_shutdown()* from your client application. This step ensures that the MME does not write to its database files while the back up is underway.
2. Back up the MME database by perform the following sequence:

```
db = qdb_connect("/dev/qdb/mme", NULL);
qdb_backup(db, QDB_ATTACH_DEFAULT);
```

- The backup command must be sent to the *main* MME database (typically called **/dev/qdb/mme**).
- The *scope* argument for *qdb_backup()* must be set to QDB_ATTACH_DEFAULT.

Alternatively, you can start the backup from the command line using the following:

```
# qdbc -a default -B -d /dev/qdb/mme
```

Restoring MME databases

The QDB automatically restores its databases when it starts. It creates databases from one of the following sources, in the following order, as required:

- From a database file that is already in a location where the QDB can run with it (the QDB will not restore over an existing database).
- From the latest complete backup in one of the backup directories specified in the QDB configuration file. For more information about the behavior of this step, see the QDB **-R** option in the chapter Starting QDB of the *QDB Developer's Guide*.
- From the source schema files and data files.

The QDB configuration file `qdb.cfg`

The QDB configuration file **`qdb.cfg`** for the MME defines QDB behavior with MME databases. A sample configuration file is included with the MME. It is organized into configuration blocks. Each configuration block defines how the QDB behaves for the database file identified in the first line of the block.

Commented lines begin with a number sign (“#”) and are ignored by the QDB. In the configuration file delivered with the MME, the **`mme_custom`** database file and all backup directory paths are commented out.

CAUTION: Do *not* change the order of the configuration blocks or of the lines in each block. Changing their order will produce unspecified behavior.

The table below describes the options specified in the QDB configuration file for the MME. Note that many options only apply to the main MME database file **`mme`**.

Option	Description
[*file name*]	The name of the MME database file to which the options that follow apply.
Filename	The path the the MME specified MME database file.
Base Schema	The path to the schema file for the MME database file.
Data Schema	For the **`mme`** database file only, the path to the data schema file.
Client Schema	For the **`mme`** database file only, the path to the client schema file.
Backup Dir	Specify at least two. The path to the directory where the QDB should write MME database backups.
Compression	The type of compression to use for backups. If no compression used the QDB performs no checksums on the data copy. The *lzo* compression algorithm is fastest, but the *bzip* algorithm offers the highest compression.
Vacuum Attached	If set to **`TRUE`**, include the attached database when performing vacuums.
Backup Attached	If set to **`TRUE`**, include the attached database when performing backups.
Size Attached	If set to **`TRUE`**, *qdb_getdbsize()* includes the attached database when it calculates the size of attached databases.
Auto Attach	Attach the specified database when starting up. See “Restoring MME databases” above.

The **Vacuum/Backup/Size Attached** options affect the database file identified by the **Auto Attach** that *follows* them. Thus, in the example configuration file provided below **mme_library** is vacuumed, backed up and sized, while none of these operations is performed on **mme_temp**.

For more detailed information about the QDB configuration file options, see "The configuration file" in the chapter Starting QDB of the *QDB Developer's Guide*.

Example qdb.cfg file

```
#
# QDB Database configuration file for MME
#

[mme_library]
Filename          =  /db/mme_library
Base Schema       =  /db/mme_library.sql
#Backup Dir       =  /bks1
#Backup Dir       =  /bks2
Compression       =  bzip

#[mme_custom]
#Filename         =  /db/mme_custom
#Base Schema      =  /db/mme_custom.sql
##Backup Dir      =  /bks1
##Backup Dir      =  /bks2
#Compression      =  bzip

[mme_temp]
Filename          =  /fs/tmpfs/mme_temp.db
Schema File       =  /db/mme_temp.sql

[mme]
Filename          =  /db/mme
Base Schema       =  /db/mme.sql
Data Schema       =  /db/mme_data.sql
Client Schema     =  /db/mme_connect.sql
#Backup Dir       =  /bks1
#Backup Dir       =  /bks2
Compression       =  bzip
Vacuum Attached =  TRUE
Backup Attached =  TRUE
Size Attached     =  TRUE
Auto Attach       =  mme_library
#Auto Attach      =  mme_custom
Vacuum Attached =  FALSE
Backup Attached =  FALSE
Size Attached     =  FALSE
Auto Attach       =  mme_temp
```

Chapter 4

Configuring Database Behavior

In this chapter. . .

This chapter describes how to add and remove indexes for the MME database, and the database configuration elements that set:

- database mountpath and timeout configuration
- synchronization behavior
- database pruning characteristics

For information about how to configuring QDB to backup MME databases, see the chapter Backing up and Restoring MME Databases.

Database configuration elements

All database configuration elements are under the elements **`<Configuration>`**/**`<Database>`**. The table below lists database configuration elements.

Element	**Attributes**	**Default**	**Description**
`<CharacterEncodingConverter>`	*dll*	n/a	A custom DLL to perform character encoding conversions. See "Creating an external DLL to provide character encoding routines" in the chapter Configuring Internationalization.
`<Mountpath>`		**`/dev/qdb/mme`**	The path to the MME database. See "Database mountpath" below.
`<Timeout>`		0	The database timeout (in milliseconds). See "Database timeout" below.

Database mountpath

You can configure the mountpath to the MME database, and a timeout setting for the database. The mountpath is the location of the MME database. The default path is **`/dev/qdb/mme`**. To change it, change the path in the element **`<Configuration>`**/**`<Database>`**/**`<Mountpath>`**.

Database timeout

The database timeout is the number of milliseconds the database has to complete a request before it fails and returns an error. The default setting is 0 (zero), which disables the timeout. There is no limit on the time the database can use to complete requests.

To enable the database timeout feature, change the value of **`<Timeout>`** to the number of milliseconds you want to give the database to complete requests.

CAUTION: With the timeout feature enabled, database operations that take longer than the timeout period will create errors. Tune the timeout value to meet the needs of your environment, and ensure that your client application handles timeout errors appropriately.

Adding and removing indexes

Indexes can be created on any table for any number of columns, using the **`CREATE INDEX`** statement in a schema, or in your application. Indexes are used automatically by QDB if they are there. An index speeds up any operation where the indexed field is used in a **`WHERE`**, **`ORDER BY`** or **`GROUP BY`** clause. However, the index will slow down any operation that modifies the indexed field.

You should remove indexes your client application doesn't use, and add indexes that will increase performance. Performance can be greatly increased when reading from tables if the columns that are being used for **`WHERE`** and **`ORDER BY`** have an index. **`PRIMARY KEY`** columns do not need an index created on them.

For more information about using indexes to optimize performance, see "Using indexes to optimize synchronization performance" in the chapter MME Configuration Tips and Troubleshooting.

Database pruning

Prune management is a technique for "pruning" (deleting) unused data to ensure that a database doesn't grow too large or exceed a specified size. The MME prunes its database:

- during synchronization operations
- in the background, after a mediastore has been marked as "nonexistent", to prune the nonexistent mediastore

You can set values for the pruning configuration elements that control pruning behavior in **`mme.conf`**.

The table below lists database pruning configuration elements. All pruning elements are under the elements **`<Configuration>`**/**`<Database>`**/**`<Prune>`**.

Element	Attributes	Default	Description
`<MaxDatabaseSize>`		0	The "high-water mark" for the MME database (in kilobytes). If the MME database exceeds this size, the MME's pruning managers will attempt to prune the database. Set to 0 (zero) to disable.

continued...

Element	Attributes	Default	Description
`<MediastoreMatching>`		0	Identify unavailable mediastore types for pruning. See "Pruning options for unavailable mediastores" below.
`<SyncInterval>`		0	Check the size of the MME database after this number of files have been synchronized. Set to 0 (zero) to disable.
`<WhenUnavailable>`	*OnEject*	**`false`**	Determines if the MME automatically prunes entries from its database when mediastores of the specified types are ejected. See "Pruning options for unavailable mediastores" below.

- You can mark specific files, such as ringtone files, as permanent, so that the MME will never prune them. See *mme_set_files_permanent()* in the *MME API Library Reference*.
- You can use *mme_delete_mediastores()* to prune from the MME database entries for mediastores whose state is unavailable, regardless of the settings of **`<WhenUnavailable>`**.

Maximum database size

The element **`<MaxDatabaseSize>`** sets the maximum size in kilobytes of the MME database. During a synchronization, the MME checks the database size periodically, at the interval set by **`<SyncInterval>`**. If the database exceeds the size set by **`<MaxDatabaseSize>`**, the MME attempts to prune the database. It performs the following steps in order until the database size is reduced below the maximum allowed:

1. Clean up the database file structure by calling *qdb_vacuum()*.
2. Delete unavailable mediastores and any associated content.
3. Stop synchronization. In this case, the last synchronized mediastore is only partially synchronized, though it is considered available, and all synchronized content can be played.

The MME checks its database size only when performing synchronizations, at the interval specified by the **`<SyncInterval>`** element.

Interval for checking database size

You can configure the MME to check the size of its database during pruning. Configure this setting by changing the value in the **`<SyncInterval>`** element in the **`mme.conf`** configuration file. The default is 0 (zero): do not check the database size until a synchronization is complete.

If you enable this feature, consider the following impacts on performance:

- The smaller the value of **`<SyncInterval>`**, the more often the MME will check the size of its database and initiate pruning and, therefore, the more slowly synchronizations will progress, but the smaller the amount by which the MME will be allowed to exceed its preferred maximum size.
- Inversely, the larger the value of **`<SyncInterval>`**, the faster synchronizations will progress because the MME will check its database size less frequently, but the greater the amount by which the MME will be allowed to exceed its preferred maximum size.

Pruning options for unavailable mediastores

The MME includes options for managing how mediastore entries are pruned from the MME database:

- About pruning ejected mediastores
- **`<WhenUnavailable>`**
- **`<MediastoreMatching>`**
- Sample script **`mme_del_unav`**

About pruning ejected mediastores

Entries for mediastores can be pruned from the MME database by one of the following methods:

- Automatic pruning conducted by the MME to maintain its database size within configured limits.
- Automatic pruning by the MME when a mediastore is ejected from the system.
- Pruning initiated by the client application, using the *mme_delete_mediastores()* function.

The MME or the client application calling *mme_delete_mediastores()* can prune mediastore entries from the MME database when the following conditions are met:

- The mediastore is of a storage type (MME_STORAGETYPE_*) listed in one of the **`<MediastoreMatching>`** elements.
- The mediastore has no **`library`** table entries marked as permanent. See *mme_delete_mediastores()* for information about overriding this condition.

The MME's automatic pruning behavior for ejected mediastores is as follows:

- The MME will prune an entry for an ejected mediastore from its database *only* if the MME is configured with the **`<WhenUnavailable>`** element's *OnEject* attribute is set to **`true`**.

- When the **`<WhenUnavailable>`** element is configured to enable pruning of ejected mediastores, the state of an ejected mediastore is set to **`non-existent`**.
- A pruned mediastore entry is not deleted from the database immediately. Its *identifier* and *driver_identifier* are set to NULL, and it is later deleted by the background pruning thread.

- The **`<WhenUnavailable>`** and **`<MediastoreMatching>`** configuration elements do *not* affect pruning activities initiated by the MME to keep its database size within configured limits.
- When the MME is configured to *not* prune entries for ejected mediastores from its database, the state of an ejected mediastore is set to **`unavailable`**.

`<WhenUnavailable>`

The **`<WhenUnavailable>`** configuration element determines if the MME automatically prunes entries from its database when mediastores of the specified types become unavailable (are ejected from the system).

The **`<WhenUnavailable>`** element has one attribute: *OnEject*, and can contain multiple **`<MediastoreMatching>`** elements:

- The *OnEject* attribute controls whether or not a mediastore of a type specified by a **`<MediastoreMatching>`** element is pruned when the media store becomes unavailable (on ejection).
- The **`<MediastoreMatching>`** elements specify which mediastore types can be pruned from the database when they become unavailable.

The default setting for the *OnEject* attribute is **`false`**: don't automatically prune mediastore entries from the MME database when mediastores are ejected from the system.

The example below shows the configuration for automatically pruning library entries for ejected mediastores of type MME_STORAGETYPE_FS (2) and MME_STORAGETYPE_DVDVIDEO (6) when these types of mediastores are ejected from the system:

```
<Database>
...
    <Prune>
    ...
<WhenUnavailable OnEject="true">
<MediastoreMatching storagetype="2"/>
<MediastoreMatching storagetype="6"/>

</WhenUnavailable>
    </Prune>
</Database>
```

The **`<WhenUnavailable>`** element does not effect automatic pruning conducted by the MME to maintain its database size within configured limits.

`<MediastoreMatching>`

The **`<MediastoreMatching>`** configuration element identifies mediastore types for which entries for unavailable mediastores can be pruned from the MME database by the MME's automatic pruning or by the client application calling *mme_delete_mediastores()*.

For each mediastore type for which you want to delete entries from the the MME database when a mediastore is ejected, create a **`<MediastoreMatching>`** element and set its *storagetype* attribute to the appropriate MME_STORAGETYPE_* value. See MME_STORAGETYPE_* in the *MME Developer's Guide* for a complete list of MME_STORAGETYPE_* values.

For example, the following configuration would prune audio CDs (1), disk-based mediastores (2), audio DVDs (3), video DVDs (6), and iPods (8):

```
<WhenUnavailable OnEject="true">
    <MediastoreMatching storagetype="1"/>
    <MediastoreMatching storagetype="2"/>
    <MediastoreMatching storagetype="3"/>
    <MediastoreMatching storagetype="6"/>
    <MediastoreMatching storagetype="8"/>
</WhenUnavailable>
```

Sample script `mme_del_unav`

The sample script below instructs the MME to delete all unavailable mediastores, subject to the restrictions set by the **`<MediastoreMatching>`** configuration elements.

`mme_del_unav`

```
#!/bin/ksh
#
# This script is used to tell the MME to delete all unavailable media stores.
#
# If the MME is not available when the script is executed, it does one of two
# things, depending on the use of the '-nowait' directive.
# If the -nowait directive is specified, the script returns an error.
# If the -wait directive is specified, it waits for the path for the number
# of seconds in the time-out value. When connected, it then waits for the
# same time before executing the command.

# Change this to set the default wait time without using a parameter.
DEF_WAIT_TIME=60

# Set defaults
MME_PATH=/dev/mme/default
```

```
QDB_PATH=/dev/qdb/mme
WAIT_TIME=DEF_WAIT_TIME
SHOW_HELP=0

# Collect options
while getopts d:m:w:h o ; do
   case $o in
   d) QDB_PATH=$OPTARG;;
   m) MME_PATH=$OPTARG;;
   w) WAIT_TIME=$OPTARG;;
   h) SHOW_HELP=1
   esac
done
shift $OPTIND-1

# The help message output...
usage () {
    echo "$0 - a utility to tell the MME to delete all unavailable media stores"
    echo "Usage: $0 -h | [ [-m <MME path>] [-d <QDB path>] [-w <seconds>] ]"
    echo "where:"
    echo " -m <MME path> is the path to the MME (default \"/dev/mme/default\")"
    echo " -d <QDB path> is the path to the QDB database (default \"/dev/qdb/mme\")"
    echo " -w <seconds> is the number of seconds for two things."
    echo "    First, it’s the maximum time it will wait to connect to the MME"
    echo "    if it’s not available when the script start."
    echo "    Second, it’s the maximum time it will wait after connecting"
    echo "    to the MME before telling it to delete unavailable."
    echo "    media stores (default $DEF_WAIT_TIME)"
    echo " -h prints this"
}

# If help requested, only do that.
if [ $SHOW_HELP -eq 1 ]
then
    usage
    return 0
fi

# The command to be executed.
CMD="mmecli -c $MME_PATH -d $QDB_PATH delete_mediastores"

# Do it now if not supposed to wait.
if [ $WAIT_TIME -eq 0 ]
then
    $CMD
    return $?
fi

# Wait on the path.
waitfor $MME_PATH $WAIT_TIME
if [ $? -ne 0 ]
then
```

```
    # Already logged by waitfor
    return $?
fi

# Path was good, wait the required time.
sleep $WAIT_TIME
if [ $? -ne 0 ]
then
    # Don’t know if this is already logged!
    echo "Error sleeping..."
    return $?
fi

# Path was good and waited the required time, so execute the command
$CMD
return $?
```

Database constants

The table below lists the constants in **config.h** with the MME compile-time settings for database management.

Constant	Value	Description
CONFIG_DEF_MAX_DATABASE_SIZE	0	The database size, in kilobytes, above which the MME triggers prune management. See “Maximum database size” above. Set to 0 (zero) to disable.
CONFIG_DEF_SYNC_INTERVAL	0	The number of files to synchronize before checking database size against maximum allowed. See “Interval for checking database size” above. Set to 0 (zero) to disable.
CONFIG_DEF_DATABASE_MOUNTPATH	**/dev/qdb/mme**	The default path to the MME’s database.
CONFIG_DEF_DATABASE_MOUNTPATH_LEN	256	The maximum length, in bytes, for the MME’s database mountpath
CONFIG_DEF_DATABASE_SETNAME	mme	The database backup name.
CONFIG_DEF_DATABASE_SETNAME_LENGTH	32	The maximum length, in bytes, of the name of a database backup set.
CONFIG_DEF_DATABASE_TIMEOUT_MS	0	The timeout setting for database access, in milliseconds. Set to 0 (zero) to disable.

Chapter 5
Configuring Device Support

In this chapter. . .

This chapter describes how to configure the MME to detect media devices.

Mediastore detection overview

The Media Content Detector (MCD) monitors the appearance and disappearance of paths on your system, and you must have it configured correctly to inform the MME of the insertion, availability, and removal of mediastores. The paths that the MCD monitors are specified in its configuration file (the default file is **`etc/mcd.conf`**).

You must also configure the MME's **`slots`** table to associate mediastores with the devices that provide them.

For example, for the MME to find USB mediastores, you need to add an entry to the MCD configuration file *and* add a corresponding entry for every USB device to the **`slots`** table. You must have an entry like the following in the MCD configuration file:

```
[/fs/hdumass*]
Callout= PATH_MEDIA_PROCMGR
Argument= /proc/mount
Priority= 11,10
Start Rule= INSERTED
Stop Rule= EJECTED
```

The above example tells the MCD to monitor all **`/fs/hdmass*`** paths, which include **`/fs/hdumass10-dos-1`** and **`/fs/hdumass20-dos-1`**.

For the MME to treat these paths as mediastores, you must add to the **`slots`** table an entry with the exact path for each device:

```
INSERT INTO slots(path,zoneid, name, slottype)
    VALUES('/fs/hdumass10-dos-1', 1, 'USB', 1);
INSERT INTO slots(path,zoneid, name, slottype)
    VALUES('/fs/hdumass20-dos-1', 1, 'USB', 1);
```

The MCD can use wildcards, but the **`slots`** table requires an entry for every device.

Device configuration elements

The table below lists device configuration elements. Device configuration elements are under the elements **`<Configuration>`**/**`<Devices>`**.

Element	Attributes	Default	Description
`<ExternalChanger>`	*enabled*	**`false`**	Determine if the MME supports external changers. Set to **`true`** to enable external CD changer support.

continued...

Element	Attributes	Default	Description
`<InternalChanger>`	*enabled*	**`false`**	Determine if the MME supports internal CD changers. Set to **`true`** to enable internal CD changer support.
	priority	0	Set the absolute priority for the CD changer monitor thread. Its default value is 0: do not set the priority.
`<MCD>`	*root*	**`/dev/mcd/`**	See "Mediastore detection path" below.
	insert_active	**`INSERTED`**	
	insert_available	**`AVAILABLE`**	
	ejection	**`EJECTED`**	
	hostpath	**`""`**	

The configuration element **`<DeviceDetection>`** is under the elements **`<Configuration>`**/**`<Database>`**/**`<Synchronization>`**.

Mediastore detection path

The MME monitors the file descriptors provided by the Media Content Detector (**`mcd`**) utility for mediastore (device) availability. It needs to know when a mediastore is:

- inserted and active
- inserted and available, but not active
- removed

The paths for these file descriptors are defined by the **`<MCD>`** element. You can change this element to set:

- the root path to the file descriptors; note that all descriptors must have the same root path.
- the names of the file descriptors

For example:

```
<MCD root="/dev/mcd/"
    insert_active="INSERTED"
    insert_available="AVAILABLE"
    ejection="EJECTED"
    hostpath=""/>
```

Device paths and filesystem mountpaths

The MME can accept the path of a device, a filesystem mountpath or a node path for device detection. The MME resolves the device path to a mountpath and uses that as the path to the media store.

This capability simplifies MCD configuration for the MME, and allows more options for system configuration. The MCD configuration options for the MCD include:

- Use the CD_MEDIA_IOBLK callout on the CD device path. In this case, the filesystem mountpath may be fixed or may come and go, and MME operation is unaffected. This also allows other mechanisms to mount and unmount the CD's filesystem path.
- Use the MCD to mount and unmount the CD's filesystem path.
- Other uses via the MCD callout DLLs.

Configuring mediastore monitoring paths

If you have not otherwise configured the MME's mediastore path monitoring, the MME expects to monitor files at the default locations:

- **`/dev/mcd/INSERTED`** for path insertion events when the mediastore is active
- **`/dev/mcd/EJECTED`** for path removal events
- **`/dev/mcd/AVAILABLE`** for path insertion events when the mediastore is availabe but not active

If you have configured the **`mcd`** utility for other than the default paths, you must configure **`<MCD>`** elements in the **`<Devices>`** element of the MME's configuration file, where the optional attributes *root*, *insertion*, and *ejection* specify the root path of the three files, the insertion file, the ejection file, and the available file respectively. If any of these optional attributes is missing, the MME uses the default.

To configure mediastore monitoring paths, provide one **`<MCD>`** element for each **`mcd`** process to be monitored. For example:

```
<Devices>
    <MCD root="/dev/mcd/" insert_active="INSERTED"
    insert_available="AVAILABLE" ejection="EJECTED"/>

    <MCD root="/dev/mcd2/" insert_active="incoming"
    insert_available="available" ejection="outgoing"/>
...
</Devices>
```

File names in the QNX 4 filesystem used by the MME are *case sensitive*, but you can use upper-case or lower-case letters for your file names: “INSERTED”, “Inserted” and “inserted” are all valid but *different* file names.

Configuring multi-node support

The MCD offers multi-node support. To enable multi-node support, for *each MCD instance on each node* on which you want to access mediastores (including the local node running the MME), add an **`<MCD>`** element, with the attribute *hostpath* set to the path for the node you want to access.

Examples 1 to 4 assume that all MCD instances are using the default configuration. To use another MCD configuration, add the appropriate attributes to the **`<MCD>`** element for the MCD instance for which you want to change the configuration, as shown in Example 5.

Example 1

To enable access to mediastores only on the remote node “yournode.yourdomain”:

```
<MCD hostpath="/net/yournode.yourdomain"/>
```

Example 2

To enable access to mediastores on the remote node “yournode.yourdomain”, *and* on the local node with the MME:

```
<MCD hostpath="/net/yournode.yourdomain"/>
<MCD/>
```

Example 3

To enable access to mediastores on the remote node “yournode.yourdomain”, on “someothernode.otherdomain” with two instances of the MCD, and on the local node with the MME:

```
<MCD hostpath="/net/yournode.yourdomain"/>
<MCD hostpath="/net/othernode.otherdomain"/>
<MCD root="dev/mcd2/" hostpath="/net/othernode.otherdomain"/>
<MCD/>
```

Example 4

If you access only local mediastores with one instance of the MCD, you can either omit the **`<MCD>`** element, or use one tag with the *hostpath* attribute set to blank:

```
<MCD hostpath=""/>
```

Example 5

To enable access to a mediastores on the remote node “yournode.yourdomain” and on the local node with the MME, and change the configurations for these MCD instances, you might use something like:

```
<MCD root="/dev/mcd/"
    insert_active="present"
    insert_available="usable"
    ejection="not_present"
    hostpath=""/>

<MCD root="/dev/mcd/"
    insert_active="present"
    insert_available="usable"
    ejection="not_present"
    hostpath="/net/yournode.yourdomain"/>
```

Configuring the `slots` table for multi-node support

If you enable multi-node support, to access mediastores you must update the information in the **slots** table *path* field to include the full path to all the mediastores (including those on the node with the MME), as this path is resolved by the MCD. For example:

```
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/media/drive', 1, 'HardDrive', 3);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/usb0', 1, 'USB', 1);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/usb1', 1, 'USB', 1);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/cd0', 1, 'CD/DVD', 2);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/ipod0', 1, 'iPod', 4);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/pfs1', 1, 'PlaysForSure', 4);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/net/yournode.yourdomain/fs/upnp0', 1, 'UPnP', 4);
INSERT INTO slots(
    path,zoneid, name, slottype)
    VALUES('/dev/wms/player1', 1, 'Bluetooth', 9);
```

The path assigned to the *hostpath* attribute does *not* have a trailing slash ("/"):

Correct: **`<MCD hostpath="/net/yournode.yourdomain"/>`**

Incorrect: **`<MCD hostpath="/net/yournode.yourdomain/"/>`**

Configuring the `mcd` utility

Mediastore path monitoring requires that you configure the **`mcd`** utility. A default configuration file, **`mcd.conf`**, is provided as a starting point. You may make copies of these files and change their contents to meet your system's requirements. Use the **`mcd config_file`** option to point the **`mcd`** to your configuration file.

The MME Quickstart Guide in *Introduction to the MME* provides an example of how to use the MCD configuration files.

The rules provided below are examples of rules to include in your MCD configuration file in order to handle different types of mediastore devices.

HDD insertion and removal

This rule may be used to provide insertion and removal events for a hard drive. The name of the folder on the filesystem that is the root of the media storage location for the hard drive should be between the square brackets.

```
[/media/drive]
Callout      = PATH_MEDIA_SCAN
Argument     = 2000
Priority     = 11,10
Start Rule   = INSERTED
Stop Rule    = EJECTED
```

The argument is the polling time, in milliseconds. For "permanent" mediastores, you may wish to increase this value.

USB device insertion

This rule may be used to detect USB device insertion and removal.

```
[/fs/usb*]
Callout = PATH_MEDIA_PROCMGR
Argument = /proc/mount
Priority = 11,10
Start Rule = INSERTED
Stop Rule = EJECTED
```

iPod device insertion

This rule may be used to detect iPod device insertion and removal.

```
[/fs/ipod*]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = INSERTED
Stop Rule   = EJECTED
```

Internet connection

This rule may be used to detect an internet connection.

```
[/dev/socket]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = INSERTED
Stop Rule   = EJECTED
```

CD mediastores

For CDs, there are a number of options, depending on system configuration. Two options are shown below.

CD Option 1

This rule is used if the CD's filesystem is mounted by a mechanism other than the MCD. This rule uses the MME's ability to be told about the device path, rather than the mountpath of a media store.

```
[/dev/cd*]
Callout     = CD_MEDIA_IOBLK
Argument    = 1000,2000
Priority    = 11,9
Start Rule  = INSERTED
Stop Rule   = EJECTED
```

CD Option 2

These rules are used for a CD if the MCD is needed for mounting and unmounting the CD's filesystem. Note that the MOUNT_FSYS callout requires a configuration file as well.

Please see the MCD documentation for more information on the format of the MOUNT_FSYS configuration file.

```
[/dev/cd*]
Callout     = CD_MEDIA_IOBLK
Argument    = 1000,2000
Priority    = 11,9
```

```
Start Rule  = MOUNT
Stop Rule   = UNMOUNT

[/fs/cd*]
Callout     = PATH_MEDIA_PROCMGR
Argument = /proc/mount
Priority    = 11,9
Start Rule  = INSERTED
Stop Rule   = EJECTED

[MOUNT]
Callout     = MOUNT_FSYS
Argument    = /dev/shmem/mcd.mnt

[UNMOUNT]
Callout     = UNMOUNT_FSYS
```

Creating the MCD file for active device insertions

This rule is required in order to create the MCD file that is used for device insertions when the device is active. The default name is shown here; if the default name is not used, the MME's configuration file must be used to tell the MME the correct name.

```
[INSERTED]
```

Creating the MCD file for device removals

This rule is required in order to create the MCD file that is used for device ejections. The default name is shown here; if the default name is not used, the MME's configuration file must be used to tell the MME the correct name and path.

```
[EJECTED]
```

Creating the MCD file for available (but not active) device insertions

This rule is required in order to create the MCD file that is used for device insertions when the device is available, but not active. The default name is shown here; if the default name is not used, you must use the MME's configuration file to tell the MME the correct name.

Even if this rule is not used, the MME requires that it appear in the MCD configuration file.

```
[AVAILABLE]
```

Sample `mcd` configuration

Below is a sample **mcd** configuration file. You can copy the contents of this file into a configuration file, then point the **mcd** utility when you start it. This configuration assumes that **/fs/cd*** is already mounted.

```
[/dev/umass*]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = MOUNT
Stop Rule   = UNMOUNT

[MOUNT]
Callout     = MOUNT_FSYS
Argument    = /etc/mcd.mnt

[UNMOUNT]
Callout     = UNMOUNT_FSYS

[/media/drive]
Callout     = PATH_MEDIA_SCAN
Argument    = 5000
Priority    = 11,10
Start Rule  = INSERTED
Stop Rule   = EJECTED

[/dev/cd*]
Callout     = CD_MEDIA_IOBLK
Argument    = 1000,2000
Priority    = 11,9
Start Rule  = INSERTED
Stop Rule  = EJECTED

[/fs/usb*]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = INSERTED
Stop Rule   = EJECTED

[/fs/ipod*]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = INSERTED
Stop Rule   = EJECTED

[/fs/pfs*]
Callout     = PATH_MEDIA_PROCMGR
Argument    = /proc/mount
Priority    = 11,10
Start Rule  = INSERTED
```

```
Stop Rule   = EJECTED

[INSERTED]

[EJECTED]

[AVAILABLE]
```

For more examples, see the MME sample configuration file **mcd.conf** for the MCD. For more information about the MCD, see **mcd** in the *MME Utilities Reference*.

The sample configuration file **2phase.cfg** delivered with the MCD does *not* work for the MME.

Device constants

The table below lists the default settings for the MME's device management constants.

Constant	Value	Description
CONFIG_DEF_DEVS_INT_CHANGER_ENABLED	**false**	Enable internal CD changer support.
CONFIG_DEF_DEVS_EXT_CHANGER_ENABLED	**false**	Enable external CD changer support.
CONFIG_DEF_DEVICE_DETECTION	**true**	Enable automatic device detection.
CONFIG_DEF_DEVICE_MONITOR_PRIORITY	0	Priority for MCD monitor thread (to detect new devices and mediastores). Set to 0 (zero) to use inherited default priority.
CONFIG_MAX_OUTPUTNAME_LENGTH	1024	Maximum length for name (including path) of output device.

Configuring the **slots** table for supported devices

You can customize the **slots** table in the schema (**mme.sql**) to associate the mediastores in your MME system with the devices that provide them. By default, the hard drive location is defined as:

```
INSERT INTO slots(path,zoneid, name, concurrency, slottype)
   VALUES('/media/drive', 1, 'HardDrive', 0, 3);
```

There are similar entries for CD/DVD, USB, PFS, UPnP, and iPod devices, and internet connections. As required, you can change the path or name of a device, or remove a device altogether. To add or modify entries for devices, insert a row with the device information into the **slots** table, using the example above as a template. To remove a device so that the MME no longer recognizes it, delete the relevant row from the **slots** table.

Using output devices accessible over Qnet

You can use an audio output device that is accessible over Qnet by specifying the full path to the device in your client application, or by setting the device path in the **`mme_data.sql`**. For example, by changing **`/dev/snd/pcmC0D1p`** to **`/net...`***full path* **`.../dev/snd/pcmC0D1p`**.

Chapter 6

Configuring Media Synchronizations

In this chapter. . .

This chapter explains how to configure the MME synchronization behavior.

You can configure the MME synchronization processes to optimize the use of system resources in your environment. Many default values, such as thread priority and folder depth are set as **mme** startup options. See **mme-generic** in the *MME Utilities Reference* for details.

The <Synchronization> elements

The table below lists the first level of synchronization configuration elements under the **<Configuration>**/**<Database>**/**<Synchronization>** element.

Element	Attributes	Default	Description
<Automatic>	*enabled*	**true**	Automatically initiate synchronizations on newly detected mediastores. See "Automatic and on-demand synchronizations".
<ChangedFilesHaveConstantId>	*enabled*	**off**	Determine how MME synchronization handles files whose size or date has changed. See "Configurable handling of updated files and folders" below.
	options	**all,recursive**	When automatic synchronization is enabled, determine specific behaviors. See "Automatic and on-demand synchronizations".
<DelayAfterMerge>		0	The time, in milliseconds, to delay after completing a synchronization merge operation.
<DeviceDetection>	*enabled*	**true**	Automatically detect devices. See "Device detection" below. This setting is independent of synchronizations.
<Events>		off	If the *folder* attribute is set to "on" send events at start and completion of folder synchronizations.

continued. . .

Element	Attributes	Default	Description
`<extensions>`		empty	If sub-elements are specified, synchronize only media files with listed extensions for the specified sub-element. See "Filtering synchronization by file type" below.
`<ForceAccurateAfterFullSync>`		**`false`**	Set the **`mediastores`** table *accurate* column to 1 (accurate) after a full synchronization of a mediastore.
`<MaxFolderItems>`		0	Limit the number of items the MME will synchronize in any one folder. See "Limiting the number of items synchronized in a folder" below.
`<MaxLines>`		5000	The maximum number of lines (files listed) permitted for a playlist. If used, this value overrides the value set by CONFIG_MAX_PLAYLIST_LINES in **`config.h`**.
`<MaxRecursionDepth>`		8	The maximum depth (number of folders) to recurse when synchronizing media.
`<MaxSyncBuffers>`		250	The maximum number of synchronization records in synchronization buffers between the foreground and background synchronization threads.
`<MaxThreads>`		8	The maximum number of synchronization threads allowed to run in the foreground.
`<MonitorPriority>`		0	The priority of the device-detection thread.
`<PassTwoSetsLastSync>`		**`true`**	Have the synchronization metadata pass (pass 2) update the *last_sync* column in the **`library`** table. Caution: see "Item selection based on time of entry" below.

continued. . .

Element	Attributes	Default	Description
`<PLSS>`		n/a	Container element for playlist configuration elements.
`<Priority>`		0	The priority of synchronization threads. Set to 0 to use **`mme`** startup options. See “Configuring synchronization thread priorities” below.
`<SyncFileMask>`		empty	Don’t synchronize files with this string in their names. See “Configurable file skipping: **`<SyncFileMask>`**” below.
`<UnknownTitlesNull>`		**`enabled`**	If enabled, use NULL values for unknown file names. If disabled, build file names. See “Configurable handling of unknown track titles” below.

Configuration elements that are used to configure metadata support are described in the chapter Configuring Metadata Support. These elements include some elements under the **`<Synchronization>`** element.

Synchronization thread priorities

Adjusting the priorities of the synchronization threads can have significant impact on system performance. Note that:

- By default, foreground synchronization threads run at the priority that the MME’s **`mme`** module uses at startup.
- Running the foreground synchronization threads at a priority two *lower* than the default priority for the MME may significantly reduce delays when changing tracks. For example, if the MME’s **`mme`** module is running at priority 10, set the prioity of foreground synchronization threads to 8: **`<Priority 8>`**/**`<Priority>`**.
- The merge thread is a background synchronization thread that writes entries in the MME’s **`library`** table. It runs at a priority one greater than the default **`mme`** priority. For example, if the foreground synchronization threads run at priority 10, the merge thread runs at priority 11.

Device detection

The **`<DeviceDetection>`** element can be used to change the MME’s default behavior and have it detect devices *only when specifically requested to do so by the client application*.

If you have set **`<DeviceDetection>`** to **`false`** to configure the MME *not* to automatically detect devices and mediastores, you must call the function *mme_start_device_detection()* to start device and mediastore detection.

CAUTION: Failure to call *mme_start_device_detection()* before attempting any task that accesses mediastores will produce unexpected results that may compromise the integrity of your system.

Automatic and on-demand synchronizations

By default, the MME automatically synchronizes any newly detected mediastore. However, you can configure the MME to require an explicit request from the client application to synchronize a mediastore by setting the **`<Automatic>`** element's *enabled* attribute to **`false`**.

This configuration prevents the MME from performing a synchronization on every mediastore change, and makes the client application responsible for selecting what levels of synchronization are done on what mediastores and at what times. The MME waits to synchronize a mediastore until the client application calls *mme_resync_mediastore()* to request a synchronization.

The `<Automatic>` element's *options* attribute

When automatic synchronization is enabled (the default), the **`<Automatic>`** element's *options* attribute determines the tasks to perform when synchronizing a mediastore. It can be assigned a comma-separated list of words. For example, the default configuration, which is to perform all configuration tasks recursively, is configured as follows:

```
<Automatic enabled="true" options="all,recursive"/>
```

If the *options* attribute is *not* specified, the MME synchronization assumes the default confirguration and performs all synchronization passes recursively.

The words that can be used for the *options* attribute and their implications for synchronizations are listed below:

- **`files`** — perform the first synchronization pass (file and folder discovery).
- **`metadata`** — perform the second synchronization pass (metadata update).
- **`playlists`** — perform the third synchronization pass (playlist update).
- **`ext_db_sync`** — perform the external synchronization pass (inform external entities).
- **`all`** — perform all synchronization passes.
- **`recursive`** — synchronize all folders (beyond the root folder) on the mediastore.

Filtering synchronizations

This section describes configuration options for filtering synchronizations. It includes:

- Filtering synchronization by storage type
- Filtering synchronization by file type
- Configurable file skipping
- Configurable handling of updated files and folders

Filtering synchronization by storage type

You can configure the MME to *not* synchronize specified storage types. The **`<nosync>`** element accepts **`<path>`** elements, which have *storagetype* attributes.

The `<nosync>` and `<path>` elements

The **`<MSS>`**/**`<nosync>`** element is a container element for one or more **`<path>`** elements specifying storage types that should not be synchronized. Two default **`<path>`** elements are listed in the table below:

Element	Attribute	Default	Description
`<path>`	*storagetype*	17	Don't synchronize navigation storage types (NAV/MAP0001).
`<path>`	*storagetype*	18	Don't synchronize upgrade storage types (UPDATE/DATA01.DAT).

Using `<path>` elements

If any of the paths specified in the **`<path>`** elements exist on a mediastore, the MME:

- doesn't scan the mediastore
- inserts an entry for the mediastore in the **`mediastores`** table, with the specified storage type

The paths specified in the **`<path>`** elements are relative to the mediastore root folder.

The MME can be configured to not synchronize the following storage types:

- Navigation: .../**`<MSS>`**/**`<nosync>`**/**`<path`** *storagetype*=**`"17">`** (NAV/MAP0001)
- Upgrade: .../**`<MSS>`**/**`<nosync>`**/**`<path`** *storagetype*=**`"18">`** (UPDATE/DATA01.DAT
- Custom storage types: .../**`<MSS>`**/**`<nosync>`**/**`<path`** *storagetype*=**`"100">`**

The default MME configuration is to not synchronize navigation and upgrade storage types.

Composing `<nosync>` pathnames

When composing pathnames for the **`<nosync>`** element, note that the pathname:

- must *not* be prefixed by a forward slash ("/"):
  - permitted: **`CD_INFO.CDI`**
  - *not* permitted: **`/CD_INFO.CDI`**
- *may* contain forward slashes, or be suffixed by a forward slash, or both:
  - permitted: **`DIR_UNDER_ROOT/`**
  - permitted: **`NAVI/NAVI.001`**
  - permitted: **`DIR_UNDER_ROOT/OTHER_DIR/`**

Examples

Navigation storage type:

```
<nosync>
<path storagetype="17">NAV/MAP0001</path>
</nosync>
```

If a mediastore has a **`NAV/MAP0001`** folder on it, the MME marks the mediastore in the database as MMPL_STORAGETYPE_NAVIGATION and does not synchronize it.

For example, if a CD is inserted it will not be synchronized if:

- it is mounted at **`/fs/cd0`**

  and:
- it has a folder **`/fs/cd0/NAV/MAP0001`**

Upgrade storage type:

```
<nosync>
<path storagetype="18">UPDATE/DATA01.DAT</path>
</nosynch>
```

If a mediastore has a **`UPDATE/DATA01.DAT`** file on it, the MME marks the mediastore in the database as MMPL_STORAGETYPE_UPGRADE and does not synchronize it.

For example, if a CD is inserted it will not be synchronized if:

- it is mounted at **`/fs/cd0`**

  and:
- it has a file **`/fs/cd0/UPDATE/DATA01.DAT`**

Custom storage type:

```
<nosync>
<path storagetype="100">SOME/DIRECTORY/FILE.INFO</path>
</nosynch>
```

If a mediastore has a **SOME/DIRECTORY/FILE.INFO** file on it, the MME marks the mediastore in the database as 100 (MMPL_STORAGETYPE_CUSTOM1) and does not synchronize it.

For example, if a USB disk on key is inserted, it will not be synchronized if:

- it is mounted at **/fs/cd0**

  and:

- it has a file **/fs/cd0/SOME/DIRECTORY/FILE.INFO**

In all cases:

- the mediastore is added to the **mediastores** table
- the mediastore content is *not* synchronized

Filtering synchronization by file type

The MME offers client application developers the ability to choose the file types that the MME synchronizes. To have the MME synchronize a file type, you need only to list the file type in the **<extensions>** element in the MME configuration file **mme.conf**.

The table below lists the elements inside the **<extensions>** element:

Element	Attributes	Default	Description
<library>		empty	If specified, synchronize only media files with the listed extensions.
<playlists>		empty	If specified, synchronize only playlist files with the listed extensions.

The default (i.e. elements not specified) is to synchronize all media files. If you specify the **<playlist>** or **<library>** element, you tell the MME to synchronize only files that match the extensions listed. You must, therefore, explicitly list the extensions for all the media file types you want to synchronize.

Library extensions require an *ftype* attribute specifying the ftype field in the library. The *ftype* must be one of:

- **audio**
- **photo**
- **video**

For example:

```
<library>
<!-- Audio types -->
<extension value="3g2"   ftype="audio"/>
<extension value="3gp"   ftype="audio"/>
<extension value="aac"   ftype="audio"/>
<extension value="cda"   ftype="audio"/>
<extension value="m4a"   ftype="audio"/>
<extension value="m4b"   ftype="audio"/>
<extension value="mp3"   ftype="audio"/>
<extension value="wav"   ftype="audio"/>
<extension value="wma"   ftype="audio"/>
<extension value="ogg"   ftype="audio"/>
<!-- Photo types -->
<extension value="bmp"   ftype="photo"/>
<extension value="gif"   ftype="photo"/>
<extension value="jpeg"  ftype="photo"/>
<extension value="jpg"   ftype="photo"/>
<extension value="png"   ftype="photo"/>

<!-- Video types -->
<extension value="m4v"   ftype="video"/>
<extension value="mp4"   ftype="video"/>
<extension value="mpeg4" ftype="video"/>
<extension value="mpg"   ftype="video"/>
</library>
```

For a current list of extensions, see the configuration file **`mme.conf`**.

Playlist extensions do not require attributes. For example:

```
<playlists>
<extension value="m3u"/>
<extension value="pls"/>
</playlists>
```

Configurable file skipping

The **`<SyncFileMask>`** element allows you to configure MME synchronization to ignore files based on a specific character string in the file name (including the file extension). This configuration is set in the MME configuration file **`mme.conf`**.

- The default configuration is to *not* skip any files based on the file name.
- There can be only *one* instance of **`<SyncFileMask>`** in the configuration file.

Syntax

Use the regular expressions described in the table below to create the mask defining the character string or strings identifying files to be ignored by the MME synchronization:

Character	Meaning
`\`	Treat the next character as a literal.
`^`	Begins with.
`$`	Ends with

The **`<SyncFileMask>`** element supports POSIX regular expressions, so you can use AND (&) and OR (**|**) operators to create your mask.

Examples

Below are some simple examples of masks to ignore files with names that:

Begin with "**.**"	**`<SyncFileMask>`**^\./**`<SyncFileMask>`**. The "**.**", which usually means any character, here means the "**.**" (dot) character, because it is preceded by a "\".
Contain "**w**"	**`<SyncFileMask>`**w/**`<SyncFileMask>`**
End with "roy"	**`<SyncFileMask>`**roy$/**`<SyncFileMask>`**

Begin with "**.**" or end with ".mp3".

`<SyncFileMask>`(^\.)**|**(\.mp3$)/**`<SyncFileMask>`**

If you create a complex mask with multiple operators, you should make sure that you don't configure the MME to ignore files that you want synchronized.

Configurable handling of updated files and folders

The **`<ChangedFilesHaveConstantId>`** element determines how, during a file and folder discovery synchronization pass, the MME handles a media or playlist file whose size or date has changed since the previous file and folder discovery pass.

Default behavior

The default MME behavior is to consider a file or folder whose size or date has changed as new, and to synchronize it accordingly, assigning it a new file ID (*fid*).

Behavior with `<ChangedFilesHaveConstantId>` enabled

If the **`<ChangedFilesHaveConstantId>`** *enabled* attribute is set to **`true`**, the MME's file and folder discovery synchronization pass does not consider files and folders whose dates or sizes have changed to be new files or folders, and:

- *maintains* (does not change) their file IDs
- sets their *accurate* fields to 0 (zero) to indicate that the file or folder metadata may have changed; this field is in the the **`library`** table for media files and in the **`playlist`** table for playlist files

Thus, if you configure the MME to maintain the file ID of a file or folder whose sizes or date has changed, to determine if file metadata is accurate, you must check its *accurate* field. If the field is 0, then the file requires a metadata synchronization pass.

If media files have not been successfully synchronized during the metadata pass, or playlist files have not been synchronized during the playlist synchronization pass, the *accurate* field for these files will be 0, regardless of the **`<ChangedFilesHaveConstantId>`** configuration.

Playlist synchronization behavior

Playlist synchronization differs from media file synchronization:

- The playlist synchronization pass synchronizes *all* playlists, including playlists marked accurate.
- A playlist is marked accurate even if not all entries in the playlist can be found in the **`library`** table.
- If a playlist is marked "accurate", the playlist synchronization operation validates the playlist entries against files in the **`library`** table, checking if any files have been added or removed.
- If a file corresponding to a playlist entry is marked as not accurate, the MME assumes that the playlist has changed.

Setting limits on synchronizations

This section describes configuration options for limiting the folders and items the MME synchronizes. It includes:

- Limiting the entries in the **`library`** table
- Limiting the number of items synchronized in a folder

Limiting the entries in the `library` table

You can configure the maximum number of entries you allow at one time in the MME **`library`** table, for any given slot.

To limit the number of entries permitted in the MME's **`library`** table, update the value in the **`slots`** table column *max_lib_entries* with the maximum number of entries permitted for the mediastore type associated with a slot. For example, to set the maximum for the mediastore "Hardrive" to 4000 entries:

```
INSERT INTO slots(path,zoneid,name,slottype,max_lib_entries)
    VALUES('/media/drive', 1, 'HardDrive', 3, 4000);
```

- A limit on entries can only be set for BFS (Block File System) mediastores (HHD, USB, SD cards, data CDs, DVDs and PFS devices).
- If the *max_lib_entries* column is 0, there is no limit to the number of entries permitted for the mediastore in the slot.
- The MME does not include device file IDs when it counts the number of entries in the **`library`** table.
- If the MME is configured to add now playing entries to the database (**`<UpdateLibraryFromNowplaying enabled="true"/>`**) and an external CD changer plays tracks that are not in the database, the MME will add entries for these track without checking the limit, and the number of entries in the **`library`** table may exceed the set limit. To prevent this situation from occurring, set the limit to a value higher than the maximum number of tracks supported by the mediastore.

See also the events associated with maximum entries in the **`library`** in the *MME API Library Reference*:

- MME_SYNC_ERROR_LIB_LIMIT
- MME_EVENT_MS_UPDATE

Limiting the number of items synchronized in a folder

Excessively large numbers of files in folders on a mediastore can make using that mediastore very slow, and cause poor response to synchronization operations, such as changing of priority folders or determination of changes to the mediastore.

The MME includes the **`<Database>`**/**`<Synchronization>`** configuration element: **`<MaxFolderItems>`**. This element limits the number of items the MME will synchronize in any one folder.

For information about limiting folder synchronization on iPods, see “Limiting the number of iPod folders synchronized” below.

MME behavior when `<MaxFolderItems>` is implemented

Note the following about MME behavior when **`<MaxFolderItems>`** is implemented:

- Currently, the block-file system (BFS) mediastore synchronizer is the only MME sychronizer to use the value in the **`<MaxFolderItems>`** element. Other synchronizers ignore it.
- The default setting for **`<MaxFolderItems>`** is 0: there is no limit on the number of items synchronized in a folder. Synchronization behavior remains as in previous releases.

- In order to keep the count consistent, independently of the type of items in the folder, and of their contents, the MME calculates the number of items in a folder before performing any configured filtering.
- Since "." and ".." are always filtered, the MME compensates for these items when it calculates the number of items in a folder.
- When an MME synchronization operation reaches the limit set by **`<MaxFolderItems>`**, it delivers the event MME_SYNC_ERROR_FOLDER_LIMIT.

When the MME is configured to limit the number of items it synchronizes in any one folder (**`<MaxFolderItems>`** is set to non-zero), the filesystem *readdir()* determines the items that the MME sees. The MME does not see any items in a folder that are beyond the limit specified by **`<MaxFolderItems>`**, and does not consider them when performing folder change checks. These unseen items may include media files, playlist files, folders, and files that have been filtered out or that, by definition, the MME does not handle (i.e. **`.xls`** files).

Adding or deleting items from a folder on a mediastore affects the presentation of the items to the MME, depending on the operation of the filesystem *readdir()* function, and may cause unexpected changes in the MME database.

Configurable handling of unknown track titles

You can configure how the MME handles entries for unknown track titles in the MME database, by setting the attribute for the **`<Synchronization>`** element **`<UnknownTitlesNull>`**.

Depending on the attribute setting for this element, the MME will either:

- place NULL for the *title* field of the **`library`** table to indicate that the track title is not known

  or:
- create a name for the track, and place this name in the *title* of the **`library`** table.

The default setting for **`<UnknownTitlesNull>`** is "enabled" — use NULL and do not attempt to build titles for tracks with unknown titles. If **`<UnknownTitlesNull>`** is set to **`disabled`**, the MME will build names for tracks, based on the type of mediastore on which the track is found, as follows:

- Block filesystems — the filename, with the file extension stripped off.
- CDDA — *track* number, where *track* is language-dependent.
- DVD-audio — *group* number *track* number, where *group* and *track* are language-dependent.
- DVD-video — *title* number *chapter* number, where *title* and *chapter* are language-dependent.

- iPods — the corrected filename.
- VCD/SVCD — *track* number, where *track* is language-dependent. Note, however, that VCD/SVCD already use identifiers, so this value is never null.

This feature uses language strings in **`mme_data.sql`** to fill in information in the following fields in the **`languages`** table:

- *unknown_track* — string for building unknown title of CDDA and DVD-audio tracks.
- *unknown_chapter* — string for building unknown title of DVD-video tracks.
- *unknown_title* — string for building unknown title of DVD-video tracks.
- *unknown_group* — string for building unknown title of DVD-audio tracks.

iPod synchronization

This section describes configuration options specific to iPod synchronizations. It includes:

- The **`<ipod>`** elements
- Limiting the number of iPod folders synchronized
- Enabling automatic and complete iPod synchronizations
- Displaying the **`.FS_info`** folder contents during iPod exploration

iPods devices may hold a very large number of media files. The preferred method for working with iPod devices is to *not* synchronize them, but instead to use the MME's explore API functions to explore specific folders as requested by the end user.

For more information, see the chapter Working with iPods in the *MME Developer's Guide*.

The `<ipod>` elements

The **`<MSS>`**/**`<ipod>`** element is a container element for elements specifying behaviors for iPod synchronizations. It contains the elements listed in the table below:

Element	Attributes	Default	Description
`<synced_folders>`	*limit*	32	The maximum number of synchronized iPod folders permitted in the MME library. Set the *limit* attribute to 0 (zero) for no limits. See "Limiting the number of iPod folders synchronized" below.
`<auto_sync>`	*permitted*	**`false`**	Override the MME's default iPod synchronization behavior, and automatically synchronize iPod devices on discovery. See "Enabling automatic and complete iPod synchronizations" below.
`<full_sync>`	*everything*	**`false`**	Override the MME's default iPod synchronization behavior, and synchroniz *all* paths in order to avoid redundant work. See "Enabling automatic and complete iPod synchronizations" below.

Limiting the number of iPod folders synchronized

In order to better manage its database, the MME provides a control for limiting the number of folders with files added to its database during the synchronization of iPod devices. This limit is set by the .../**`<Synchronization>`**/**`<MSS>`**/**`<ipod>`**/**`<synced_folders>`** element. The default is 32 folders. A value of 0 means no folder limit.

When it synchronizes an iPod device, the MME synchronizes all the requested folders on the iPod. There is no limit (other than available space for the MME database) to the number of folders or files synchronized. However, on completion of the synchronization, the MME checks for old folders with files from previous synchronizations of the newly synchronized iPod device and, if necessary, it deletes folders from old synchronizations to bring the total number of folders synchronized to be equal to or as close as possible to the value specified in **`<synced_folders>`**.

For example, assuming the value of 32 for **`<synced_folders>`**:

- The MME has previously synchronized the 25 folders with their files inside the iPod folder **`classical`**.
- The end user requests a synchronization of all folders inside the iPod folder **`elvis`**, which contains 17 folders with files inside them.
- When the MME completes synchronization of **`elvis`** and its contents, it has a total of 42 folders with files inside them (25 from **`classical`** plus 17 from **`elvis`**).
- In order to reduce the total number of synchronized iPod folders to 32, the MME deletes from its database information for the first 10 folders inside the previously synchronized iPod folder **`classical`**.

Similary, assuming the default value of 32 for **`<synced_folders>`**, if a previous synchronization of an iPod yielded 40 folders, and the new synchronization of the same iPod (or, what is more likely, of a portion of the iPod) yields 36 folders, the MME will delete the 40 folders from the previous synchronization, leaving the information and metadata for the 36 new folders in the MME database.

Enabling automatic and complete iPod synchronizations

The MME release implements two new configuration elements that allow you to change the MME's default behavior when it synchronizes iPods. Both these elements are contained in the **`<MSS>`**/**`<ipod>`** element:

- **`<auto_sync permitted="false"/>`** — when automatic synchronization is globally enabled (**`<Automatic enabled="true"/>`**), the **`<auto_sync>`** element set to **`true`** configures the MME to automatically synchronize iPods. The default setting is **`false`**: do not synchronize iPod devices until requested to do so.
- **`<full_sync everything="false"/>`** — when this element is set to **`true`**, the MME automatically synchronizes the entire contents of iPod devices. The default setting is **`false`**: do not synchronize the entire contents of iPod devices.

- The MME's explorer API is the preferred method for discovering and accessing files on iPods. If you synchronize an iPod device, it is best to use directed synchronization so that you synchronize only the directories requested by the user.
- Setting **`<auto_sync permitted="true"/>`** and **`<full_sync everything="true"/>`** to have the MME automatically synchronize the complete contents of iPod devices can be useful to demonstrate iPod integration in the MME, but *this configuration is not recommended for production environments*.

For more information about synchronizing iPods, see:

- "Limiting the number of iPod folders synchronized" above
- "Synchronizing iPods" in the *MME Developer's Guide* chapter Working with iPods

Displaying the `.FS_info` folder contents during iPod exploration

The **`<Configuration>`**/**`<Explorer>`** element sets behavior characteristics for the MME's Explorer API. It contains the **`<MSX>`** element, which in turn contains the **`<ipod>`** element.

The **`<ipod>`** element has one attribute, *show_all*, which determines whether the **`.FS_info`** folder and its contents are visible during exploration of an iPod. The default is **`false`**: do not show the **`.FS_info`** folder and its contents.

Plugin support

Like the MME, plugins can take their configuration values from the MME configuration file, **`mme.conf`**, or from built-in defaults. Plugins should provide their built-in defaults so they can be used if the MME configuration file is not specified or available.

The MME configuration file includes a reserved media data (MDP) plugin configuration area under the **`<Synchronization>`**/**`<MDS>`**/**`<plugin name>`**. In this part of the configuration file, you may create your own structure for assigning your initial configuration, using the **`<dvdifo>`** elements as an example.

DVD IFO parser support

The MME has configuration elements to allow control over DVD IFO parsing and for adding DVD device types to the library. These elements are the same for DVD-audio and DVD-video mediastores, and are found under the elements **`<Configuration>`**/**`<Database>`**/**`<Synchronization>`**/**`<MDP>`**/**`<dvdifo>`**/**`<dvdaudio | dvdvideo>`**. They are described in the table below.

Element	Attributes	Default	Description
`<devicefid>`	*create*	**`true`**	Determines if a device file ID for the entire mediastore is added to the library during synchronization.
`<chapterfids>`	*create*	**`true`**	Enables or disables the IFO parser. If the IFO parser is enabled, synchronization will find individual tracks on the DVD.

Item selection based on time of entry

The MME includes a configuration option that you can set to prevent the synchronization metadata pass (pass 2) from updating the *last_sync* column in the **`library`** table. This capability allows you to select entries from the **`library`** table based on the time the entries were first entered in the database by the synchronization file and folder discovery pass (pass 1), instead of based on when the metadata for these entries was updated.

The default for the **`<PassTwoSetsLastSync>`** element is **`true`**: update the *last_sync* column during the synchronization metadata pass.

Synchronization constants

The table below lists the default synchronization management constants.

Constant	Value	Description
CONFIG_DEF_SYNC_AUTOMATIC	**`true`**	Automatically synchronize mediastores on detection.

continued...

Constant	Value	Description
CONFIG_DEF_THREAD_PRIORITY	0	The priority for synchronization threads. Set to 0 to use **mme** startup values. See “Configuring synchronization thread priorities” above.
CONFIG_DEV_DVDA_FID_CREATION	**true**	Add a device file ID for the entire mediastore to the library during synchronization.
CONFIG_DEV_DVDA_IFO_PARSING	**true**	Enable or disable the IFO parser. If the IFO parser is enabled, synchronization will find individual tracks on a DVD-audio.
CONFIG_DEV_DVDV_FID_CREATION	**true**	Add a device file ID for the entire mediastore to the library during synchronization.
CONFIG_DEV_DVDV_IFO_PARSING	**true**	Enable or disable the IFO parser. If the IFO parser is enabled, synchronization will find individual tracks on a DVD-video.
CONFIG_DEF_DVD_SKIP_DURATION_ZERO	**true**	Don’t synchronize entries (i.e. DVD-video chapters) with a duration of 0 milliseconds.

Chapter 7

Configuring Metadata Support

In this chapter. . .

This chapter explains how to configure the MME metadata support, including image pre-processing.

Metadata in the MME database

This section describes configuration settings that determine some metadata characteristics in the MME database:

- Populating the **library** table metadata during playback
- Limiting the length of metadata strings

For information about configuring synchronization behavior, see the chapter Configuring Media Synchronizations.

Populating the `library` table metadata during playback

The MME can be configured to write to the **library** table the metadata it has for the currently playing track in the **nowplaying** table.

This option is set by the **<Synchronization>** element **<UpdateLibraryFromNowplaying>** in the MME configuration file. The default setting is “off”: do not update the **library** metadata from the **nowplaying** table.

With this option enabled, depending on your environment and your performance requirements, you may configure your system to perfrom only the first synchronization pass, then populate the metadata in the **library** table only as tracks are played.

Limiting the length of metadata strings

The **<MetadataStringMaximums>** element contains **<metadatastring>** elements that you can use to limit the length of the metadata strings that the MME writes in its database.

The default configuration is to *not* limit the length of metadata strings. To change this configuration, remove the comments from around the **<MetadataStringMaximums>** element, and update the attributes for the **<metadatastring>** elements as required.

These attributes are:

- *field* — identify the metadata; corresponds to a field in the **library_*** tables
- *maximum* — sets the maximum number of characters the MME will write to the database for the *field*

Below is a sample **<MetadataStringMaximums>** element:

```
<MetadataStringMaximums>
<metadatastring field="artist" maximum="1024"/>
<metadatastring field="album" maximum="1024"/>
<metadatastring field="genre" maximum="1024"/>
<metadatastring field="composer" maximum="1024"/>
<metadatastring field="category" maximum="1024"/>
```

```
<metadatastring field="title" maximum="1024"/>
<metadatastring field="conductor" maximum="1024"/>
<metadatastring field="soloist" maximum="1024"/>
<metadatastring field="ensemble" maximum="1024"/>
<metadatastring field="opus" maximum="1024"/>
<metadatastring field="description" maximum="1024"/>
</MetadataStringMaximums>
```

Artwork support

The MME's metadata interface is used to retrieve artwork associated with media files. This section describes MME metadata interface configuration options. Metadata interface configuration elements are under the **`<MetadataInterface>`** top-level container element in the MME configuration file **`mme.conf`**:

- **`<MetadataInterface>`** elements
- Image extraction
- External artwork

For information about how to use the metadata interface to retrieve artwork, see the chapter Metadata and Artwork in the *MME Developer's Guide*.

`<MetadataInterface>` elements

The table below lists the top-level container elements inside the in **`<MetadataInterface>`**:

Element	Default	Description
`<ExternalArtwork>`	n/a	Container for elements defining external artwork search behavior. See "External artwork" below.
`<ImageExtraction>`	n/a	Container element for image extraction configuration elements. See "Image extraction" below.
`<ImageProcessing>`	n/a	Container element for elements used to configure custom processing modules. See "Image pre-processing" below.

Example

Below is an example of a sample metadata interface configuration:

```
<MetadataInterface>
<ImageExtraction>
        <TemporaryLocation path="/tmp"/>
<PersistentLocation enabled="true" path="/imagecache" maxsize="0" prune_algorithm="0"/>
</ImageExtraction>
</MetadataInterface>
```

Image extraction

The elements inside the **`<ImageExtraction>`** element configure locations and paths to be used by the MME's image extraction API. These elements are listed in the table below:

Element	Attributes	Default	Description
`<ImageExtension>`	*value*		User-defined extension to append to the generated image URL that is returned when loading images. The default is to append no extension to the image URL.
`<PersistentLocation>`	See below.	**`"/imagecache"`**	The permanent location for extracted images; see "**`<PersistentLocation>`** attributes" below.
`<TemporaryLocation>`	*path*	**`/tmp/`**	The path to a temporary location for storing images. This location needs to persist only for the duration of the MME life span. The MME must have read/write access to this location.

`<PersistentLocation>` attributes

The **`<PersistentLocation>`** element supports the following attributes:

- *path* — the path to a "permanent" location for storing images. This location must persist when the MME powers down and restarts. The MME must have read/write access to this location.
- *enabled* — boolean enabling or disabling of the persistent image cache.
- *maxsize* — the maximum number of image bytes to cache before pruning older cache entries from the cache.
- *prune_algorithm* — the algorithm to use when pruning files from the persistent image cache. The algorithms are as follows:
  - 0 (Oldest Hit Files) — (default) remove cached entries with the oldest access times first.
  - 1 (Least Hit Files) — remove the least frequently accessed cached entries first.
  - 2 (Least Hit/FIFO) — remove the oldest and least hit cached entries first.

Support for the **`<PersistentLocation>`** element will be implemented in a future release.

External artwork

The MME includes configuration elements that define if and how the MME's metadata API functions search for external artwork (images not embedded in a media file, but placed in folders adjacent to the media file). The table below lists these elements:

Element	Attributes	Default	Description
`<ArtworkFileNames>`			Container element for **`<FileSpecifier>`** elements that define regular expressions used for external artwork searches. See "**`<FileSpecifier>`** elements" below.
`<SearchBehaviour>`	See below.	**`never`**	Configure behavior for external artwork searches. See "**`<SearchBehaviour>`** attributes" below.

`<SearchBehaviour>` attributes

The **`<SearchBehaviour>`** element defines when and how the MME's metadata API searches for external artwork, if it has been requested to load artwork for a media file. This element has three attributes:

- *when* — set when to search for external artwork; possible values are:
  - **`never`** — default: never search for external images; other **`<SearchBehaviour>`** attributes are irrelevant when the *when* attribute is set to **`never`**
  - **`always`** — always search for external images
  - **`noimages`** — search for external artwork *only* if no images are embedded in the media file
- *ignore_case* — make the search matching rule case sensitive; set to either **`true`** or **`false`**; default is **`true`**: not case sensitive
- *search_limit* — the maximum number of files in a folder to examine before abandoning the external artwork search; default is the value set by CONFIG_DEF_MDI_EXT_ARTWORK_SEARCH_LIMIT

For example:

```
<ExternalArtwork>
    <SearchBehaviour when="always" ignore_case="true" search_limit="100"/>
...
</ExternalArtwork>
```

<FileSpecifier> elements

The **<FileSpecifier>** elements defines regular expressions that the MME will match against when searching for external artwork files. You can define multiple **<FileSpecifier>** elements, up to the maximum set by CONFIG_MAX_ARTWORK_FILE_SPECIFIERS. A **<FileSpecifier>** element has one attribute, *value*, which specifies a foldername, and a file extension:

```
<ExternalArtwork>
    ...
    <ArtworkFileNames>
        <FileSpecifier value="folder\.jp[e]?g"/>
        <FileSpecifier value="album\.jp[e]?g"/>
...
    </ArtworkFileNames>
</ExternalArtwork>
```

Note that optional letters can be used in the file extension string. For example, the configuration below instructs the MME to search the **folder** and **album** directories for files with extensions of either **.jpg** or **.jpeg**. If the **<SearchBehaviour>** element's *ignore_case* attribute is set to **true** the MME will also search for files with **.JPG** or **.JPEG** extensions.

CONFIG_MAX_ARTWORK_* constants

The following constants specify limits used when searching for external artwork:

Name	Value
CONFIG_MAX_ARTWORK_FILE_SPECIFIERS	32
CONFIG_MAX_ARTWORK_FILE_SPECIFIER_LENGTH	64
CONFIG_MAX_ARTWORK_FILE_EXTENSION_LENGTH	8
CONFIG_DEF_MDI_EXT_ARTWORK_SEARCH_LIMIT	100

Image pre-processing

The MME's image processing capabilities are configured through the **<MetadataInterface>**/**<ImageProcessing>** element.

To enable image pre-processing, you must:

1. Configure the image processing library.
2. Enable one or more image processing modules.
3. Configure image processing profiles.

For information about image pre-processing, see the chapter Metadata and Artwork in the *MME Developer's Guide*.

Configuring the image processing library

To enable image processing you must configure the image processing library by having the following configuration file **/etc/system/config/img.conf** present on the system and configured for the required codecs, as shown below:

```
[img_codec_bmp.so]
mime=image/bmp:image/x-bmp:image/x-bitmap:image/x-xbitmap:image \
    /x-win-bitmap:image/x-windows-bmp:image/ms-bmp:image \
    /x-ms-bmp:application/bmp:application/x-bmp:application \
    /x-win-bitmap:application/preview
ext=bmp
[img_codec_gif.so]
mime=image/gif:/image/x-xbitmap:image/gi_
ext=gif
[img_codec_jpg.so]
mime=image/jpeg:image/jpg:image/jp_:application \
    /jpg:application/x-jpg:image/pjpeg:image \
    /pipeg:image/vnd.swiftview-jpeg:image/x-xbitmap
ext=jpg:jpeg
[img_codec_png.so]
mime=image/png:application/png:application/x-png
ext=png
[img_codec_sgi.so]
mime=image/sgi:image/x-sgi:image/x-sgi-rgba:image \
    /x-sgi-bw:image/rgb:image/x-rgb
ext=sgi:rgb:rgba:bw
[img_codec_tga.so]
mime=application/tga:application/x-tga:application \
    /x-targa:image/tga:image/x-tga:image/targa:image/x-targa
ext=tga
[img_codec_pcx.so]
mime=application/pcx:application/x-pcx:image \
    /pcx:image/x-pcx:image \
    /x-pc-paintbrush:zz-application/zz-winassoc-pcx
ext=pcx
```

For more information about the **img.conf** configuration file, see *img_lib_attach()* in the *Advanced Graphics Developer's Guide*.

Enabling an image processing module

The MME includes the image processing module **mme-imgprc-gf.so**. However, by default, no image processing modules are loaded into the MME.

The **<ConversionThreadOptions>** and **<Libraries>** elements, are located inside the **<ImageProcessing>** element in the MME configuration file.

The table below describes these elements:

Element	Attributes	Default	Description
`<ConversionThreadOptions>`	*priority*	0	Set attributes associated to the image conversion thread run when a image loading occurs, with a profile. The *priority* is an absolute priority. The default is 0: run at the same priority as the MME.
`<Libraries>`			Container for one or more **`<dll>`** elements. See "**`<dll>`** element" below.

`<dll>` element

The table below describes the **`<dll>`** element:

Element	Attributes	Default	Description
`<dll>`	*name*	empty	Set the name of the image processing module. One or more of these elements must be set to enable the MME's image processing capabilties.

To enable the MME's image processing capabilties, you must remove the comments around the **`<Libraries>`** and **`<dll>`** configuration elements, then restart the MME. You can add new **`<dll>`** elements for new image processing modules, as required.

Example

The example below shows a portion of the MME configuration file enabling the **`mme-imgprc-gf.so`** image processing module:

```
<ImageProcessing>
...
<Libraries>
<dll name="mme-imgprc-gf.so"/>
</Libraries>
...
</ImageProcessing>
```

CAUTION: If you do not enable the image processing module, or if there are no image processing modules configured, attempts to use a defined image processing profile through the MME's metadata interface will fail with an EINVAL error.

Configuring image processing profiles

Image processing profiles determine how an image is transformed and transcoded when it is loaded through the MME's metadata API. Image profiles are defined by multiple **`<Profile>`** elements and their sub-elements inside the **`<ImageProcessing>`** element.

The table below describes the **`<Profile>`** element:

Element	Attributes	Default	Description
`<Profile>`	*index*		Multiple profile elements specify how an image should be processed when uploaded through the MME's metadata API. The *index* attribute is a *required* profile number, to which you set the *mme_metadata_image_load()* function's *image_format_profile* parameter so the the MME processes the image according to this profile.

The table below describes the **`<Profile>`** element's sub-elements that determine image processing specifics:

Element	Attributes	Default	Description
`<Transform>`			Container for one or more possible sub-elements, such as **`<Scale>`** and **`<Rotate>`** that define how to transform an image being loaded.
`<Transcode>`			Container for transcoding elements, such as **`<Output>`**, which determines th output format for an image.

Values passed to element attributes should be enclosed in double quotation marks. For example:

```
<SomeElement some_attribute="some_value">
```

The `<Transform>` element and its sub-elements

The **`<Transform>`** element contains sub-elements that define how to transform an image that is being loaded with the current profile. All sub-elements are processed in the order in which they are defined. Currently, supported elements are:

- **`<Scale>`**
- **`<Rotate>`**

`<Scale>`

The **`<Scale>`** element supports the following attributes:

- *width* — the width of the image, in pixels (default) or as a percentage
- *height* — the height of the image, in pixels (default) or as a percentage
- *percentage* — determine how the *width* and *height* attributes are interpreted. If *percentage* is set to **`true`**, the values for *width* and *height* are interpreted as percentages of the original image width and height. If *percentage* is set to **`false`** or not specified, the values for *width* and *height* are interpreted as a number of pixels.

If only one of the *width* and *height* attributes is specified, the image is scaled to the specified size, maintaining its original aspect ratio.

`<Rotate>`

The **`<Rotate>`** element supports the following attribute:

- *method* — specify how to rotate the image. Supported rotations are:
  - 90cw — rotate 90 degrees clockwise
  - 90ccw — rotate 90 degrees counter clockwise
  - 180 — rotate 180 degrees

The `<Transcode>` element and its sub-elements

The **`<Transcode>`** element contains sub-elements that define how to transcode an image that is being loaded with the current profile. All sub-elements are processed in the order in which they are defined. Currently, the only supported element is:

- **`<Output>`**

If the **`<Transcode>`** element is:

- *not* specified in a profile, the image will be encoded back into its original format
- specified in a profile, it must be the *last* element listed inside the **`<Profile>`** element

`<Output>`

The **`<Output>`** element supports the following attributes:

- *encoding* — the output image coding. Currently supported output codings are:
  - jpeg — JPEG image format
  - bmp — BMP image format
  - gif — GIF image format
  - png — PNG image format
  - tiff — TIFF image format
  - ppm — PPM image format
- *format* — the output image format. Currently supported formats are:
  - mono — monochromatic bitmap with 1 bit per pixel, packing 8 pixels per byte
  - grey — 8-bits per pixel graymap
  - rgb24 — 24-bits per pixel RGB, with 8 bits per channel as an ordered byte sequence
- *quality* — the image quality, for codecs that support compression. Valuse can be from 0-100, with 100 as the highest quality

Example

Below is an example of an image processing element and its sub-elements:

```
<ImageProcessing>
<Libraries>
<dll name="mme-imgprc-gf.so"/>
...
</Libraries>

<Profile index="0">
    <Transform>
        <Scale width="500"/>
    </Transform>
</Profile>
<Profile index="1">
    <Transform>
        <Rotate method="90cw"/>
        <Scale width="150" height="100"/>
    </Transform>
    <Transcode>
        <Output format="jpeg"/>
    </Transcode>
</Profile>
...
</ImageProcessing>
```

Metadata synchronizers

The MME includes support for CDText, Gracenote and Musicbrainz metadata synchronizer plugins:

- Metadata synchronizer ratings
- Gracenote
- MusicBrainz

Metadata synchronizer ratings

The container elements for these plugins are used to enable or disable the plugins, and to set their ratings:

Element	Attributes	Default	Description
`<cdtext>`	*enabled*	**`true`**	Enable and set the rating for the CDText metadata synchronizer.
	rating	80	
`<gracenote>`	*enabled*	**`true`**	Enable and set the rating for the Gracenote metadata synchronizer.
	rating	71	
`<musicbrainz>`	*enabled*	**`true`**	Enable and set the rating for the MusicBrainz metadata synchronizer that performs internet HTTP queries to the Musicbrainz web service.
	rating	71	
`<musicbrainz_embedded>`	*enabled*	**`true`**	Enable and set the rating for the MusicBrainz metadata synchronizer that uses a QDB database.
	rating	70	

Using synchronizer ratings

The metadata synchronizer container elements have attributes to enable or disable the relevant synchronizers, and to set their ratings. If more than one synchronizer is, the MME attempts to obtain metadata through the synchronizer with the highest rating. If a synchronizer is not specifically disabled by its configuration element's *enabled* attribute, it is considered enabled. The example below shows the CDText synchronizer enabled, and the Gracenote synchronizer disabled:

```
<Synchronization>
    ...
    <MDP>
    ...
        <cdtext rating="80"/>
        <gracenote enabled="false" rating="70">
     </MDP>
</Synchronization>
```

Gracenote support

Configuration elements for the Gracenote plugin are under the element **`<MDP>`**, in element **`<gracenote>`**. As well as containing other elements for configuring Gracenote plugins, this element has two attributes that enable or disable the plugin, and rate the plugin if it is enabled.

The elements contained under the **`<gracenote>`** element are listed in the table below:

Element	Attributes	Default	Description
`<keyfile>`		empty	Path to Gracenote key file.
`<databasepath>`		empty	Path to the Gracenote database.
`<classical>`	*enabled*	**`true`**	See "Gracenote classical genres" below.

Gracenote implementation requires special licensing. Contact QNX for more information.

Configurable rating for Gracenote server plugin

The MME includes support for a rating for the Gracenote client/server plugin. This feature uses the **`<MDP>`**/**`<gracenote_srvr>`** element and its sub-element **`<mountpath>`** to set a rating for the Gracenote metadata synchronizer plugin.

The table below describes the **`<gracenote_srvr>`** element:

The table below describes the **`<gracenote_srvr>`** sub-element **`<mountpath>`**:

Element	Attributes	Default	Description
`<mountpath>`		empty	Path to the Gracenote server; defining this path enables the rating.

To set the rating for the Gracenote server plugin, simply adjust the *rating* value amd provide the path to the server. For example:

```
<MDP>
    <gracenote_srvr rating="88">
        <mountpath>/dev/gracenote</mountpath>
    </gracenote_srvr>
    ...
</MDP>
```

Gracenote classical genres

Setting the **`<classical>`** element's *enabled* attribute to **`true`** instructs the MME to extract the Gracenote metadata specified by the **`<genrekey>`** elements' *genre* attributes.

The code sample below shows the explicitly configured genres in the MME configuration file:

```
<classical enabled="true">
    <genrekey genre="Baroque"/>>
    <genrekey genre="Chamber Music"/>>
    <genrekey genre="Choral"/>>
    <genrekey genre="Classical Guitar"/>>
    <genrekey genre="Contemporary"/>>
    <genrekey genre="Ensembles"/>>
    <genrekey genre="General Classical"/>>
    <genrekey genre="Japanese Classical"/>>
    <genrekey genre="Medieval"/>>
    <genrekey genre="Romantic Era"/>>
    <genrekey genre="Piano"/>>
    <genrekey genre="Renaissance Era"/>>
    <genrekey genre="Strings"/>>
</classical>
```

MusicBrainz support

The MME supports using the MusicBrainz metadata synchronizer to retrieve metadata from a QDB database, and from the MusicBrainz web service.

Configuration elements for the MusicBrainz metadata synchronizersplugin are under the element **`<MDP>`**. These elements enable or disable the two types of metadata synchronization, and rate them plugin if they are enabled. These elements are described with the other metadata sychronizer configuration elements under the "Metadata synchronizers" heading above.

Path to Musicbrainz database

The **`<musicbrainz_embedded>`** element includes the **`<databasepath>`** element, which defines the path to the QDB database with the MusicBrainz metadata. The default path defined by this element is **`/dev/qdb/musicbrainz`**

Music Brainz implementation requires special licensing. Contact QNX for more information.

Chapter 8

Configuring Playback

In this chapter. . .

This chapter explains how to configure the MME's playback behavior. It includes:

- Playback behavior configuration elements
- Configuring playback behavior
- Digital Rights Management (DRM)

Playback configuration elements

All playback configuration elements are under the **`<Playback>`** element. The table below lists these elements. For simplicity, the **`<Playback>`** element is included in the table.

Element	Attributes	Default	Description
`<Playback>`	*played_count*	"end"	Configure how the MME increments the number of times a track has been played. See "Configuring how plays are counted" below.
`<AtEndOfSeek>`		**`play`**	Set MME behavior at end of a seek operation. This element can be set to either "play": play forward, or "seek": continue seeking. Note that devices, such as iPods, that manage their own track sessions may not honour this setting.
`<ConsecutivePlayErrorsBeforeStop>`		5	Set the number of consecutive times the MME will attempt to initiate playback of different tracks in a track session count before it reports a tracksession failure. See "Configuring track session failure settings" below.

continued. . .

Element	Attributes	Default	Description
`<DeviceRemovalSavesState>`		**`true`**	If this option is enabled, the MME automatically saves the track session state when a mediastore is removed during playback. With this option enabled, there is no need to call *mme_trksession_save_state()* after receiving an MME_PLAY_ERROR_DEVICEREMOVED event.
`<SkipUnplayable>`	*enabled*	**`false`**	If this option is enable, the MME automatically skips files marked as unplayable without sending an error message. See "Automatically skip files marked as unplayable" below.
`<TrksessionViewAutoWrite>`		**`true`**	Determines if the MME automatically writes track session views to the database, or if keeps them in memory and writes them out only when specifically requested to do so by a call to *mme_trksessionview_writedb()*.
`<TrksessionViewDelayAfterWrite>`		0	The time, in milliseconds, to delay after finishing writing to the **`trksessionview`** table.
`<TrksessionViewWriteSize>`		1000	Set the number rows that the MME writes or deletes from the **`trksessionview`** table at at time. See "Setting the number of tracks written to the **`trksessionview`** table" below.

Playback constants

The table below lists the default playback management constants.

Configuring playback behavior

This section describes how to use the playback configuration elements and their attributes to set playback behavior.

Setting the number of tracks written to the `trksessionview` table

The element **`<Playback>`**/**`<TrksessionViewWriteSize>`** sets the number of rows (IDs of tracks in the track session) that the MME writes or deletes from the **`trksessionview`** table at at time. The more rows the MME writes at a time, the better the MME performance for track changes during playback, but the longer the **`trksessionview`** table will be locked when it is updated.

Configuring track session failure settings

The element **`<Playback>`**/**`<ConsecutivePlayErrorsBeforeStop>`** sets the number of consecutive times the MME will attempt to initiate playback of different tracks in a track session count before it delivers the event MME_EVENT_FINISHED_WITH_ERROR to report a playback failure requiring action by the client application.

Note that if the number of tracks in a track session or, if in repeat mode, the number of tracks to repeat is less that the value of **`<ConsecutivePlayErrorsBeforeStop>`**, the MME will count to the lesser value before reporting the track session failure. For example, if repeat mode is set to MME_REPEAT_SINGLE and **`<ConsecutivePlayErrorsBeforeStop>`** is set to 5, the MME will deliver the MME_EVENT_FINISHED_WITH_ERROR event after it has failed to play the one track it has been asked to repeat. It will not make five attempts to initiate playback of the same track.

Incrementing the error count

The MME increments the consecutive play error count if:

- playback cannot be started on a track
- an error prevents playback of a track to completion

Resetting the error count

The MME resets the consecutive play error count if a client:

- starts playback of a tracksession at any given offset
- resumes playback of a tracksession
- stops playback
- forces playback to the next or the previous track

Configuring seek mode

You can configure how the MME behaves when *mme_play_set_speed()* reaches the end of a track. If **`<AtEndOfSeek>`** is set to **`seek`**, the seek operation is continued through to the next or previous track in the track session. If there is no next or previous track, seeking stops. If **`<AtEndOfSeek>`** is set to **`play`**, in a forward seek operation, once the end of the current track is reached, the MME begins playing the next track in the track session at normal speed. In a reverse seek operation, when the beginning of the current track is reached, it is played at normal forward speed.

Playback pre-queuing

The MME offers playback pre-queuing. When playback pre-queuing is implemented, the MME indicates to the **`io-media-*`** component playing the current track the next track it should play before it finishes, so that **`io-media`** can reduce to a minimum any gap between tracks.

Support for playback pre-queuing depends on the specific **`io-media-*`** component used to play tracks. The **`io-media`** **`mmf_graphbuilder`** module, for example, uses the **`pre-buffer`** option to set how much of the next graph it builds between the time the currently used graph has finished reading a track and the time it finishes playback. For more information, see "**`mmf_graphbuilder`** options" on the **`io-media`** page of the *MME Utilities Guide*.

Playback pre-queuing is enabled in the MME by default; when it is enabled, **`io-media`** implements playback pre-queuing according to its settings.

If you playback pre-queuing is not supported by the **`io-media-*`** component used to play tracks, the MME does not tell **`io-media`** what track it wants to play after the currently playing track has finished. It waits until **`io-media`** announces that it has completed playback of the current track before requesting playback of a new track.

Playback pre-queuing affects autopause behavior. For more information, see *mme_setautopause()*.

Automatically skip files marked as unplayable

The **`<SkipUnplayable>`** element configures the MME's behavior when it encounters a file marked as unplayable. The default setting for this element is **`false`**: attempt to play all files and handle errors according to the error handling configurations.

To configure the MME to skip files that have been previously marked as unplayable, set the **`<SkipUnplayable>`** element's *enable* attribute to **`true`**. With this configuration, the MME treats unplayable files as though they did not exist. It:

- checks the value of the *playable* field in the **`library`** table
- skips any files with the *playable* field set to 0 (**`library.playable == 0`**)
- does *not* send any events or errors to indicate that it skipped an unplayable file

- Unplayable files are visible in the track session view if they match the select statement used to populate the the track session.
- You can have your client application filter out files marked as unplayable by including the clause **`WHERE playable=1`**, in the SQL SELECT statement you use to build your library-based track sessions.

See also “Marking unplayable files” in the *MME Developer’s Guide* chapter Playback Errors.

Configuring how plays are counted

The playback includes an attribute that allows you to set how the MME increments the number of times a track has been played and records it in the *fullplay_count* column of the **`library`** table.

This attribute, *played_count* can be set to any of the following values:

- “off” — do not increment the count
- “start” — increment the count when playback of the track begins
- “end” — increment the count when playback of the track ends

The default value for *played_count* is “end”.

Configuring Digital Rights Management (DRM)

The MME’s PlaysForSure (PFS) module **`iofs-pfs`** supports Microsoft’s Windows Media DRM 10 for WMA content on PFS devices. If you will use DRM-enabled devices on your system, you should start **`io-fs`** with the **`drm`** option, so that it checks for the required files and exits if it does not find them. This strategy ensures that the MME does not fail when it attempts to playback DRM-protected media. For more information, see **`iofs-pfs.so`** in the *MME Utilities Reference*.

Chapter 9
Configuring Media Copying and Ripping

In this chapter. . .

This chapter explains how to configure the MME's media copy and ripping behavior.It includes:

- Media copy and ripping configuration elements
- Configuring the destination
- Media copy and ripping constants

Media copy and ripping configuration elements

The table below list media copy and ripping configuration elements. All media copy and ripping configuration elements are under the **`<Configuration>`**/**`<Copying>`** element.

Element	Attributes	Default	Description
`<DeleteOnNonRecoverableError>`		**`false`**	If enabled, delete entries in the copy queue for files that cause unrecoverable copying or ripping errors (MME_COPY_ERROR_NORIGHTS or MME_COPY_ERROR_CORRUPTION) and deliver the event MME_EVENT_COPY_FATAL_ERROR.
`<Encoding>`		empty	
`<Encoding>`/**`<ENCODEFORMATID>`**		2	Default encode format for ripped media. The value for this element corresponds to a row in the **`encodeformats`** table. The default setting (2) is for the **`.wav`** format.
`<Mode>`		**`background`**	Set media copy and ripping mode. If set to **`background`**, playback has priority. If set to **`priority`**, media copy and ripping operations have priority.
`<UpdateMetadata>`	*enabled*	**`true`**	If the *accurate* field in the **`library`** table for the file is set to 0 (zero) indicating that the metadata is not accurate, before copying or ripping the file, the MME synchronizes the source file so its metadata is accurate.

Configuring the destination

This section describes the media copying and ripping destination configuration elements:

- **`<Destination>`** elements
- Overwriting files
- Files with insufficient metadata

`<Destination>` elements

The elements in the table below configure destination parameters for media copying and ripping operations. They are under the **`<Copying>`**/**`<Destination>`** element.

Element	**Attributes**	**Default**	**Description**
`<Filename>`			Default name to assign to ripped files: $0TRACK-$TITLE(date=$DATESTAMP, time=$TIMESTAMP, srcfid=$SRCFID).
`<FileOverwrite>`		**`false`**	If set to **`true`**, overwrite destination files during ripping operations.
`<Folder>`			Default destination folder for ripped files: /ripped/$ARTIST/$ALBUM/.
`<IgnoreNonAccurate>`	*enabled*	**`false`**	If set to **`true`**, when file metadata is not accurate, use the default file name defined by the **`<Filename>`** element.
`<MSID>`		1	Default destination mediastore for copied and ripped media.
`<PreservePath>`	*enabled*	**`false`**	If set to **`true`**, preserve folder paths from source mediastore on the destination mediastore. For information about how to override this setting for individual copy operations, see Using the $*PRESERVE_PATH* template strings in the *MME Developer's Guide*.

Overwriting files

The element **`<Copying>`**/**`<Destination>`**/**`<FileOverwrite>`** allows the overwriting of destination file during ripping operations. If the attribute of this element is set to **`true`**, when the MME encounters in the destination directory a copy of a file it is ripping, it will overwrite the file, but still deliver the event MME_COPY_ERROR_FILEEXISTS.

Files with insufficient metadata

If metadata is not available for the template, the MME will insert a default value, such as “Unknown Artist” for $ARTIST from the **`languages`** table.

The **`<IgnoreNonAccurate>`** element allows you to choose a different filename if the metadata is not marked as accurate. If you have valid metadata, you may want to use the destination filename you have provided, but if the metadata is not valid, you may wish to choose a different filename to guarantee its uniqueness.

For example, if you choose the destination filename $0TRACK-$TITLE and the metadata is not available, the MME will assign the track name “**`01-Unknown Title.wma`**”. If you have another CD with no metadata, you will get the same track name for tracks on that CD. Setting the **`<IgnoreNonAccurate>`** flag tells the MME to ignore the filename template you requested and fall back to the default template specified in the **`config.h`** and **`mme.conf`** configuration files. A template such as “$0TRACK-$TITLE_$SRCFID” will guarantee that all filenames are unique by appending the source tracks’ file IDs (*fid*s) to the filenames.

Media copy and ripping constants

The table below lists default media copy and ripping management constants.

Constant	Value	Description
CONFIG_DEF_COPYING_MODE	0	Perform media copy and ripping in the background. Playback has priority.
MME_MEDIACOPIER_MODE_BKG	1	Perform media copy and ripping in the foreground. Copy and ripping have priority.
CONFIG_DEF_COPYING_DEST_MEDIA_STORE	2	The default destination folder for copy and ripping. See “Configuring the destination” above.
CONFIG_DEF_COPYING_ENCODEFORMATID	2	The default ripping is to WAV format.
CONFIG_DEF_COPYING_DEST_FOLDER	*	The default destination folder for ripping. Name must be unique. See “Configuring the destination” above. * Default value is “/ripped/$ARTIST/$ALBUM/”.
CONFIG_DEF_COPYING_DEST_FILENAME	**	The default filename template for copied and ripped files. See “Files with insufficient metadata” above. ** Default value is “$0TRACK-$TITLE(date=$DATESTAMP,time=$TIMESTAMP,srcfid=$SR

continued…

Constant	Value	Description
CONFIG_DEF_COPYING_PRESERVEPATH_ENABLED	**`false`**	Enable creation of folder paths to the copied file. If set to **`false`** the copy operation will copy all files to the same folder.
CONFIG_DEF_COPYING_IGNORE_NONACCURATE_ENABLED	**`false`**	Enable use of default copying template if metadata is not accurate. If set to **`false`**, files with inaccurate or incomplete metadata may produce copied files with metadata that is not unique.

Chapter 10

Configuring Other Components

In this chapter. . .

The MME's behavior can be configured by settings chosen for its components and plugins, and for the QDB, and for some Neutrino components that are external to the MME. This chapter includes:

- Configuring other MME components
- Configuring other components
- Media format support

Configuring other MME components

Information about how to configure the MME component utilities can be found under the relevant sections of the *MME Utilities Reference*.

Information about some MME component configurations whose effect are most directly visible to MME clients, are described in this section:

- Configuring how **`io-media`** handles playback read errors
- Configuring **`io-media`** for video
- **`io-media`** **`strict`** option
- Configuring the time **`io-media`** waits for metadata
- Override the default graph for slot types
- Location of **`io-media`** DLLs

For information about how to change the **`io-media`** configuration file, see "Updating the **`io-media`** configuration file" in **`io-media`** in the *MME Developer's Guide*.

Configuring how `io-media` handles playback read errors

You can configure the MME's read error recovery behavior to best suit your end user's environment by configuring how **`io-media`** handles read errors. You can configure:

- the distance of the first skip forward. Default is 200 milliseconds.
- the percentage to increase the forward skip by at each subsequent retry. Default is 100 percent (double the skip forward time at each retry).
- the maximum number of skip-forward retries before failure. Default is 10 retries before failing.

To configure **`io-media`**'s behavior when it encounters problems reading media, you need to configure the **`io-media`** module **`mmf_trackplayer`** behavior. Syntax is as follows:

- **`skip-on-error=`**no — don't attempt to skip forward.

- **`skip-on-error=`***n* — jump once by *n* milliseconds; if this read is unsuccessful, give up and report a read failure.
- **`skip-on-error=`***n,p* — jump by *n* milliseconds; if a read is unsuccessful, repeat, increasing the jump distance by *p* percent each time until the end of the track is reached.
- **`skip-on-error=`***n,p,m* — jump once by *n* milliseconds; if a read is unsuccessful, repeat, increasing the jump distance by *p* percent each time, up to *m* times, or until the end of the track is reached.

For more information, see **`io-media-generic`** in the chapter MME Utilities Reference, and “Playback read error recovery” in the *MME Developer’s Guide* chapter Playback Errors.

Configuring `io-media` for video

You can configure **`io-media`** for optimal video performance on your system. You should use the **`io-media`** configuration file to:

- set the number of threads used to decode video
- limit the H.264 codec profiles your system will play

For more information about using the **`io-media`** configuration file to configure resources, see **`io-media-generic`** in the *MME Utilities Guide*

For more information about how to use video output device modifiers, see “Adding modifiers to a video output device URL” in the chapter Control Contexts, Zones and Output Devices of the *MME Developer’s Guide*.

Configuring the number of video decode threads

Adjusting the number of threads the system uses when decoding video can significantly affect performance, especially on systems with multiple cores. The **`mmf`** resource MM_IPP_VIDEO_DECODER_NUM_THREADS is available for configuring the number of threads used for video decoding.

To set the MM_IPP_VIDEO_DECODER_NUM_THREADS resource, add an entry in the **`mmf_graphbuilder`** section of the **`io-media`** configuration file.

For example:

```
resource {
    name = "MM_IPP_VIDEO_DECODER_NUM_THREADS"
    type = long
    value = 2
    optional = yes
}
```

The example above instructs **`mmf`** to use two threads to decode the video. This number of threads would be optimal for a single-core Atom processor. Four threads would be more suitable for a dual-core Atom processor **`"value = 4"`**.

Automatic allocation of video threads

By default, **`mmf`** is configured to have the H.264 decoder (**`ipp_h264_decoder.so`**) autodetect the number of CPUs on your system and set the number of video decode threads for optimal performance.

To explicitly set the configuration so that the H.264 decoder autodetects CPUs and sets the number of video decoding threads, set the MM_IPP_VIDEO_DECODER_NUM_THREADS resource *value* to 0 (**`value = 0`**).

For more information about using the **`io-media`** configuration file to configure resources, see **`io-media-generic`** in the *MME Utilities Guide*

Limiting H.264 profiles

Video processing can be CPU intensive and, depending on the codec profile, may not perform optimally on all platforms. The **`mmf`** includes the resource MM_MP4_PARSER_ACCEPT_H264_PROFILE, which you can use to limit the H.264 codec profiles that your system will play; that is, to reject video files with H.264 codec profiles that your system does not have adequate resources to play smoothly.

The default behavior is to play all H.264 profiles. If you set the MM_MP4_PARSER_ACCEPT_H264_PROFILE resource, **`io-media`** will reject as unplayable any video files whose profiles you have not specified.

For example, to instruct **`mmf`** to decode only files with Baseline or Main profiles, add the following to the **`mmf_graphbuilder`** section of your **`io-media`** configuration file:

```
resource {
    filter = "parser"
    name = "MM_MP4_PARSER_ACCEPT_H264_PROFILE"
    type = long
    value = 3
    optional = yes
}
```

MM_MP4_PARSER_ACCEPT_H264_PROFILE

The MM_MP4_PARSER_ACCEPT_H264_PROFILE resource is a bitfield of playable profiles, as described in the table below:

Profile	Value
Baseline	1
Main	2
Extended	4

continued...

Profile	**Value**
High	8
High 10	16
High 422	32
High 444	64

MM_MP4_PARSER_*

The MM_MP4_PARSER_* constants define the maximum levels and bitrates permitted for each profile:

- Baseline:
  - MM_MP4_PARSER_MAX_H264_BASELINE_LEVEL
  - MM_MP4_PARSER_MAX_H264_BASELINE_BITRATE
- Main:
  - MM_MP4_PARSER_MAX_H264_MAIN_LEVEL
  - MM_MP4_PARSER_MAX_H264_MAIN_BITRATE
- Extended:
  - MM_MP4_PARSER_MAX_H264_EXTENDED_LEVEL
  - MM_MP4_PARSER_MAX_H264_EXTENDED_BITRATE
- High:
  - MM_MP4_PARSER_MAX_H264_HIGH_LEVEL
  - MM_MP4_PARSER_MAX_H264_HIGH_BITRATE
- High 10:
  - MM_MP4_PARSER_MAX_H264_HIGH10_LEVEL
  - MM_MP4_PARSER_MAX_H264_HIGH10_BITRATE
- High 422:
  - MM_MP4_PARSER_MAX_H264_HIGH422_LEVEL
  - MM_MP4_PARSER_MAX_H264_HIGH422_BITRATE
- High 444:
  - MM_MP4_PARSER_MAX_H264_HIGH444_LEVEL
  - MM_MP4_PARSER_MAX_H264_HIGH444_BITRATE

`io-media` `strict` option

`io-media` has an optional **`strict`** option that limits the parsers it tries to use for a file to the preferred parser associated with the file's file extension. This option causes **`io-media`** to fail immediately and return an error if a specified parser fails to identify a file.

Setting the **`strict`** implies that **`io-media`** rejects files with extensions that do not match their contents, as well as files without extensions; but it improves performance, especially when handling damaged files, because **`io-media`** does not waste time sniffing files with every parser in its list or on the system (with each parser taking 10 seconds or more to timeout) before **`io-media`** returns and reports an error.

To use the **`strict`** option, modify your **`io-media`** configuration file (**`io-media-generic.cfg`** to include "strict = yes" for each parser for which you want this option. For example:

```
format {

    url =  "*.mp[a123] "
    parser =  "mpega_parser "
    strict = yes
}
```

- The **`strict`** option is optional; it does not have to be included in your **`io-media`** configuration file. However, if you include it, it must be set to **`yes`**.
- To maintain consistent behavior across your system, use the same **`strict`** option for all your parsers.
- You may prefer inconsistent behaviour for your projects. For instance, you might prefer files on a CD-ROM to use a strict policy to speed up failures, but files on a hard drive or a USB stick to be probed more thoroughly because these devices are faster and don't get scratched as easily as CDs. To configure your system for this sort of behavior, you can split the format descriptions information in two, for example, by adding an entry for **`/fs/cd?/*.mp[a123]`** before the entry for **`*.mp[a123]`**.

Configuring the time `io-media` waits for metadata

An **`io-media`** option allows you to set a maximum time **`io-media`** will wait for metadata from device, such as an iPod, that manages its own track sessions.

This option applies to the **`mediafs_2wire`** graph. The default waiting period is 200 milliseconds.

To configure this option, change the value of **`metadata-timeout`** in the **`io-media`** configuration file:

```
module-options {
    module =  "mediafs_2wire "
    metadata-timeout = 200
}
```

How the `io-media` configuration affects metadata events

The period set by **`metadata-timeout`** defines the maximum amount of time **`io-media`** waits for a device to deliver metadata.

After a track change **`io-media`** waits for metadata from the device. until one of the following conditions is true:

- The device delivers metadata.
- The timeout period expires.

When either of the above conditions is true, **`io-media`** sends an event to the MME to indicate to the MME that it should send a MME_EVENT_NOWPLAYING_METADATA event to the client application.

If the device doesn't deliver metadata, **`io-media`** continues waiting for it, until either the device delivers metadata, or the track changes.

When the device delivers metadata, **`io-media`** sends an event to inform the MME, and the MME sends a new MME_EVENT_NOWPLAYING_METADATA event to the client application.

The client application should query the **`nowplaying`** table for metadata after every MME_EVENT_NOWPLAYING_METADATA event it receives. If there is no metadata in the **`nowplaying`** table, the client application can display "Unknown *" or another appropriate message string to the user. If metadata is present, it can display the metadata, as required.

Configuring the metadata timeout for your devices

The timeout period can be used to avoid flicker on the HMI screen by having the client application wait an appropriate amount of time before refreshing the screen after a track change.

For example, if a device takes 220 milliseconds to deliver metadata, leaving the **`metadata-timeout`** at the default 200 milliseconds will cause the HMI to display "Unknown *" message strings after 200 milliseconds, then 20 milliseconds later refresh the screen with the metadata — causing the screen to flicker for the end-user.

In a case such as this, in order to give the device time to deliver the metadata, you should set the **`metadata-timeout`** to 230 (or more) milliseconds. Your HMI can either clear the screen at the track change, leaving it blank for the 230 milliseconds between the track change and the arrival of the metadata, or leave old information on the screen, then update the screen once when it has metadata to display.

Override the default graph for slot types

The MME includes configuration elements that let you override the default graph associated with a slot type. These elements are the top-level **<RendererOverride>** element, and the **<renderer>** element. The table below describes the **<renderer>** element and its attributes:

Element	Attributes	Default	Description
<renderer>	*storagetype*		Override the default graph for the storage type specified by the *storagetype* attribute (if used), at the mountpath specified by *mount*, and use the graph specified by *name*.
	mount		The path for the media whose default graph is to be overridden.
	name		The graph to use for media at the specified mountpath.

To override the default graph for slot types, add the new elements to the MME configuration file, under the **<Configuration>**, and set the attributes for the **<renderer>** element following the example below:

```
<RendererOverride>
    <renderer storagetype="1" mount="/fs/cd0" name="cam_trackplayer"/>
    <renderer storagetype="5" mount="/fs/shinwa0" name="trackplayer"/>
    <renderer storagetype="1" name="audiocd_player"/>
    <renderer mount="/media/drive" name="diskdrive_player"/>
</RendererOverride>
```

If the *storagetype* attribute is not specified, the specified graph is used for *all* mediastores mounted at the path specified by *mount*.

Location of `io-media` DLLs

io-media DLLs All **io-media** DLLs should be installed in their own, exclusive directory, such as, for example, **/lib/dll/media**.

No other libraries should be installed at this location, because **io-media** loads *all* DLLs from its DLL directory in order to check if it can recognize them as MMF filters or as other Addon Interface (aoi) DLLs. Placing **io-media** DLLs in a directory with other DLLs, such as for example, **/proc/boot**, may adversely affect system performance.

For more information about **io-media** DLLs, see "Configuring **io-media** to use DLLs" in the *MME Utilities Reference* chapter **io-media-generic**.

Configuring other components

This section describes some important configurations that affect MME behavior and perfomrance.

Configuring the QDB

The **qdb** SQL database is a core component of the MME system. You can alter its behavior by setting command-line options and setting pragmas.

QDB command-line options

For a complete list of QDB command-line options, see the chapter Starting QDB in the *QDB Developer's Guide*, which describes the QDB command-line options.

Setting pragmas

PRAGMA is a special command used to modify the operation of the QDB process or to query the library for internal (non-table) data. The available pragmas fall into these basic categories:

- Pragmas used to modify the operation of the QDB process in some manner, or to query for the current mode of operation
- Pragmas used to query the schema of the current database
- Pragmas used to query or modify the databases two version values, the schema-version and the user-version
- Pragmas used to debug the library and verify that database files are not corrupted.

For a full description of the pragmas you can set, see the **PRAGMA** command documentation in the *QDB Developer's Guide*.

Configuring Neutrino components external to the MME

These components are also used by the MME, but are not covered here. You can refer to their entries in the Neutrino *Utilities Reference* for configuration and startup information:

- **devf-generic** — additional customization may be possible using the Flash Filesystem and Embedding TDK.
- Audio drivers — the entry for **io-audio** in the Neutrino *Utilities Reference* also lists the available **deva-*** audio drivers.
- **io-usb** — additional customization may be possible using the USB Driver Development Kit.

Media format support

When you are configuring your MME system, you should make sure that you install all the components required to support the media formats you will be processing. For information about formats supported by third party components and the required MME components for these, see the appendix Binary File Dependencies.

Chapter 11

Configuring Internationalization

In this chapter. . .

This chapter explains how to configure the MME and its components to support different languages:

- Language and locale settings
- Adding languages
- Creating an external DLL to provide character encoding routines

Language and locale settings

The **`<Locale>`** element in the MME configuration sets the MME's default language. This setting defines the language strings the MME uses for display messages.

The locale code is a string containing a 5-character language and region code. This consists of a 2-character ISO639-1 language code, followed by a "_" character, followed by a 2-character ISO3166-1 alpha-2 region code. See **`http://www.loc.gov/standards/iso639-2/php/code_list.php`**.

At present the MME uses only the first two characters of this code. The default language is English: **`en`**.

If the MME does not find a **`<Locale>`** element in its configuration file, it uses the value in CONFID_DEF_LOCALE as its default langauge setting. The default setting for CONFID_DEF_LOCALE is **`en`** for English. It is good practice to set this value to the prefered default language for the environment where the MME will run.

For information about how to manage access to DVDs based on region codes, see "Managing DVD access" in the *MME Developer's Guide* chapter Playing and Managing Video and DVDs.

Setting the language preferences

To use a language other than English as the default language, you must:

- populate the MME's **`languages`** table with the appropriate strings; see "Adding languages" below for more information.
- change the default language by changing the two-letter code for the **`<Locale>`** element

To change the language the MME will use for strings that indicate unknown media metadata locale and language, call the function *mme_setlocale()*.

Setting the preferred playback language

Neither the configuration file language setting nor the language set with *mme_setlocale()* affect the preferred language for media playback.

Use *mme_media_set_def_lang()* to set the preferred language for media output. Typically, you should set this attribute soon after you connect to the MME, but you can do it at any time to change the preferences. To get the current preferred language, call *mme_media_get_def_lang()*.

Adding languages

This section describes how to add supported languages to the MME.

Customizing display messages

You can customize the MME to display static message strings in languages other than English, by populating the **languages** table with translated strings, and then setting the *active* field to **1** for that row. For example, to set the language to Japanese, replace the default contents of the **languages** table with this:

```
UPDATE languages SET active=1 WHERE language='Japanese';
```

Different languages use different sorting rules even if they use the same alphabet. After making a language change, your client application will need to re-build the **library** table and its support tables using the sorting rules for the new language:

```
BEGIN TRANSACTION;  REINDEX ***; COMMIT;
```

Rebuilding the **library** table takes time. Do not undertake this task unnecessarily.

Use the function *mme_setlocale()* to set the language for media with unknown language in the metadata, and the function *mme_media_set_def_lang()* to set the default language for media playback.

Setting language strings to NULL

You can set language strings to NULL. Language strings set to NULL can be coalesced into other values using the SQL engine (DMS or QDB), or they can just be received as NULL and handled by the client application. If you set langauge strings to NULL, your client application can detect when a language string has not been set up and populate it if necessary. To set a language string to NULL, simply insert the value into the **languages** table. For example:

```
-- English
INSERT INTO languages(
    language,
    lang_code,
    unknown,
    unknown_artist,
    unknown_album,
    unknown_genre,
    unknown_category,
    unknown_composer,
    synchronizing,
    unknown_language,
    unknown_conductor,
        unknown_soloist,
        unknown_ensemble,
        unknown_opus
        )
values(
        "English",
        "en",
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        );
```

Customizing the sort order

Different languages use different sort-order conventions. The MME includes support for setting the language for sorting data retrieved by SQL SELECT statements and sorted by the ORDER BY keyword. To enable this feature, you need to:

1. Install the library **`libqdb_cldr.so`** along with the files for language sorting included with your MME release.
2. Configure the QDB configuration file **`qdb.cfg`**.
3. Start QDB with the **`-s`** option specifying the default sort order language.

If you do not specify a collation sequence based on a language and location, data will be sorted by using the default sorting method: the SQLite BINARY collation sequence.

Install `libqdb_cldr.so` and the files for language sorting definition

Install the library **`libqdb_cldr.so`** with the other system libraries (any location in the **$LD_LIBRARY_PATH**) on your target. You can install the files for language sorting definition to **`/etc/cldr`**; the **`libqdb_cldr.so`** library will search here by default. For example, your files might be installed as follows: **`lib/libqdb_cldr.so`** and **`/etc/cldr/`***language_files*.

If you want to install the files for language sorting definition to a different location you can put the **`cldr`** directory (containing the definition files) along with the other configuration and definition files at the location you choose on your system, and set the path to this location in the **$QDB_CLDR_PATH** environment variable: **`QDB_CLDR_PATH= /my_config_file/cldr`**.

Configure `qdb.cfg`

Add the line **`Collation = cldr@libqdb_cldr.so`** to each section in the QDB configuration file **`qdb.cfg`**, telling the QDB to use the library **`libqdb_cldr.so`** for collation. For example:

```
[mme_library]
Filename = /fs/tmpfs/mme_library
Base Schema     = /home/jmammone/mme/alpha125/run/db/mme_library.sql
Backup Dir = /home/jmammone/mme/alpha125/run/db/bks1
Backup Dir = /home/jmammone/mme/alpha125/run/db/bks2
Compression = bzip
Collation = cldr@libqdb_cldr.so

[mme_temp]
Filename = /fs/tmpfs/mme_temp.db
Schema File = /home/jmammone/mme/alpha125/run/db/mme_temp.sql
Backup Dir = /home/jmammone/mme/alpha125/run/db/bks1
Backup Dir = /home/jmammone/mme/alpha125/run/db/bks2
Collation = cldr@libqdb_cldr.so
...
```

Start QDB

To select the sort language, use the **`-s`** option when you start QDB. This option sets the language sort order in the variable *cldr*. For example, to set US English as the sort order:

```
# qdb -vv -R set -c ./qdb.cfg -s en_US@cldr
```

To set French (France) as the default language sort order, replace **`en_US@cldr`** with **`fr_FR@cldr`**, as follows:

```
# qdb -vv -R set -c ./qdb.cfg -s fr_FR@cldr
```

CAUTION: If your QDB configuration file is set up to use the new collation library **`libqdb_cldr.so`**, you must specify the **`-s xx_xx@cldr`** option when starting QDB. If you do not use the **`-s`** option, QDB may produce unexpected results and compromise the integrity of your system.

Using the specified sort order language

To use the language sort order specified in *cldr*, add **`COLLATE cldr`** after the **`ORDER BY`** clause in your SQL statement. For example:

```
SELECT artist_id,artist
    FROM library_artists ORDER BY artist COLLATE cldr;
```

Setting a default sort order language in the schema files

The MME offers a method for setting a default sort order language, to avoid having to add **`COLLATE cldr`** to **`SELECT`** statements already in use in a client application. You can add a line to the **`CREATE TABLE`** statement for each table in the MME schema files, then regenerate your tables. For example, to use the language set in *cldr* in the **`library_artists`** table:

```
CREATE TABLE library_artists (
    artist_id    INTEGER PRIMARY KEY,
    artist       TEXT UNIQUE COLLATE cldr
    );
```

You can also change the sort order for keywords returned by SELECT statements by writing a *collation* function for the QDB. For more information, see "Writing User-Defined Functions" in the *QDB Developer's Guide*.

Creating an external DLL to provide character encoding routines

About character encoding and conversion in the MME

All MME interfaces use UTF-8 character strings. In the majority of cases, the MME and **`io-media`** can use the information they have about media formats to correctly convert into UTF-8 characer strings the character strings in the media they process .

The mojibake problem

While the MME and **`io-media`** can usually convert character strings into UTF-8, depending on the media files your implementation will be required to handle, some character encodings may produce mojibake (unintelligible character strings) for the following reasons:

- Not all file format specifications adequately define how to convert the encoded character strings in media files into a human readable format.
- Some media format specifications, such as some versions of ID3, which require that all character strings be encoded as ISO 8859-1 strings, do not adequately define character encodings for non-Western European alphabets and characters (Polish, Russian, Korean, etc.). Media publishers ignore the specification and use another specification to encode their character strings.
- The work-arounds implemented result in media files whose character strings are encoded according to character encoding specifications other than those required by the media format specifications. For example, an MP3 media file, which according to the ID3 media format specification should have its characters encoded as ISO 8859-1 strings, may contain character strings encoded as ISO 8859-2 to correctly encode Polish.

If your application is to support file formats whose specifications inadequately define encodings for character strings in your environment, the encodings that the MME and **`io-media`** propose by default may not be correct, and a custom DLL may be required to determine the actual encoding of the character strings.

Creating a character encoding conversion DLL

You can create an external DLL for the MME and **`io-media`** to:

- carefully examine character strings in media files to determine if they are in fact encoded according to the media format specifications
- in the event that the strings are not encoded as specified, determine their actual encoding
- convert the character strings into UTF-8 so they can be correctly displayed to the end-user

This DLL should include:

- your custom character encoding conversion function *convert_to_utf8()*
- your configuration function *convert_setup()*, so that your client application can ask the MME to pass a configuration parameter to your DLL; this parameter can be used to configure specific behavior, such as, for example, having the DLL bypass the encoding detection and assume a specified character encoding based on the information it receives from **`io-media`** (eg. assume ISO-8859-4 when **`io-media`** reports ISO-8859-4).

See "Character encoding conversion function prototypes" below for more information about these functions.

Loading the DLL

The DLL can be use by the MME and by **`io-media`**.

Informing the MME about character conversion routines

After you have built the DLL, put it somewhere in your **LD_LIBRARY_PATH** search path, then use the **`<CharacterEncodingConverter>`** element under the **`<Database>`** element in the **`mme.conf`** file to tell the MME the name of the DLL. For example:

```
<CharacterEncodingConverter dll="convert_utf8.so"/>
```

If you do not set the **LD_LIBRARY_PATH** environment variable, you must specify the full path to your custom DLL in the **`<CharacterEncodingConverter>`** element.

Informing **io-media** about character conversion routines

If you use a custom routine to perform character conversions, you must specify the conversion DLL in the **io-media** configuration file, under the **mmf** module options. For example:

```
module-options {
    module = "mmf"
    audio_writer = "mmipc_writer"
    keepdlls = "used"
    dlldir = "$MM_INIT"
    utf8hook = "convert_utf8.so"
}
```

For more information about **mmf** module options, see "**mmf** options" in the *MME Utilities Reference* chapter on **io-media**.

How the MME and io-media will use the conversion DLL

If you create a custom character encoding DLL, and configure the MME and **io-media** to use it, the DLL is used as follows:

- **io-media** calls into the DLL:
  - during the MME's second synchronization pass, which uses **io-media** and updates the metadata in its database
  - when it plays a track; if the MME does not have the metadata for the track marked as accurate in the library, **io-media** uses the metadata provided by the DLL
- Converted character strings go to all instances of **io-media**.
- The MME calls into the DLL for operations that may require character conversions in metadata. For example:
  - during the third (playlist) synchronization pass
  - when accessing other metadata sources, such as CD-TEXT

Character encoding conversion function prototypes

The MME defines two prototypes for functions you can write to detect and convert character encodings:

- *convert_setup()*
- *convert_to_utf8()*

convert_setup()

```
int convert_setup( const char *default_encoding,
                   int allow_detection )
```

Arguments

default_encoding	A pointer to a parameter. You must define the format of this string, which can include any information of interest to your character conversion DLL.
allow_detection	A flag that determines if the MME and the character conversion DLL are permitted to perform encoding detection. If it is set to 1 detection is permitted; if it is set to 0, detection is not permitted.

Description

The function *convert_setup()* provides an interface for passing a string from the MME to custom routines to help them determine character encodings. It should accept this string, and a setting determining if character encoding detection is permitted. The string can be, for example, the name of an encoding, such as "8859-4" or "shift-JIS", or a numeric value formatted as an ASCII string, or anything else that the HMI can provide to help the DLL determine what it should do with character strings.

The MME function *mme_charconvert_setup()* is designed to call *convert_setup()*. For more information see *mme_charconvert_setup()* in the *MME API Reference*.

CAUTION: The function *convert_setup()* may be called at any time. The MME provides no locking mechanism, so the DLL must maintain the integrity of its own variables.

If it is successful, this function should return 0 (zero). If this function is not successful, it should return -1.

convert_to_utf8()

```
int convert_to_utf8( const char *in_str,
                     char *out_str,
                     uint16_t in_size,
                     uint16_t out_size,
                     char *in_str_encoding_hint )
```

Arguments

n_str	A pointer to the input string in any encoding.
out_str	A pointer to the output string buffer to hold the resulting UTF-8 string.
in_size	The input size of the buffer, in bytes. If this parameter is 0, the buffer size is not known, and the DLL should call *strlen()* to find the buffer's length.
out_size	The output buffer size, in bytes.

in_str_encoding_hint

A hint at the encoding scheme. The hint is a string that will be null-terminated when called by the MME or **io-media**. See “Character formats” below for a list of defined input formats. For other input formats the MME and **io-media** pass NULL in *in_str_encoding_hint*.

Description

The function *convert_to_utf8()* should perform character conversions.

If it is successful, this function should return 0 (zero). If this function does not perform the requested conversion it should return -1: the MME and **io-media** will attempt to perform their default conversion.

You may choose to use this behavior to allow the MME to perform its default conversion with, for example, UTF-16. Assuming that UTF-16 strings are valid Unicode, your function can return a -1 when asked to convert these strings, thus passing on responsibility for the conversion to the MME and **io-media**.

Character formats

For convenience, **mm/charconvert.h** includes definitions for the following character formats:

- CHAR_FORMAT_ISO8859_1 — **iso8859-1**
- CHAR_FORMAT_ISO8859_2 — **iso8859-2**
- CHAR_FORMAT_UTF16BE — **utf16be**
- CHAR_FORMAT_UTF16LE — **utf16le**
- CHAR_FORMAT_SHIFTJIS — **shift_jis**
- CHAR_FORMAT_KOI8U — **koi8-u**
- CHAR_FORMAT_ISO2022_CN — **iso2022-cn**
- CHAR_FORMAT_WINDOWS_1251 — **windows-1251**
- CHAR_FORMAT_UTF8 — **utf8**

These character formats are literal strings, so the character encoding conversion DLL should use *strcmp()* (*not* “==”) to compare them. They can be used to initialize character arrays. For example:

```
static const char format_iso[] = CHAR_FORMAT_ISO8859_1;
```

Appendix A

Binary File Dependencies

In this appendix. . .

This appendix lists the media formats supported by the MME 1.1.0, and the binary files that must be installed to support these media formats. It is organized into two sections:

- Supported formats — the media formats supported by the standard MME product built with **io-media-generic**.
- Filter descriptions — brief descriptions of the filters used to process different media formats.

For information about support for specific boards, please see the documents at the QNX download center.

damping_audio_writer filter

MME release 1.1.0 includes the **damping_audio_writer** filter, which can can fade a PCM stream IN or OUT. This filter can be used instead of the **audio_writer** filter. To have **io-media** dynamically load **damping_audio_writer**, start it with options set as follows:

```
# io-media-generic -Mmmf,audio_writer=damping_audio_writer
```

For more information, see "**damping_audio_writer** filter" in the **io-media** chapter of *MME Utilities Guide*.

The QNX Aviage Multimedia Suite supports many different media formats. In order to ensure our media suite is robust, we test against industry standard media sets:

- the damaged media set from Almedio Inc. (formerly known as A-BEX Laboratories Inc.)
- the standard media library from P3 Systems

Any failure against these media sets is considered a valid error.

In addition to testing against these industry standard media sets, we test against large libraries of commercially available media. We consider any problems with commercially available media as valid errors, and correct these problems whenever possible.

All media files that we are unable to play, including those that have been intentionally corrupted or constructed in a way that does not comply with industry standards, are investigated for root cause. In the course of investigating such issues, we may compare the support our player provides for these media files with the support provided by other industry standard media players. Note, however, that:

- some players may not effectively check for corruption, and attempt to play files anyway
- desktop players may use resource-intensive strategies to make sense of corrupted files —strategies that require system resources not available in an embedded environment

Successful play of non-compliant files by a competitor's player does not mean that QNX will make its player work with files that do not conform to industry standards.

Supported formats

Legend	
Required	These binary files are required to support the media format.
Optional	These binary files are needed to support specific operations for the media format.
(S)	Statically linked.
(D)	Dynamically loaded (includes downloaded to DSP, and dynamically executed).

The tables below list the media formats supported by the MME built with **`io-media-generic`** and the binary files required for this support.

AAC

The files required to process AAC media with **`io-media-generic`** are:

- **`aac_parser.so`** — parser filter (D)
- **`audio_writer.so`** — writer filter (S)
- **`queue_filter.so`** — system filter (S)
- **`qnx_raac_decoder.so`** — decoder filter (D)
- **`stream_reader.so`** — system filter (S)

The files required to support specific operations for AAC media with **`io-media-generic`** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem. (S)
- **`http_streamer.so`** — AOI streamer, needed to read files from an HTTP server. (S)
- **`rawfile_writer.so`** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **`wavfile_writer.so`** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

CDDA

The files required to process CDDA media with **`io-media-generic`** are:

- **`audio_writer.so`** — writer filter (S)
- **`cdda_parser.so`** — parser filter (S)
- **`cdda_streamer.so`** — AOI streamer (S)
- **`queue_filter.so`** — system filter (S)
- **`stream_reader.so`** — system filter (S)

The files required to support specific operations for CDDA media with **`io-media-generic`** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem. (S)
- **`rawfile_writer.so`** — writer filter, needed to write output to a file instead of to an audio card. (S)

- **`tmpfile_streamer.so`** — AOI streamer, needed to play media from destination tracks during ripping. (S)
- **`wavfile_writer.so`** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

MP3

The files required to process MP3 media with **`io-media-generic`** are:

- **`audio_writer.so`** — writer filter (S)
- **`fraunhofer_mp3_decoder.so`** — decoder filter with integer arithmetic. Use either this filter *or* **`xing_mpega_decoder.so`**. (D)
- **`mpega_parser.so`** — parser filter (S)
- **`queue_filter.so`** — system filter (S)
- **`stream_reader.so`** — system filter (S)
- **`xing_mpega_decoder.so`** — decoder filter with floating point arithmetic. use either this filter *or* **`fraunhofer_mp3_decoder.so`**. (D)

The files required to support specific operations for MP3 media with **`io-media-generic`** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem when ripping. (S)
- **`http_streamer.so`** — AOI streamer, needed to read files from an HTTP server. (S)
- **`rawfile_writer.so`** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **`wavfile_writer.so`** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

MPEG4

The files required to process MPEG4 media with **`io-media-generic`** are:

- **`audio_writer.so`** — writer filter (S)
- **`ipp_h264_decoder.so`** — video decoder filter (D)
- **`mp4_parser.so`** — parser filter (D)
- **`queue_filter.so`** — system filter (S)
- **`qnx_raac_decoder.so`** — decoder filter (D)
- **`stream_reader.so`** — system filter (S)

The files required to support specific operations for MPEG4 media with **io-media-generic** are:

- **fildes_streamer.so** — AOI streamer, needed to read files from a filesystem. (S)
- **http_streamer.so** — AOI streamer, needed to read files from an HTTP server. (S)
- **rawfile_writer.so** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **wavfile_writer.so** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

Ogg Vorbis

The files required to process Ogg Vorbis media with **io-media-generic** are:

- **audio_writer.so** — writer filter (S)
- **ogg_decoder.so** — decoder filter (D)
- **queue_filter.so** — system filter (S)
- **stream_reader.so** — system filter (S)

The files required to support specific operations for Ogg Vorbis media with **io-media-generic** are:

- **fildes_streamer.so** — AOI streamer, needed to read files from a filesystem. (S)
- **http_streamer.so** — AOI streamer, needed to read files from an HTTP server. (S)
- **rawfile_writer.so** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **wavfile_writer.so** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

Ogg Vorbis encoding

The file required to encode media to the Ogg Vorbis format with **io-media-generic** is:

- **ogg_encoder.so** — encoder filter (D)
- **rawfile_writer.so** — writer filter (S)

The files required to support specific operations for encoding media to the Ogg Vorbis format with **io-media-generic** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem. (S)
- **`tmpfile_streamer.so`** — AOI streamer, needed to play media from destination tracks during ripping. (S)

WAV

The files required to process WAV media with **`io-media-generic`** are:

- **`audio_writer.so`** — writer filter (S)
- **`queue_filter.so`** — system filter (S)
- **`stream_reader.so`** — system filter (S)
- **`wav_parser.so`** — parser filter (S)

The files required to support WAV file encoding with **`io-media-generic`** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem. (S)
- **`http_streamer.so`** — AOI streamer, needed to read files from an HTTP server. (S)
- **`rawfile_writer.so`** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **`wavfile_writer.so`** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

WMA

The files required to process WMA media with **`io-media-generic`** are:

- **`audio_writer.so`** — writer filter (S)
- **`queue_filter.so`** — system filter (S)
- **`stream_reader.so`** — system filter (S)
- **`wma9_decoder.so`** — decoder filter (D)
- **`wma9_parser.so`** — parser filter (D)

The files required to support WMA file encoding with **`io-media-generic`** are:

- **`fildes_streamer.so`** — AOI streamer, needed to read files from a filesystem. (S)
- **`http_streamer.so`** — AOI streamer, needed to read files from an HTTP server. (S)

- **`media_streamer.so`** — AOI streamer, needed for access to streams from PlaysForSure devices and DRM support. (D)
- **`rawfile_writer.so`** — writer filter, needed to write output to a file instead of to an audio card. (S)
- **`wavfile_writer.so`** — writer filter, needed to write output to a WAVE file instead of to an audio card. (S)

Filter descriptions

The tables below describes the filters used to process different media formats. Your MME installation may not include all listed filters.

AOI Streamers

Filter	Description
`cdda_streamer.so`	Provides access to a data stream from a CD-ROM (CDDA data).
`fildes_streamer.so`	Provides access to a data stream from a filesystem.
`http_streamer.so`	Provides access to a data stream from an HTTP server.
`media_streamer.so`	Provides access to a stream from a PlaysForSure device; includes DRM support.
`streamer_dvd.so`	Provides access to a stream coming from a DVD-V or DVD-A disc.
`tmpfile_streamer.so`	Provides access to a file that could be potentially still growing; and includes special handling of EOF. Used to play media during priority background ripping.
`wms_streamer.so`	Provides access to a stream from a device, such as an iPod, connected to the Bluetooth stack.

Control filters

Filter	Description
`wms_control.so`	Controls a device connected to the Bluetooth stack.

Decoder and encoder filters

Filter	Description
`fraunhofer_mp3_decoder.so`	Fixed-point MP1, MP2 and MP3 decoder for ARMLE platforms.

continued...

Filter	Description
`ipp_h264_decoder.so`	Decodes H.264 video formats on Intel platforms.
`ogg_decoder.so`	Decodes the Ogg Vorbis format.
`ogg_encoder.so`	Encodes the Ogg Vorbis format (PPCBE, SHLE and x86 platforms).
`wma9_decoder.so`	Decodes the WMA9 audio format.
`xing_mpega_decoder.so`	Decodes the MP1, MP2 and MP3 audio formats. An FPU unit is requried on the platform.

Parser filters

Filter	Description
`aac_parser.so`	Parses the AAC audio format.
`cdda_parser.so`	Parses the CDDA audio format.
`mp4_parser.so`	Parser filter — extracts an AAC audio stream from an MP4 container.
`mpega_parser.so`	Parses the MP1, MP2 and MP3 audio formats.
`wma9_parser.so`	Parses the WMA9 audio format.
`wav_parser.so`	Parses the WAVE audio format.

System filters

Filter	Description
`libdspipc.so`	Driver interface library to MMIPC.
`stream_reader.so`	Abstracts the provenance of a stream (http, CDDA, filesystem, DRM stream); first filter in all **`mmf`** (version 2) graphs.

Writer filters

Filter	Description
`rawfile_writer.so`	Writes output to a file instead of to an audio card.
`wavfile_writer.so`	Writes PCM data, wrapped with a WAVE header, to a file instead of to an audio card. Does not do any encoding.

Index