QNX® Aviage Multimedia Suite 1.2.0

Introduction to the MME

For QNX® Neutrino® 6.3.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 February 13, 2009.

Contents

List of Figures

About This Guide

Introduction to the MME accompanies the QNX Aviage multimedia suite's multimedia core package, and is intended for application developers who will use the suite's MultiMedia Engine (MME) to develop multimedia applications. It includes:

- MME Architecture — overview of the MME architecture and functionality
- MME Quickstart Guide — easy "quickstart" instructions for getting the MME started and playing media
- MME Frequently Asked Questions — questions frequently asks by developers who use the MME, and answers to these questions

For detailed information about how to develop multimedia client applications with the MME, see the other MME documentation available to application developers:

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

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`

continued...

Reference	Example
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*
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 Architecture

In this chapter. . .

Overview

The QNX Aviage multimedia suite consists of several packages, including the multimedia core package, codec packages that provide WMA9, MP3, and AAC decoding and encoding, and software packages that support iPod and PlaysForSure media players.

The major component of the multimedia core package is the MultiMedia Engine (MME). The MME provides the main interfaces for configuring and controlling your multimedia applications. Designed to run on the QNX Neutrino OS, which can be installed on a wide variety of hardware platforms, the MME provides consumer-product developers a component-based solution that reduces the work required to develop and deliver multimedia products to their end customers.

The MME is designed to simplify and speed development of end-user applications that require device and filesystem access, content synchronization, playback control, and audio and graphics delivery. It handles multiple clients, sessions and streams, and abstracts hardware and protocol dependencies from other functional areas. It provides integration with a wide variety of media sources, including those requiring Digital Rights Management (DRM), and provides a high-level API for media transport control, device control and browsing, and media library management; and it automatically detects media devices and integrates their contents into a general database view. The applications the MME can be used to develop include:

- transport media systems
- in-seat entertainment systems
- medical device imaging and sound monitoring units
- industrial control systems

The MME lets Human-Machine Interface (HMI) developers apply their talents to designing the best possible user experience instead of focussing on managing the media. When you build a client application that uses the MME, you can focus on:

- designing and building the best possible user interface (HMI)
- implementing simple playback functionality such as track session creation, “play”, “next”, “pause” etc.
- configuring audio and video output

You need to know about the configurations for your system’s storage devices, but you can leave a long list of responsibilities to the MME:

- device and mediastore insertion and removal — HDD, CD, DVD, USB key with media, etc.
- mediastore synchronization — find, itemize, extract, and manage media content and metadata

- input and output media connection management
- extensible support for specialized consumer devices, as well as for hardware offload to digital signal processors (DSPs)

MME functional areas

The MME is designed to bring together media management and playback control, providing a single, consistent interface for client applications. Internally, it has the broad functional areas described below.

Mediastore access

Mediastores are any source for multimedia data, including hard drives, DVDs, CDs and media devices such as iPods or MP3 players. The MME's mediastore access capabilities include:

- detection of devices, and integration of content from static and dynamic media sources: drives, external players, USB stores, iPods, networks
- media stack and protocol implementations for diverse protocols: iPod Access Protocol, MTP, etc., many with DRM requirements
- management of different media filesystem and stream formats: DOS FAT32, UDF 2.01, etc.

Mediastore content management

The MME's mediastore content-management capabilities include:

- media metadata extraction and storage — tags, third-party metadata, cover art, etc.
- media content browsing and searching
- playlist extraction and custom playlist creation

Media playback and recording

The MME's media playback and recording capabilities include:

- media content recording, copying and ripping
- media stream processing:
  - a flexible plugin architecture for filters that decrypt, parse, decode, encode and encrypt media data streams
  - a transparent communication mechanism to external codecs for flexible HW/SW partitioning of filter functionality
- media stream rendering and deployment:
  - route to output devices (displays, amplifiers, headphones, etc.)
  - store registered and encoded media data to storage device (HDD)

The MME interface

The MME API includes a primary interface and a secondary interface. The primary interface (**mme**) offers the media management functionality required of a multimedia middleware platform, while the secondary interface (**qdb**) offers the required database functionality. Together the primary and secondary interfaces offer multimedia applications a consistent API that provides:

- **media transport**, **rendering** and **control** — control of playback, and of aspects of media rendering such as volume, brightness, and playback modes (**mme**)
- **notification** — receiving information about the status of media, devices and operations; and effecting changes to media library content (**mme**)
- **navigation** — browsing media devices that support internal navigation, such as a DVD menu system or an iPod (**mme** or **qdb**, or both)
- **database capabilities** — searching, sorting, organizing, and updating of media information and metadata (**qdb**)

Component-based architecture

The MME is comprised of several independent components. Each MME component executes independently as a Neutrino resource manager process. A resource manager is a user-level server process that accepts messages from other programs and, optionally, communicates with hardware.

The MME's component-based architecture delivers:

- *flexibility* — developers using the MME to build multimedia products can easily configure the MME and its individual resource managers to meet their specific needs.
- *easy deployment* — multimedia applications build with the MME can be deployed on a single processor, or on a distributed network of processors with no changes to their application code.
- *reliability* — the MME's resource managers all have their own failure and restart; a resource manager failure doesn't mean a system failure.
- *portability* — all MME components offer standard interfaces, such as POSIX or SQL.

The MME resource managers

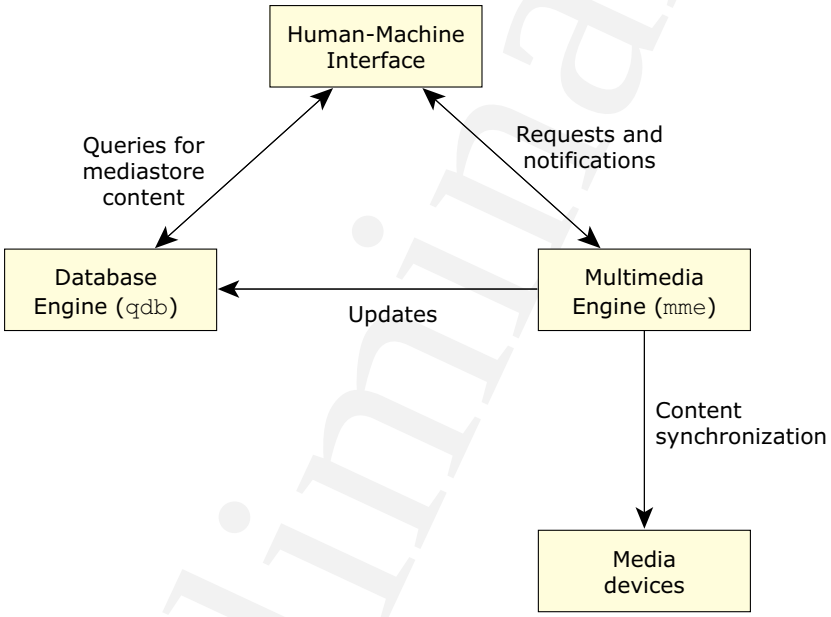
The MME resource managers can be placed into these groups:

- high-level resource managers that provide the interfaces to HMI client applications
- low-level resource managers that don't normally interface with HMI client applications

The resource managers that provide the interfaces to multimedia client applications are:

- the **qdb** database engine
- the **mme** multimedia engine

Both the **mme** and the **qdb** resource managers support device-specific functionality within a POSIX framework. Together they make up the interface to HMI client applications, providing them with an API to control, browse, copy or rip, and play media, as well as the ability to monitor and manage multimedia processing. The **mme** controls the low-level resource managers that directly access and process media data.



MME architecture showing **mme** *and* **qdb** *resource managers.*

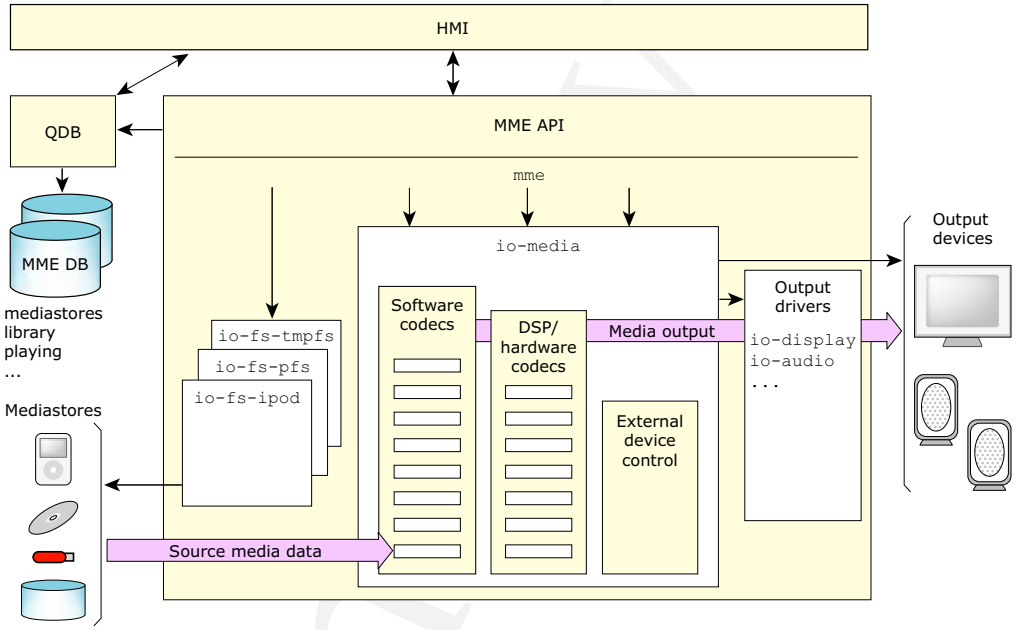
Multimedia client applications don't normally interface with the MME's lower-level resource managers. You may nonetheless find it useful to know about these resource managers and understand what they do, especially if you are tuning your system configuration. The MME's low-level resource managers include:

- the **io-fs** resource managers that provide access to media devices and mediastores
- the **io-media** resource managers that are responsible for media stream processing and rendering, for managing complex media streams, and for performing tasks that include CD and DVD playback, file playback, media copying, and media recording

Other low-level resource managers that are not specific to the MME, but which the MME uses, include:

- the **devb** resource managers, such as **devb-eide** and **devb-atapi**, that provide access to data on block-oriented devices and filesystems

- the **io-usb** resource manager that provides access to media on USB stores
- **io-audio**, **io-display**, and other resource managers that provide an abstraction of media output devices



High-level view of the MME components.

The MME database

The MME database is a repository of all information regarding media devices, media files and media streams known to the current MME instance. In addition to being a compendium of the metadata of all media files seen by the MME, this database stores MME system configuration and runtime states, such as track sessions and synchronization progress marks. Key data that the MME stores in its database includes:

- media files
- media file metadata
- track session runtime states
- progress state of mediastore synchronizations

A typical MME configuration might host the MME's database out of a RAM filesystem to enable fast performance on systems without persistent storage. In this type of implementation, the database could be made effectively persistent by configuring it to synchronize itself with a copy in flash memory.

Media file information and metadata

When the MME detects a mediastore, it synchronizes the mediastore content with its database. During synchronization, it generates a file ID with the file information and metadata for each media file described in its database. Client applications can reference this file ID to control media playback and other operations through the MME. Client applications can also pass SQL queries through MME API functions to select tracks for a particular album or artist to place in a track session.

To optimize performance when a mediastore is reinserted, the MME maintains synchronized media file information and metadata in its database even after a source mediastore is removed. However, to maintain database size within acceptable limits, when the MME adds new media entries to its database it may "prune" the database by removing stale entries to make room for the new information.

Runtime state and synchronization progress markers

The MME stores in its database the runtime state of track sessions, and the progress state of mediastore synchronizations.

The abstraction layer: `mme`

The **`mme`** resource manager is an abstraction layer and database manager in the MME. The functions it provides can be grouped into the following activities:

- mediastore detection and data synchronization
- media copying and ripping
- media browsing
- media playback
- drive and changer support

The **`mme`** resource manager has no device-specific APIs. It delegates state-sensitive device handling to device drivers, such as the drivers in the **`io-fs`** family.

Mediastore detection and data synchronization

Mediastore detection is the process by which the MME detects the presence of new mediastores. Synchronization is the process by which the MME examines mediastores and updates its database with information about the media files on the stores and with the metadata for this media. Through the **`mme`** resource manager, the MME performs mediastore synchronization in the following situations:

- when the MME application is launched
- when the MME learns of a change, such as a mediastore insertion or removal
- when a client application sends the MME a resynchronization command

Synchronizer selection

The **mme** resource manager uses several types of synchronizers: mediastore, metadata and playlist synchronizers. These synchronizers rate themselves based on their ability to handle different types of mediastores, files and playlists so that when the **mme** resource manager receives a request to synchronize a mediastore, it can select the best synchronizer for each task.

Synchronization processes

After it has chosen the most appropriate synchronizers for the task, the **mme** resource manager begins its three-step synchronization operation — media identification, metadata retrieval and playlist retrieval — to populate the MME library with the media relevant information and metadata.

Synchronization progresses as a recursive tree walk of the filesystem directory structure for the mediastore. The **mme** resource manager performs this recursive tree walk of the filesystem automatically, regardless of why the synchronization was started, *except* in the case of synchronizations initiated by the client application with a specific request to not perform this recursive tree-walk.

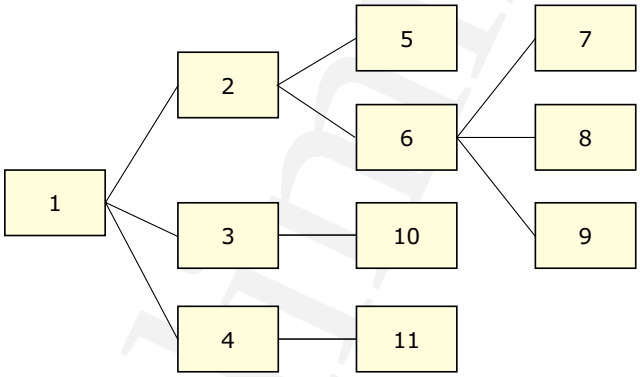


Illustration of the synchronization recursive tree walk

Filesystem change handling

The **mme** resource manager doesn't detect filesystem changes, such as media file creations and deletions, performed within filesystems. A client application that is aware of the changes must, therefore, instruct the MME to re-synchronize the modified mediastore. This re-synchronization can be done in a very broad or a very specific manner, depending on the needs of the application.

Media ripping

The MME currently supports the following modes for media ripping operations:

- background (muted) ripping
- priority background ripping

These ripping modes define the MME mediacopier's overall behavior and its interaction with playback track sessions. They can be configured by the client application developer.

The term "ripping" refers to both the copying and encoding of media files into a new format, and to the simple copy of media files without changing their formats.

Media ripping set up

The **mme** resource manager uses a copy queue table which lists the files to be ripped or copied, along with the details for each operation: source, destination, format, bitrate, etc. The client application can call an MME API function to insert entries into this table, specifying destination file parameters such as encoding format, name template, and destination folder paths, then call another API function to start the media copy or ripping operation.

Metadata for copied and ripped files

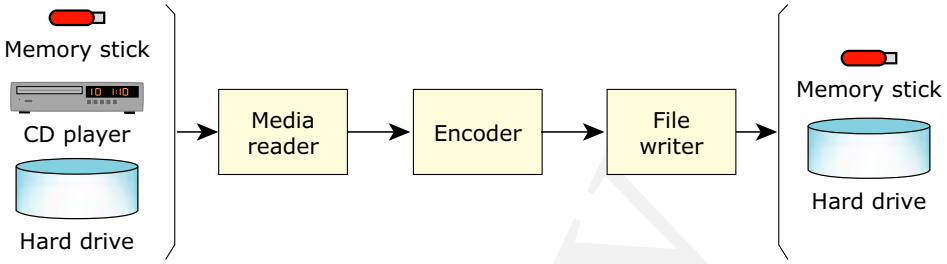
The media ripping operation uses the MME's metadata synchronizers to obtain metadata for the ripped media files (Gracenote, cdtext, etc.) and to automatically:

- generate metadata tags for the copied or ripped files
- insert the metadata into the appropriate MME database table, making it available for use by the MME and its client applications

Background (muted) media ripping

Background ripping operations are performed independently of playback operations. However, when accessing media sources (such as CD drives) that don't support multiple readers well, the **mme** resource manager gives precedence to playback operations. If a user tries to play media from a mediastore being used for background ripping, the mediacopier:

- cancels all copy and ripping operations from that mediastore
- discards any partial data
- automatically continues to the next copy ID in the copy queue table to attempt a new ripping operation



MME background ripping operation.

When playback on the mediastore ends and it becomes available for ripping operations again, the client application can instruct the **mme** resource manager to resume the operation. The **mme** resource manager will resume the ripping process, starting with the first entry in the copy queue table. In most cases, this entry will be the entry that was dropped when playback started. However, because the MME doesn't prevent the client application from inserting new entries into the copy queue when ripping is blocked by playback, it can't guarantee that a resumed ripping operation will begin with the file that was dropped when the operation was blocked.

The mediacopier blocks when it has processed all entries in the copy queue table.

Priority background ripping

The **mme** resource manager's priority background ripping mode grants the mediacopier priority access to the mediastore over playback operations. If a playback process needs to access tracks from a mediastore being accessed by the ripping operation, it uses the ripped tracks, not the source tracks on the source mediastore.

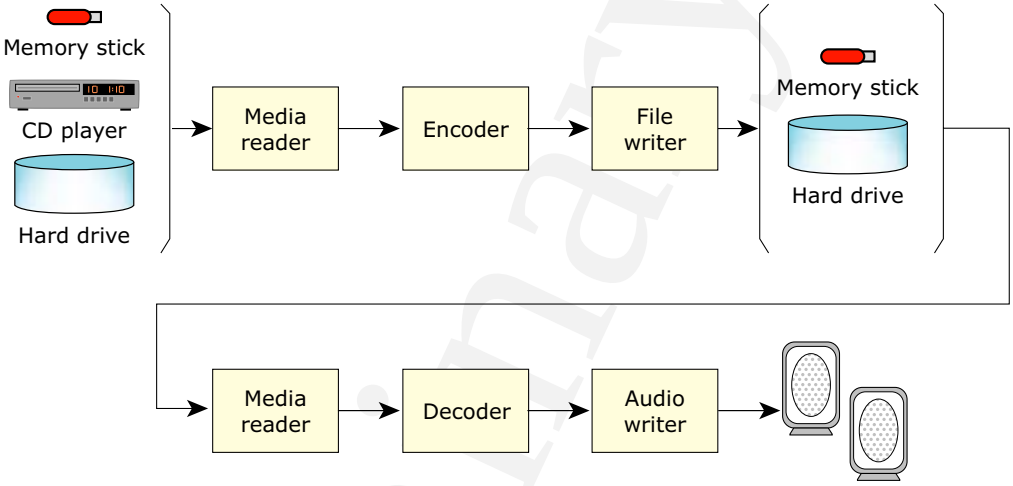
Priority background copying and ripping assumes that the system has sufficient computing resources to stay ahead of any requested playback and ensure continuous play. Priority background copy and ripping proceeds as follows:

- To allow the mediacopier thread to stay ahead of the playback session, the **mme** resource manager uses the playback track session to define the order of tracks to be ripped. This order is not necessarily the order of entries in the copy queue table.
- To ensure continuous play, the **mme** resource manager blocks playback of a ripped track until a sufficient amount of data has been ripped to allow playback to begin and complete without interruption.
- If, during priority background copying or ripping, the user selects "next" or "previous", the mediacopier either continues copying or ripping the current track or drops this track and begins a new track:
  - If the track required to fulfill the "next" or "previous" request has already been ripped, the mediacopier simply continues processing the current track.
  - If the track required to fulfill the "next" or "previous" request has not been copied or ripped, the mediacopier drops the current source track, discarding its data, and begins processing the newly requested track.

- If playback is stopped, the mediacopier continues processing the current track.

- The playback thread is free to perform trick mode operations, such as fast forwarding and rewinding, within the range of the copied or ripped data.

As with background media ripping, the mediacopier blocks when it has processed all entries in the copy queue table.



MME priority background ripping operation.

Media browsing

The MME offers users a directed synchronization API function to browse media files inside a device as a hierarchical tree. MME client applications can direct synchronization to start from a device's root directory ("/" or from any other directory whose path is known) and use the metadata from the synchronized media to provide the client application with the information the end-user needs in order to continue browsing. For example, the client application can begin browsing an iPod device at a known path location such as **`/Music/Genres`**, then use the data it retrieves to compose folder and file lists that the end-user can use to continue browsing.

Media playback

The MME uses the **`io-media`** resource manager to handle media playback and rendering, and to implement graph classes for media stream rendering and transport control. The client application can send messages requesting play, pause, stop, change volume, etc. to the MME, which transmits these to **`io-media`**. When it receives a play command, **`io-media`** issues commands to the correct graph instance to start playback. During playback, **`io-media`** sends media state notifications (position, speed, direction, etc.) to the MME, which then forwards these back to the user application.

Drive and changer support

The MME's extensible architecture supports both internal and external changers. With appropriate components, the MME can access internal changers through a **devb** driver, and external changers through a bus.

The MME supports the creation of track sessions that span multiple disks. It monitors disk availability in the disk changer and issues slot change requests as needed. It looks for disk insertions, ejections and slot changes, and updates its database to track what media is present in the changer, and what media is active in the changer.

Filesystem management: io-fs

The MME supports media files on multiple filesystems, and automatically responds to the presence of new devices and mediastores. Supported filesystems include:

- block devices — HDD, DVD, CD, USB, etc.
- flash devices — NAND, NOR, etc.
- device abstractions — iPods, PFS devices, etc.

To interact with media device filesystems, the MME works through the **io-media** resource manager, which interacts differently with different filesystems. The **io-media** resource manager interacts:

- directly with standard filesystems
- through the **io-fs** filesystem framework for all other types of filesystems, including link-connected and removable-volume devices

The **io-fs** resource managers are a filesystem framework with media device specific extensions that support media devices such as iPods and PFS devices. In the MME, the **io-fs** resource managers provide:

- device discovery
- a filesystem representation of the media files on a device
- primitives for retrieving media file metadata
- playback control commands

The current set of **iofs** modules includes:

- **iofs-tmpfs** — connects to temporary filesystems in memory
- **iofs-pfs** — connects to PlaysForSure devices
- **iofs-ipod** — connects to iPod devices

Device discovery

The **io-fs** resource managers offer automatic device discovery. When they detect a new device on the MME system, they retrieve information from the device about its capabilities and make this information available to the MME. This information may include:

- content — media files, DRM negotiations
- capabilities — track session management, streaming, playback controls such as “repeat” and “random”

iofs-tmpfs module

The **iofs-tmpfs** module is integrated as part of **io-fs-media**. It is a driver used by the MME to avoid the performance costs of running on slow storage devices, such as flash drives. It provides the MME with a POSIX filesystem interface that allows RAM to be used as the storage medium. To have the MME’s implementation of the QDB database engine run only in RAM, simply configure the QDB database to use the filesystem mount path configured in **tmpfs**.

iofs-pfs module

The MME uses the **iofs-pfs** module to connect to, and browse and play media on PlaysForSure (PFS) devices. PFS is a Microsoft media standard for devices that use the Media Transport Protocol (MTP) and implement Digital Rights Management (DRM).

PFS connectivity

The **io-fs** implementation of PFS connectivity comprises three layers:

- PFS module layer
- MTP layer
- PTP layer

PFS module layer

The **io-fs** PFS module layer is responsible for presenting a filesystem view of the device to the **io-fs** resource manager. When **io-fs** initializes the PFS module, it sets up a structure filled with function pointers that it can call into. In this way, the PFS module can “translate” POSIX commands into MTP requests, and MTP requests into POSIX commands.

MTP layer

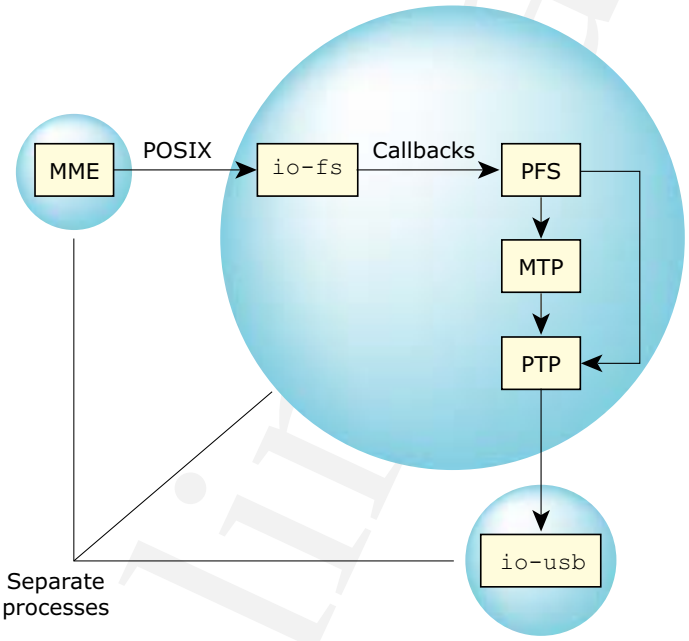
The MTP layer is Microsoft-supplied software that handles MTP messages. QNX has modified the MTP library to run in the QNX Neutrino environment.

Devices that support MTP provide a view of media content consisting of objects with properties. These objects and their properties are accessed via a command-and-response protocol with an optional data transfer phase.

PTP layer

The PTP layer handles Picture Transfer Protocol (PTP) messages. PTP is an implementation of the Still Image class of USB service, a protocol used to access multimedia content on digital cameras and PFS devices. The **io-fs** PTP layer communicates directly with **io-usb**, the Neutrino USB driver.

The figure below shows the relationship between the MME system, the **io-fs** PFS module, the MTP and PTP layers, and **io-usb**.



The MME and the **iofs-pfs** *module.*

Digital Rights Management (DRM)

The MME's **iofs-pfs** module uses Indirect License Acquisition (ILA) to obtain the key to decrypt DRM protected media content — to play a song, for example. The **iofs-pfs** module obtains this key in a license response message from a PlaysForSure device when it registers itself with that device. Registration with PFS devices must be renewed periodically. Registration with a PFS device is initiated when a user first attempts to access DRM protected content on the device. When the MME's **iofs-pfs** module first encounters DRM protected content on a PFS device, it:

- identifies the media objects in the device's mediastore that have been encrypted using Microsoft's WMDRM technology

- uses the MTP DRM extensions to register itself with the PFS device. This registration process includes a proximity check: the PFS module has the PFS device verify that a receiver is nearby and available to accept the registration.

DRM content decryption uses an AES block cipher with a 128-bit key. The AES key is unique to each playback session: it is different every time a song is played.

When the **iofs-pfs** module registers itself with the PFS device, it sends the device a certificate. The certificate contains the 1024-bit RSA public key that the device will use to encrypt the seed used to determine the AES key.

When the user selects DRM-protected content:

- The PFS module requests a license for that content from the PFS device.
- The device returns the license, which includes an encrypted seed.
- The PFS module uses a private RSA key to:
  - decrypt the seed
  - use the seed to determine the key for the AES block cipher.
- The PFS module processes every 128 bits of encrypted content using the 128-bit AES key to yield the decrypted content.

iofs-ipod module

The **iofs-ipod** module provides the MME with a filesystem view of a connected iPod device.

Using the iPod directory structure

The **iofs-ipod** module creates a directory structure from a connected iPod by querying the device's internal database. Each item on an iPod's menu is a database query. For example, selecting **Albums** queries the database for albums. Each further item on an iPod's menu is a subquery of the query represented by the parent menu item. Using this organizing principle, the iPod module creates for the MME a filesystem directory structure that resembles an iPod menu structure.

This filesystem directory lets the HMI perform operations on the iPod via the MME. For example, performing **cd Music; ls** through the MME produces the same result as a user selecting the "Music" option on the iPod. Both actions yield the same item list.

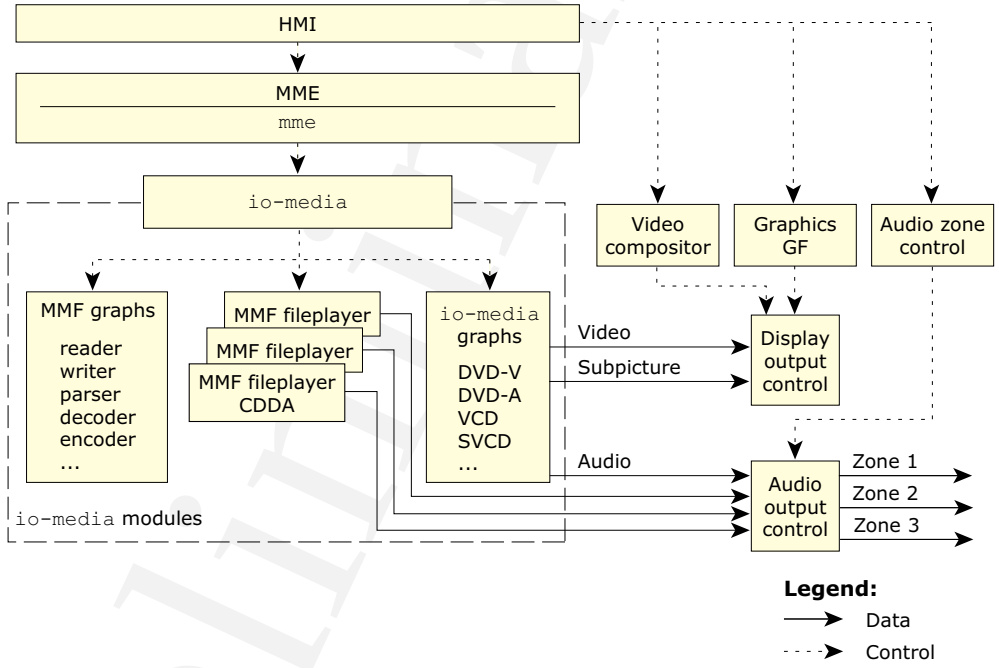
Supported iPod functionality

The MME can assume responsibility for sending control commands to the iPod to initiate playback, to stop or pause play, etc., and for routing the iPod's analog audio output directly to an amplifier. However, since iPod devices do not export their digital content, playback of content on an iPod device remains on the iPod device itself.

Media transport, control and rendering: `io-media`

The **io-media** resource manager provides low-level transport, control and rendering of media data. HMI access to **io-media** and its functionality is provided through the MME API and its **mme** abstraction layer.

The **io-media** resource manager uses graphs to process media tracks. Different graph classes perform different tasks, such as playing or ripping media tracks. The MME specifies the task, the media input source, the output device, the class of graph it requires, and a name for the graph instance it will use. The **io-media** resource manager uses this information to create a graph instance, which processes the media track.



*The **io-media** resource manager in the MME.*

In the figure above, control lines are shown as dashed lines, and data lines are shown as solid lines.

During media processing, the MME can send commands to **io-media** — for example, to pause playback or stop recording a stream, and **io-media** communicates progress and events, such as read errors, back to the MME, which reports these back to its HMI client application.

The MME can request multiple tasks from **io-media**; for each task **io-media** creates a new graph instance to handle the processing. At any point in time, each client application connection to the MME may have one instance of a playing graph, plus other graph instances performing background activities, such as ripping or capturing a live media stream to a buffer.

Different **io-media** variants bring together as a single executable selections of different modules, with each module implementing a selection of graphs.

trackplayer io-media graph class

trackplayer is a general purpose class of **io-media** graph dedicated to playing standard CDs (LPCM), MP3 files and WMA encoded files. It can interface to a standard CD or DVD drive, and includes multiple components that perform tasks, such as decoding, parsing and streaming.

The **trackplayer** is mounted on the QNX filesystem as **/dev/io-media/trackplayer**, and instances of the class appear as files in that directory. Depending on the performance of the platform, the MME can use several instances of **trackplayer** to drive separate audio streams to different output zones. For instance, in an automobile, passengers in the front seats and in the back seats could listen to different audio streams, with each stream being handled by a different instance of **trackplayer**.

The dvdtrackplayer and dvdnavigator io-media graph classes

The **dvdtrackplayer** graph class plays DVD audio tracks and DVD video chapters. The **dvdnavigator** graph class allows the client application to send button commands, select from disk menus, and otherwise interact with DVD player hardware. It requires CSS/CPPM decryption, and it interfaces exclusively to either a DVD drive or a DVD changer.

MMF: the multimedia framework

Many **io-media** graphs are built from filters. The multimedia framework (MMF) is a collection of filters for implementing specific media stream processing functions, such as the parsing or decryption of data. To process a multimedia stream, **io-media** uses only the filters it needs to get the multimedia data, process it, and send it to a target device.

The MMF can be used to create any of:

- a software-only solution running on a single CPU (such as an Intel core)
- a hybrid solution that has a CPU and one or more DSPs (such as a RISC core and TI DSPs)
- a dedicated System-on-a-Chip (SoC) solution

In addition, because the MMF uses the QNX Addon Interfaces library to define its standard interfaces, new components such as codecs written by QNX or by third parties are easily incorporated into the MMF as they become available. In many cases, these additions can be made without recompiling the MMF or the applications that use it.

MMF filters

MMF filters:

- are compiled as either static or dynamic libraries; the dynamic versions of the filters are located by default in **lib/dll/mmedia** (on the target system)
- perform a specific task on multimedia data, such as reading a file from disk, or parsing, decoding or writing data to an audio device
- are plugins, and therefore implement a standard interface that the MMF understands; these interfaces are defined in the QNX Addon Interface library, **libaoi**

Filters are classified by the specific tasks they perform. These classifications are described in the table below.

Classification	Description	Examples
Reader	Reads data from a source.	Audio card readers, data stream readers and video capture card readers.
Parser	Parses a stream of data into component parts.	MPEG bit stream parsers, WAV format parsers and AVI format parsers.
Decoder	Decodes compressed or encoded data.	MPEG audio and video decoders, and DIVX decoders.
Encoder	Encodes raw data into a specific format.	MPEG audio and video encoders.
Writer	Sends data to a destination.	Audio card writers, video output writers and output file writers.

Chapter 2

MME Quickstart Guide

In this chapter. . .

This *Quickstart Guide* is designed to help you get the QNX Aviage Multimedia Suite up and running on a desktop target system, such as an x86 or a VMware workstation running QNX Neutrino. After you've completed the steps below, you should be able to play some WAVE files on the target system.

Note that the MME is the main component of the QNX Aviage Multimedia Core Package, which is part of the QNX Aviage Multimedia Suite. The MME is used for configuration and control of your multimedia applications.

Instructions for installing the MME on other hosts are included in the *MME 1.1.0 Installation Note*, available on the QNX website: **`www.qnx.com/products/`**.

Read through the complete *Quickstart Guide* before starting your installation.

Step 1: Requirements

Your target machine needs to be running one of the following:

- **QNX Neutrino 6.4**
- **QNX Neutrino 6.3.2** with the following patches:
  - 611 (**`umass-enum Patch`**: **`patch-630SP3-0611-umass-enum.tar`**) to use USB mediastores.
  - 684 (Audio Driver Patch for the Intel 82801G (ICH7 Family) High Definition Audio Controller: **`patch-630SP3-0684-Intel-HDA.tar`**) to enable network audio support.

You need the MME installers:

- The MME standard product installer, **`mmedia-core-1.1.0-`***nnnnnnnnnnn***`-nto.sh`**, where *nnnnnnnnnnn* is an 11-digit build number.
- Other installers for additional file formats (such as MP3 and WMA) and external media players (iPod and PFS devices).

If you don't have a QNX Momentics CD, you can download an evaluation version from: **`www.qnx.com/products/`**. This target machine must also support audio output. For a list of supported audio drivers, go to **`www.qnx.com/developers/hardware_support/index.html`** and check the drivers for your platform.

Step 2: Uninstall the Multimedia Suite (upgrades only)

You need to perform this step only if you are upgrading from an older version of the MME. If you are installing the MME for the first time, go to Step 3.

To uninstall the old MME:

1. Log in as **root**.
2. Go to the uninstall directory. For example:

   ```
   # cd $QNX_TARGET/install/mmedia-core/1.0.0/
   ```

3. Confirm that you are in the right directory. All Neutrino uninstallers are called **uninstall.sh**, so if you are in the wrong directory you will uninstall the wrong files.
4. Run the uninstall shell script, **uninstall.sh**:

   ```
   # ./uninstall.sh
   ```

5. Delete the old MME database tables and schema files:

   ```
   # cd /db
   # rm *mme* usb*
   ```

If you do not know where the MME database and schema files are stored, see your **qdb.cfg** file for the location of your database. This file is usually located under **/db** (from the root directory).

CAUTION: You must update all system components to the current QNX Aviage Multimedia Suite 1.1.1 components. Your environment must contain *no* 1.0.*n* or 1.1.0 multimedia components when you begin your installation.

Step 3: Install the MME runtime files

To install the MME runtime files:

1. Log in as **root**.
2. Log into your myQNX account on our website, then go to the Download area.
3. Download the MME installer, **mmedia-core-1.1.0-***nnnnnnnnnn***-nto.sh**, where *nnnnnnnnnn* is an 11-digit build number. The installer is in the form of a shell script.

CAUTION: If you are doing an upgrade, first go to Step 2 and uninstall the old MME.

4. Run **chmod** to make the script executable. For example:

   ```
   # chmod a+x mmedia-core-1.1.0-20071841543-nto.sh
   ```

5 Run the script at the system prompt. For example:

```
# ./mmedia-core-1.1.0-20071841543-nto.sh
```

6 Follow the instructions on your screen.

If you will be installing more than one Multimedia Suite component, you can enter all the required licenses at this time. You can also enter addditional licenses when you install the components that require them.

CAUTION: Any client applications that were compiled against QNX Aviage Multimedia Suite (MME) 1.0.*n* must be recompiled in order to work with QNX Aviage Multimedia Suite (MME) 1.1.0.

Step 4: Copy the SQL schema files

The MME uses an SQL schema to describe the database tables it uses to store media metadata. You should perform this step every time you get a new version of the MME, because the MME runtime is dependent on a specific schema version.

The MME will use the new schema to generate new tables:

1 If you are installing the MME for the first time, you need to create a directory for the database:

```
# mkdir /db
```

2 Copy the new SQL schema and the QDB configuration file:

```
# cd $QNX_TARGET
# cp -c sql/* /db/
```

Step 5: Start `io-audio`

You need to run an audio driver in order to hear music:

1 Check if **`io-audio`** is running:

```
# pidin | grep io-audio
```

2 If **`io-audio`** isn't running, you should start it. For example, if **`audiopci`** is your audio driver:

```
# io-audio -daudiopci
```

3 To verify that audio is working correctly on your machine, you can use the Neutrino **`wave`** utility to play a sample **`.wav`** file. For example, if you have this file on your system, you can use the sample file **`hallelujah.wav`**. Any **`.wav`** file will do, however:

```
# wave /usr/share/audio/hallelujah.wav
```

```
SampleRate = 11025, Channels = 1,
SampleBits = 8 Format Unsigned 8-bit
Frag Size 256
Rate 11025
Mixer Pcm Group [PCM Subchannel]
```

Your machine must support audio output. For a list of audio drivers supported for your platform, see **www.qnx.com/developers/hardware_support/index.html**.

Step 6: Start the temporary filesystem

Now start the temporary filesystem. The MME needs this filesystem running for some of its database tables:

```
# io-fs-media -d tmp -cpages=4 -cbundles=0
```

Step 7: Start the QNX database (qdb)

The QNX database (**qdb**) provides database services to the MME. You must start the database before you start the MME:

```
# qdb -c /db/qdb.cfg -v -Otempstore=/fs/tmpfs -Rset
```

The **-Otempstore=/fs/tmpfs** option is the capital letter "O". It directs the QBD to use temporary files in a memory location instead of on a disk. This configuration reduces possible disk loads generated by the MME's complex SQL statements. The **-Rset** option specifies the QDB database recovery mode.

Step 8: Start mcd

The MME uses the Media Content Detector (MCD) utility to detect mediastore availability. The MCD requires a configuration file. The MME installation includes a sample configuration file, **mcd.conf**, which you can use to get started on systems, such as an x86 or a VMware workstation running QNX Neutrino, that automatically mount CD filesystems.

Start the MCD as follows:

```
# mcd $QNX_TARGET/etc/mcd.conf
```

Step 9: Start io-media

The MME requires **io-media** to decode media. For **dlldir**, pass in the location where you installed the filters used by **io-media**. The default location is:

```
# io-media-generic -M mmf,dlldir=$QNX_TARGET/x86/lib/dll/mmedia
```

Step 10: Give the MME some media to play

By default, the MME searches recursively for media files along pre-configured paths, starting with **/media/drive/**, and including **/fs/cd0** and **/fs/usb0**.

The easiest way to give the MME some media to play is to insert a CD or a USB stick with playable media files into your machine. When you start the MME, it will use the MCD utility to detect the mediastore you inserted and synchronize its library with the mediastore.

If you want to play tracks from your hard drive, you must copy them into the directory **/media/drive/**:

1. Create the directory **/media/drive/** on your target system.
2. Copy some media files into the directory.

On an x86 platform, the MME supports the following media formats: CDDA and WAVE (standard installation), and MP2, MP3, M4A and WMA (additional licences and runtime files required). The MME may parse other media formats but may not be able to play them because it does not currently have decoders for them.

Step 11: Start mme

You need to specify the path to the **mme.conf** configuration file at **/etc/**:

```
# mme-generic -c $QNX_TARGET/etc/mme.conf
```

You can now use the **qdbc** and **mmecli** utilities to query the database, create a track session, and play some music. The MME installer puts **qdbc** in the directory **/usr/bin/**, and **mmecli** in **$QNX_TARGET/target/qnx6/examples/**.

Step 12: Check the synchronization

Before playing some files, it's a good idea to confirm that the MME has detected the files in the mediastore you are using, and synchronized them with its library of media.

Use **qdbc** to query the MME database: point **qdbc** at the MME database and retrieve the file ID (*fid*), mediastore ID (*msid*) and filename (*filename*) for all media files in the library, as shown below:

```
# qdbc -d /dev/qdb/mme "select fid,msid,filename from library;"
Rows:3  Cols: 3
Names:  +fid+msid+filename+
00000: |1|1|05 Come Away With Me.wav|
00001: |2|1|09 I've Got to See You Again.wav|
00002: |3|1|12 Nightingale.wav|
```

The result of the query shown above indicates that the MME sees three WAV files in **/media/drive**, which has the mediastore ID of 1 (msid=1).

Step 13: Play some music

The MME uses a track session to play media. A track session is defined by an SQL statement that selects one or more files from the database. Use the MME command line tool **mmecli** to create and play a track session:

1. Create a track session. Note that the second argument is the letter "l" (el):

```
# mmecli newtrksession l "select fid from library where ftype !=5"
(rc=0,errno=0) new trksessionid=1
```

2. Set the track session. The second argument is the numeral "1" (one). This is the number of the track session you will play.

```
# mmecli settrksession 1
(rc=0,errno=0) trksessionid=1
```

3. Play the track session.

```
# mmecli play
(rc=0,errno=0) Playing from track session fid/bid = 0
```

You should now hear some music.

- The SQL statement used to create a track session:
  - must include (at least) the *fid* of a file to play
  - includes **ftype!=5** to exclude entries for mediastores from the track session
  - does *not* require a final semicolon character
- The standard MME installation will play CDDA and WAV files only. The MME will synchronize other media file types, but will report an error if you try to play unsupported file types (such as MP3 and WMA files) before installing the runtime files for these codecs. For instructions, see "Playing additional formats" below.

mmecli commands

Some things to note about the **mmecli** commands:

- Run **use mmecli** to see a complete list of the commands you can execute on the MME. For more detailed information, refer to the chapter MME Utilities Reference in the *MME Developer's Guide*.
- The **mmecli** utility has an interface that uses CURSES and displays extra information, such as events, volume, balance, repeat and random modes, and progress with synchronization, playback and other operations. To use this interface, run **mmecli** with no options:

  ```
  # mmecli
  ```

  Use the same **mmecli** commands as with the **mmecli** command-line interface. To exit, type **quit**.

Creating a startup script

To simplify MME startup, you can write a simple startup script that performs all the startup steps:

```
slay -fv io-fs-media qdb io-media-generic mme-generic mcd
io-fs-media -d tmp -cpages=4 -cbundles=0
waitfor /fs/tmpfs
qdb -c /db/qdb.cfg -v -Otempstore=/fs/tmpfs -Rset
waitfor /dev/qdb
mcd $QNX_TARGET/etc/mcd.conf
io-media-generic -M mmf,dlldir=$QNX_TARGET/x86/lib/dll/mmedia
waitfor /dev/io-media/graphs
mme-generic -c $QNX_TARGET/etc/mme.conf
waitfor /dev/mme/default
```

The startup script above starts the modules required for a simple system. For a more comprehensive system with, for example, the capability to manage and play media from iPod and PFS devices, you would need to add instructions to start the drivers for these devices.

Adding more media files

If you want to add more media files to your system after you've started the MME, you'll need to force it to resynchronize its database.

1. Add the files to the system.
2. Get the mediastore ID (*msid*) of the hard drive:

   ```
   # qdbc -d/dev/qdb/mme "select msid,mountpath from mediastores;"

   Rows: 2  Cols: 2
   Names: +msid+mountpath+
   00000:|1|/media/drive|
   00001:|2|/fs/cd0|
   ```

   The second line returned by the QDB lists the table column headings. Thus, the *msid* for the CD at *mountpath* **/fs/cd0** is 2.
3. Instruct the MME to resynchronize the mediastore where you added the files:

   # mmecli resync_mediastore *msid* **0 7**

   To resynchronize files on your hard drive at **/media/drive**, use its *msid*, which is 1 (one). Setting the third argument to 0 (zero) tells the MME that no folder is specified, and therefore synchronize the entire mediastore. Setting the last argument to 7 tells the MME to synchronize all information and metadata.
4. Create and set a new track session, and play the media files, as described in Step 13.

Playing additional file formats

If you want to play file formats (such as MP3 or WMA) not supported by the MME standard installation, you need to install the appropriate runtime files:

1. Stop **io-media**.
2. Follow the instructions in Step 3 for the following installers, as required:
   - AAC — **mmedia-aac-1.1.0-***nnnnnnnnnn***-nto.sh**
   - MP2 and MP3 — **mmedia-mp3-1.1.0-***nnnnnnnnnn***-nto.sh**
   - WMA — **mmedia-wma9-1.1.0-***nnnnnnnnnn***-nto.sh**
3. Start **io-media**.

Using external media players

If you want use an external media player, such as an iPod or a PFS device, you need to:

1. Install the runtime files that support these devices.
2. Use **iofs-ipod** or **iofs-pfs**, depending on the type of media player.

The arguments for **io-usb** may be different for embedded targets.

iPods

You can connect to newer iPods (Generation 5 and more recent) either through a serial connection or, with an Apple authentication chip, through a USB connection. Older iPods (Generation 4 and older) support only the serial connection.

Before designing your client application, you should contact Apple to obtain:

- the required authentication IC chip and associated licenses
- for serial connections, the specifications for cables supporting the serial protocol

For more detailed information about connecting to iPods, see the *MME Developer's Guide* chapter Working with iPods.

PlaysForSure devices

To set up the MME to play media from PFS devices:

1. Follow the instructions in Step 3 for the following installers::
   - **pfs-1.1.0-***nnnnnnnnnn***-nto.sh**
   - **wmdrm10-nd-1.1.0-***nnnnnnnnnn***-nto.sh**
2. If **io-usb** isn't already running, start it:

```
# io-usb -duhci -dohci -dehci
```

3. Start the **io-fs** module for PFS devices:

```
# io-fs-media -dpfs
```

4. If the MME isn't running, follow the instructions in Step 11 to start it.

When you have finished playing media from an iPod or PFS device, you do not need to disconnect it from the MME. Just physically remove it from the system.

Using USB mediastores

If you want to synchronize USB devices, you need to enable the MME to discover these devices:

1. If **io-usb** isn't already running, start it:

```
# io-usb -duhci -dohci -dehci
```

2. If the QDB isn't running, follow the instructions in Step 7 to start the database.
3. If you have installed the MME on a target running a QNX Neutrino 6.4.0 release or later, the **enum-usb** utility enumerates the devices on a USB bus.

   If you have installed the MME on a target running a QNX Neutrino 6.3.*n* release, start the **umass-enum** utility:

```
# umass-enum -a -d /dev/ -h umass,umasscd
```

- If you have installed the MME on a target running a QNX Neutrino 6.3.*n* release, to use USB mediastores you must have installed **umass-enum** patch **patch-630SP3-0611-umass-enum.tar**. See Step 1 above.
- If you are using the MME installed on VMware, USB doesn't seem to work with all laptops (mainly IBM/Lenovo). The fix is to edit the VMware Configuration File (***.vmx**) to add the line: **uhci.newCore = "FALSE"**
- USB devices are mounted as read/write devices. To avoid issues such as marking files as busy, or flagging the partition incorrectly, you should unmount a USB device when you have finished with it: **# umount /fs/usb0**

Sample applications

Finally, you may want to look at and modify some sample applications:

- Just play some tracks: **mme-shuffle**.
- Simple media player, with unlimited SQL requests: **mme-player-simple**.

- Media copy and rip, and play: **mme-cdripper**.
- Enable selection and playback of media from different sources: **mme-player-multisource**.

You can also use **mmexplore** to explore or browse mediastores that might not already be synchronized; it synchronizes only what is needed to see the mediastore's contents.

Troubleshooting

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

- Check what's running on your target:

  ```
  # pidin arg
  ```

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

  ```
  # io-media-generic  -M mmf,dlldir=$QNX_TARGET/x86/lib/dll/mmedia -DD
  ```

  To start **mme** in verbose mode:

  ```
  # mme-generic -c $QNX_TARGET/etc/mme.conf -vvv
  ```

- Confirm that the MME has detected your mediastore in its filesystem mountpath: **# ls /fs**

  If the MME does not detect your mediastore, edit the MME's MCD configuration file with the information the MCD needs to find the mediastore. See "Mediastore detection path configuration" in the *MME Configuration Guide* for more information.

If you need to contact us, the information you gather will help us solve your problem.

Chapter 3

MME Frequently Asked Questions (FAQs)

These FAQs (Frequently Asked Questions) present short answers to questions client application developers often ask about the MME. For more detailed information, see the relevant sections of the *MME Developer's Guide*.

The MME doesn't find the mediastores

Question

I have configured the MCD to detect mediastores, but the MME doesn't find these mediastores. What is wrong with my configuration?

Answer

To detect mediastores, your system must have the MCD *and* the **`slots`** table configured correctly.

The MCD (Media Content Detector) 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 a configuration file. The default file is **`$QNX_TARGET/etc/mcd.conf`**, but your system may be using another file.

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.

```
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.

The MME can't tell the difference between two USB sticks

Question

I have two different USB sticks, and the MME doesn't recognize that they are different mediastores. This looks like a bug.

Answer

This is not a bug. It is expected behavior for USB sticks that are exactly the same size and that do not have unique identifiers. Many USB sticks do not have at least one of a **WMPInfo.xml** file, a volume name or a unique serial number. Without one of these unique identifiers, the MME has no mechanism for distinguishing between two USB sticks of the same size.

To solve the problem in a development environment, you can either assign a unique volume name to each USB stick, or synchronize them with Windows Media Player, which automatically creates a **WMPinfo.xml** files.

In a production environment, you can have the HMI write a volume name or other unique identifier to USB sticks the first time they are inserted.

The MME doesn't synchronize all files on a CD

Question

I have a CD with MP3 data files, and audio tracks. The MME only synchronizes the data files, and ignores the audio tracks..

Answer

This behavior is the expected behavior for the MME 1.0.0. The MME uses the first entry in a CD's table of contents (TOC) to determine if the CD is an audio or a data CD, and makes only one entry in the mediastores table for the CD.

This behavior means that the MME presents a CD with both audio and data files to the client application as either an audio CD or a data CD, based on the type of file in its first TOC entry. If the CD is presented as a data CD, the client application can access only the CD's data files; if the CD is presented as an audio CD, the client application can access only the CD's audio tracks.

See also "CD detection and presentation" in the chapter Working with Mediastores of the *MME Developer's Guide*

Synchronizing unsupported file formats

Question

The MME spends a lot of time synchronizing files in formats that my implmentation does not support. How can I speed things up?

Answer

The default setting for the MME is to synchronize all mediastores and files. However, you can configure the MME to *not* synchronize:

- specified mediastores — use the **`<nosync>`** configuration element to specify the path of the mediastore to skip.
- specified file types — use the **`<extensions>`** to specify the file types to skip during synchronization.
- individual files, based on a specified string in the file name — use the **`<SyncFileMask>`** configuration element to configure MME synchronization to skip files based on a specific character string in the file name (including the file extension).

For more information, see the chapter Configuring Media Synchronizations in the *MME Configuration Guide*.

The MME can also be configured to *not* synchronize specified mediastores, by specifiying the paths to these mediastores in the configuration element.

Delay at end of synchronization

Question

The MME performs well through entire synchronizations, but sometimes at the end there is a delay of several seconds before it sends the MME_EVENT_MS_SYNCCOMPLETE event to inform me that it is done. Is this a bug?

Answer

This delay is expected behavior. When the MME synchronizes prunable mediastores that it has synchronized earlier, it may clean up unused metadata in its database. This clean up may take up to several seconds, depending on the size of the MME database, and cause a corresponding delay between delivery of the MME_EVENT_MS_*PASSCOMPLETE event and delivery of the MME_EVENT_MS_SYNCCOMPLETE event. For more information, see "Database clean up during synchronization" in the chapter Synchronizing Media of the *MME Developer's Guide*.

No audio from playback

Question

The MME appears to be playing a media file correctly, but no output is audible.

Answer

The default MME configuration file (**`mme_data.sql`**) may specify output to a device that does not work for your board.

Check that the output location specified in your MME configuration file points to the device with the speakers from which you expect the audio output. If the specifed output location does not point to the device with the speakers, do one of the following:

- Connect the speakers to the device specifed by the MME configuration file.

  or:

- Change the MME configuration:

  1. Stop the MME.
  2. Change the MME configuration file to specify the correct output device. For example:

     ```
     INSERT INTO outputdevices( type, permanent, name, devicepath )
           VALUES( 1, 1, 'defaultoutput', '/dev/snd/correct_output_device' );
     ```

  3. Remove any existing MME database files.
  4. Restart the MME.

For more information, see the chapter Control Contexts, Zones and Output Devices in the *MME Developer's Guide*.

Random mode doesn't work on a CD changer

Question

Random mode works for different mediastores, but it doesn't work for CDs.

Answer

Not all external CD changers support random and repeat modes. If random mode works for other mediastores, then you are probably using these modes correctly. The most likely answer is that your exteranl CD changer does not support random mode. Check the specifications for the CD changer. For more information, see the *MME Developer's Guide*: "Using random and repeat modes" in the chapter Playing Media, and *mme_setrandom()* and *mme_setrepeat()* in the chapter MME API.

The MME keeps trying to play a bad track

Question

When it encounters a damaged track on a CD, the MME keeps trying to play the track. How can I make it just move on to the next track?

Answer

The default MME configuration for handling damaged tracks is to skip forward 200 milliseconds and retry playback 10 times, doubling the distance of the skip forward at each retry. You can change this configuration to suit your environment.

The MME database fills up with unused track sessions

Question

The MME's **`trksessions`** table keeps growing with unused track sessions. What can I do about this?

Answer

The MME does not automatically remove track sessions when you remove the mediastores with the tracks for these track sessions. When you prune a mediastore from your database, you should call the function *mme_rmtrksession()* to delete from your database all track sessions that use tracks on the pruned mediastore. For more information, see "Deleting track sessions" in the chapter Playing Media of the *MME Developer's Guide*.

Problems with iPod synchronization

Question

The MME does not synchronize iPods when they are plugged in, but it synchronizes all other mediastores.

Answer

The size and design of iPods can result in long waits while the devices are being synchronized. Therefore, the MME is designed to automatically synchronize all mediastores it detects *except* iPods. When your client application detects an iPod device, it should initiate a directed synchronization on the iPod folders the user wants to access. To minimize the time required to deliver metadata to the end user, when it is synchronizing iPods, the MME updates the *title* field in the library, giving users enough metadata to select a track and start playback.

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

The MME doesn't display correct file IDs when playing iPods

Question

Why does the MME not change the fid when it is playing music on an iPod?

Answer

iPods manage their own track sessions and do not report the file IDs of their track sessions to the MME. See "Getting track information when playing media on iPods" in the *MME Developer's Guide* chapter Working with iPods.

PFS device returns "Not supported 101B" error

Question

When I try to access a PFS device, I get a **`WMPinfo.xml`** file with the following: **`<NotSupported>`**101b**`</NotSupported>`**. Is there a bug in the PFS device driver?

Answer

The **`Not supported 101B`** means that you are trying to access a PFS device that does not support the **`GetPartialObject`** MTP command without having specified that you want this support when you started **`io-fs-media`**.

The default configuration for **`io-fs-media`** is to *not* support PFS devices that don't support the **`GetPartialObject`** MTP command. To start **`io-fs-media`** to support PFS devices that don't support the **`GetPartialObject`** MTP command, you must specify the **`getsize`** option and the buffer size when you start **`io-fs-media`**.

For more information, see:

- "Devices that don't support **`GetPartialObject`**" in the *MME Developer's Guide* chapter Working with PFS Devices

- the *MME Technotes* chapter User-specified MTP Commands to PFS Devices
- **iofs-pfs.so** in the *MME Utilities Guide*

io-media loads DLLs it doesn't need

Question

io-media loads all the DLLs in the DLL directory, not just the ones it requires.

Answer

This is the expected behavior. **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.

To ensure system efficiency, 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.

For more information about **io-media** DLLs, see "Where to install **io-media** DLLs" in the *MME Utilities Reference*.

Why is there jitter in the playback time positions reported by the MME?

Question

The MME appears to deliver inaccurate time positions during playback. The times reported show jitter, as though the playback speed is speeding up or slowing down. What can I do to correct this?

Answer

This behavior is expected; there is nothing to correct. The accuracy and frequency of time updates depends upon the implementation of the **io-media** graphs used to process the media, and on the accuracy and frequency of updates delivered by the underlying drivers and hardware. Graphs should attempt to deliver a timing resolution of 100 milliseconds or better, but this resolution is not always available.

The MME delivers the MME_EVENT_TIME event to the client application only when it receives a time update from the device or driver (through **io-media**). Thus, if, for example, the MME's notification interval to the client application is set to 100 milliseconds, but a driver delivers time position updates to the MME only every 300 milliseconds, the client application will only receive time updates every 300 milliseconds and may see jitter in the time reporting.

Trick play fails on a high bitrate encoded file

Question

The MME successfully plays a high bitrate encoded file, but is unable to fast forward or reverse on the file.

Answer

This behavior may indicate incorrect bitrate information in the file header, or some other problem with the file header. Depending on the nature of the problem with the header, different media players may or may not be able to play the file.

iPod resets frequently

Question

iPods are resetting frequently, for no apparent reason.

Answer

iPods have a tendency to reset if you do not use the **`-c`** option when you start **`io-usb`**.

Change your startup to start **`io-usb`** with the **`-c`** option. With this option selected, the launcher application selects the device configuration to use, instead of just using the device's first configuration.

Index

G

H

I

J

L

M

O

P

Q

R

S

T

U