Multimedia Architecture Guide

QNX

©2014–2015, QNX Software Systems Limited, a subsidiary of BlackBerry Limited. All rights reserved.

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

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

QNX, QNX CAR, Momentics, Neutrino, and Aviage are trademarks of BlackBerry Limited, which are registered and/or used in certain jurisdictions, and used under license by QNX Software Systems Limited. All other trademarks belong to their respective owners.

Electronic edition published: March 31, 2015

Contents

About This Guide 5
Typographical conventions 6
Technical support 8

Chapter 1: Multimedia Architecture 9

Chapter 2: Device detection and information retrieval 13

Chapter 3: Metadata synchronization and retrieval 15

Chapter 4: Media playback 17

Index 19

About This Guide

The *Multimedia Architecture Guide* provides an overview of the multimedia components in the QNX SDK for Apps and Media and describes how they work together.

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

To find out about:	Go to:
The layers in the multimedia architecture and the communication between components	*Multimedia Architecture* (p. 9)
The components used to detect media devices	*Device detection and information retrieval* (p. 13)
The components used to upload media metadata	*Metadata synchronization and retrieval* (p. 15)
The components used to play media	*Media playback* (p. 17)

Copyright © 2015, QNX Software Systems Limited

Typographical conventions

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

The following table summarizes our conventions:

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

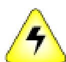
We use an arrow in directions for accessing menu items, like this:

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

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

Notes point out something important or useful.

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



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

Copyright © 2015, QNX Software Systems Limited

Note to Windows users

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

Copyright © 2015, QNX Software Systems Limited

Technical support

Technical assistance is available for all supported products.

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

Copyright © 2015, QNX Software Systems Limited

Chapter 1
Multimedia Architecture

The QNX SDK for Apps and Media uses several resource managers, services, and libraries to perform the multimedia tasks of detecting mediastores, synchronizing media metadata with databases, and playing audio and video files.

These components form part of a robust and versatile platform that supports all types of media applications. The organization of these components looks like this:

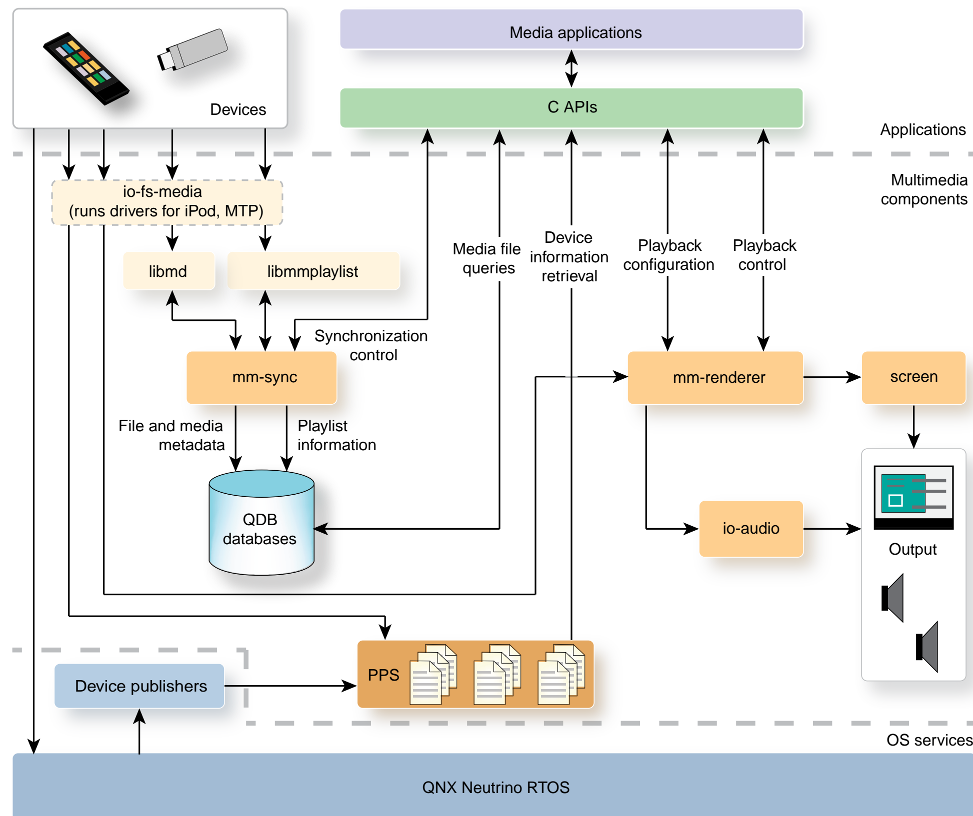


Figure 1: Multimedia Architecture

Communication

As *Figure 1* (p. 9) shows, communication between media applications, multimedia components, and OS services is done with C APIs, QDB databases, and Persistent Publish/Subscribe (PPS) objects.

The multimedia synchronizer service, `mm-sync`, and the multimedia rendering component, `mm-renderer`, expose C APIs to client applications for starting and monitoring media operations.

The QDB databases store metadata describing the media content found on devices. Querying these embedded databases to retrieve metadata is much faster than reading it from files on physically separate

devices. Also, this design means that the metadata is kept in persistent storage, so applications can read the metadata of files stored on devices no longer attached to the system.

The PPS objects store information describing mediastores (devices) currently attached to the system. This information is published by *device publishers* and it includes hardware connectivity details, filesystem mountpoints, and state information. Because of this last type of information, the contents of some of these objects are dynamic.

For MTP devices, additional PPS objects are published by the MTP driver. These objects contain details about the device's software and vendor, and the last modification time of its media content. The *PPS Objects Reference* describes the MTP-related PPS objects. The MTP driver runs in the `io-fs-media` process and provides a POSIX filesystem interface for accessing the device.

For iPod devices, the iPod driver publishes several PPS objects. These objects store information on playback status, Bluetooth profiles, and more, and are explained in detail in the iPod documentation included with the QNX Apps and Media Interface for Apple iPod. Depending on the protocol in use, the iPod driver may run in `io-fs-media` or another process.

Devices with POSIX filesystems (e.g., USB sticks) don't need a driver running in `io-fs-media`; this service is used only for non-POSIX devices to provide a common mechanism to access media content. This means that for any device type, your applications can use POSIX system calls to read information from PPS, which is necessary for discovering media content and reacting to content updates (e.g., new media files after pictures are taken by the device operator).

Applications

Your media applications can read device information from PPS objects and interact with `mm-sync`, `mm-renderer`, and QDB at any time and in any order because these services are independent of one another. For instance, an application could synchronize some or all of a device's metadata with `mm-sync`, query the device's QDB database for specific track information to make it visible to users, and then, in response to user requests, begin playing certain tracks with `mm-renderer`. Or, it could skip the synchronization and begin playing the first audio track found on a newly attached device by invoking `mm-renderer`.

It's up to developers to implement the application front-end, whether it's an interactive HMI component or a command interface, and the logic that invokes the multimedia services to carry out media tasks. The platform ships with several sample utilities that allow you to test various multimedia services from the command line, without having to learn their APIs:

`mmcli`

Tests the APIs of multimedia components by forwarding commands to a loaded library or service.

`mmrplay`

Plays or records media through `mm-renderer`, based on command-line options.

`mmsyncclient`

Forwards media synchronization commands to `mm-sync` and reports synchronization status.

The source code for `mmrplay` and `mmsyncclient` is included in the platform's *source code samples package*, which is separate from the installers that set up the host system but is available at the same download location. The package also includes the source code of the multimedia plug-and-play utility,

 Copyright © 2015, QNX Software Systems Limited

mm-pnp, which is a demo program that provides a walkthrough of the API call sequences that detect when the user attaches a mediastore and then access, extract, and play its content.

The *Multimedia Test Utilities Guide* explains the purpose of mmcli, mmrplay, and mm-pnp, how to start these utilities with command lines, and how to configure and use them. The *Multimedia Synchronizer Developer's Guide* provides usage instructions for mmsyncclient.

Multimedia components

The multimedia components work together to perform three main tasks:

- Detecting devices and retrieving their information
- Synchronizing metadata to QDB databases
- Playing audio and video files

The other sections in this guide explain the order of interaction and the information flow between the components to carry out each of these tasks.

The QNX Apps and Media reference images come with these multimedia components:

Name	Description	Path(s)
Device information objects in PPS	Store attributes describing device connectivity, driver processes, and mountpoints of device filesystems.	**/pps/qnx/device/***, **/pps/qnx/driver/***, **/pps/qnx/mount/***
mm-sync	Synchronizes metadata from tracks and playlists on media devices into QDB databases. Metadata includes creation and runtime information for files and playlists.	**/base/usr/sbin/mm-sync**
QDB	Manages embedded databases that store metadata read from media devices.	**/base/usr/sbin/qdb**
libmd	Reads metadata fields from files on media devices. This component library is used by mm-sync but can be linked into and called from an application.	**/base/usr/lib/libmd.so**
libmmplaylist	Retrieves playlist entries, which are track URLs referenced in playlist files. This component library is used by mm-sync but can be linked into and called from an application.	**/base/usr/lib/libmmplaylist.so**
mm-renderer	Plays audio and video tracks, and reports playback state. You can play multiple items concurrently but independently.	**/base/usr/sbin/mm-renderer**
screen	Renders video output to the display. This service is used by mm-renderer but it can be used directly by applications to manipulate the video output window.	**/base/usr/lib/libscreen.so**
io-audio	Starts audio drivers to enable outputting of audio streams through hardware. This service is used by mm-renderer and shouldn't be used directly by media applications.	**/proc/boot/io-audio**
io-fs-media	Runs drivers that provide a POSIX filesystem interface to media devices. For some device types, other multimedia components use this service to read file information and media streams.	**/base/usr/sbin/io-fs-media**

Copyright © 2015, QNX Software Systems Limited

OS services

The OS layer includes *device publishers*. When users attach devices, device publishers create PPS objects and write device information into them. The publishers remove the objects that store information about specific devices when users remove those devices. Your media applications can monitor the entries of the PPS directories that store these objects and then read the object contents to discover new media sources and to learn the mediastore mountpoints, which they can explore for playable content.

The *Device Publishers Developer's Guide* explains the types of PPS objects that store device information, the directories in which these objects are published, and the included publisher services and the device types that they support.

Copyright © 2015, QNX Software Systems Limited

Chapter 2
Device detection and information retrieval

The device publishers update device information in PPS objects when users attach or detach devices. Your applications can read this information and use it to access media content and decide what to synchronize and play.

The interaction between these components proceeds as shown here:

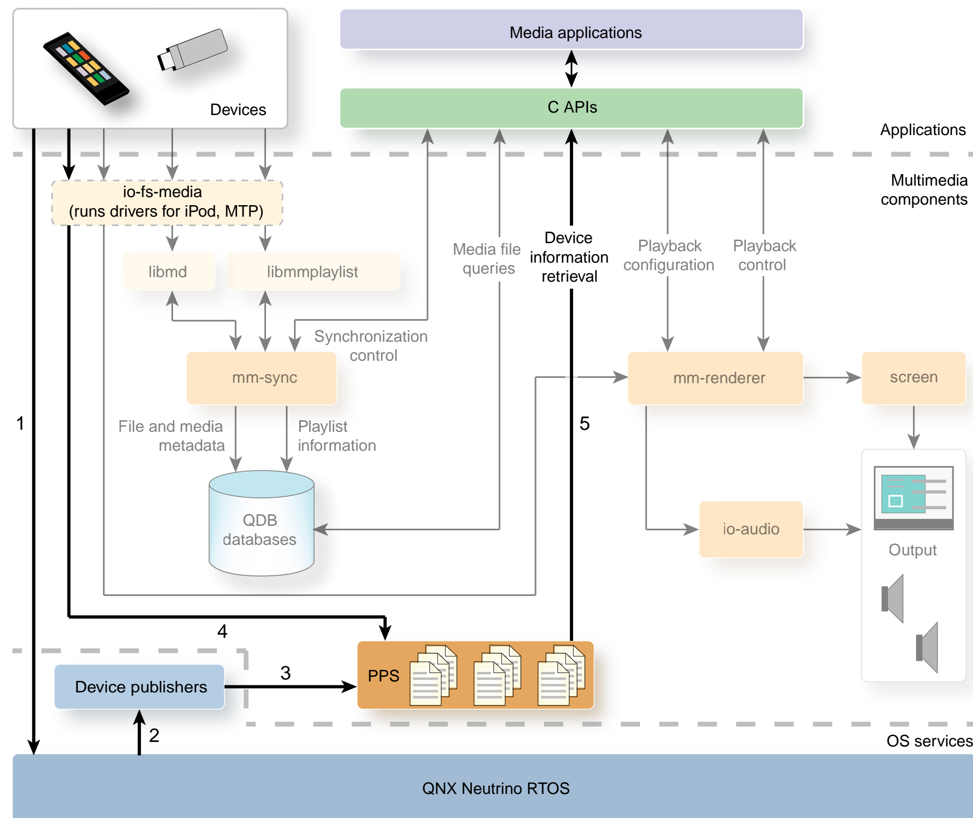


Figure 2: Device Monitoring and Information Retrieval

1. **Detecting device attachments**

   Device publishers don't physically detect when users attach or detach devices. Other OS-layer processes—device drivers and protocol stacks—monitor I/O hardware for physical state changes that indicate device attachments or detachments (e.g., SD card insertions or USB device connections). Because they interface with hardware, the drivers and stacks contain up-to-date details on the physical connectivity and filesystem mountpoints of attached devices. The publishers must communicate with these system processes to learn of device attachments and detachments.

2. **Obtaining device information**

   Different publishers use different methods for obtaining the latest device information. The `usblauncher` publisher queries the USB stack (`io-usb`) process for device information (for

details, see “The usblauncher Service” in the *Device Publishers Programmer's Guide*). The `mmcsdpub` publisher monitors specific **/dev** paths and when it notices new or updated entries, it communicates with the drivers to obtain device information (for details, see “Role of device drivers and `mcd`”).

3. **Publishing device, driver, and filesystem information to PPS**

   After retrieving information about newly attached devices from other OS processes, the publishers output this information in text format to PPS objects. Each object stores information that describes a single device. Also, the publishers use different object types for storing device connectivity, driver process, and filesystem information. For more details, see “PPS object types”.

   When publishers learn from a driver or protocol stack process that a device has been detached, they delete the PPS objects related to that device.

4. **Publishing additional device information to PPS**

   For MTP and iPod devices, additional PPS objects are published by the driver. The MTP driver publishes the software version, manufacturer name, vendor name, and the last time that media content was modified on the device (for details, see the MTP-related PPS objects in the *PPS Objects Reference*). The modification time field is dynamic because the driver updates it whenever media content is added, modified, or deleted.

   With iPods, the extra information written in PPS depends on which Apple protocol the device runs; for details, refer to the iPod documentation included with the QNX Apps and Media Interface for Apple iPod.

5. **Detecting devices in media applications**

   Before your applications can synchronize or play any media, they must learn which devices (mediastores) are attached to your system. Your applications must monitor the device-related PPS objects to readily receive information about newly attached devices. This information includes the mountpoints, which your applications can use to explore the relevant filesystem locations to identify and access media tracks.

   For example, when a USB device is inserted, its default mountpoint is **/fs/usb0**. The publisher that monitors USB device attachments and removals (`usblauncher`) writes this mountpoint information to a PPS object in **/pps/qnx/mount/**.

   Other information fields can help you enforce media policies. For instance, suppose that you want to filter playback based on the mediastore type. By examining the `media_type` attribute, your application can choose to play tracks on some device types (say, audio CDs) but to ignore tracks on other device types (say, DVDs).

   For more details on detecting mediastores from an application, see the first two steps of the process described in the “Synchronizing media content from applications” section in the *Multimedia Synchronizer Developer's Guide*.

Copyright © 2015, QNX Software Systems Limited

Chapter 3
Metadata synchronization and retrieval

Media applications invoke mm-sync to synchronize metadata on mediastores with QDB databases. The mm-sync service uses dedicated libraries to extract track and playlist metadata from mediastores and then stores this information in databases. Applications can later query these databases to retrieve the metadata.

The interaction between these components proceeds as shown here:

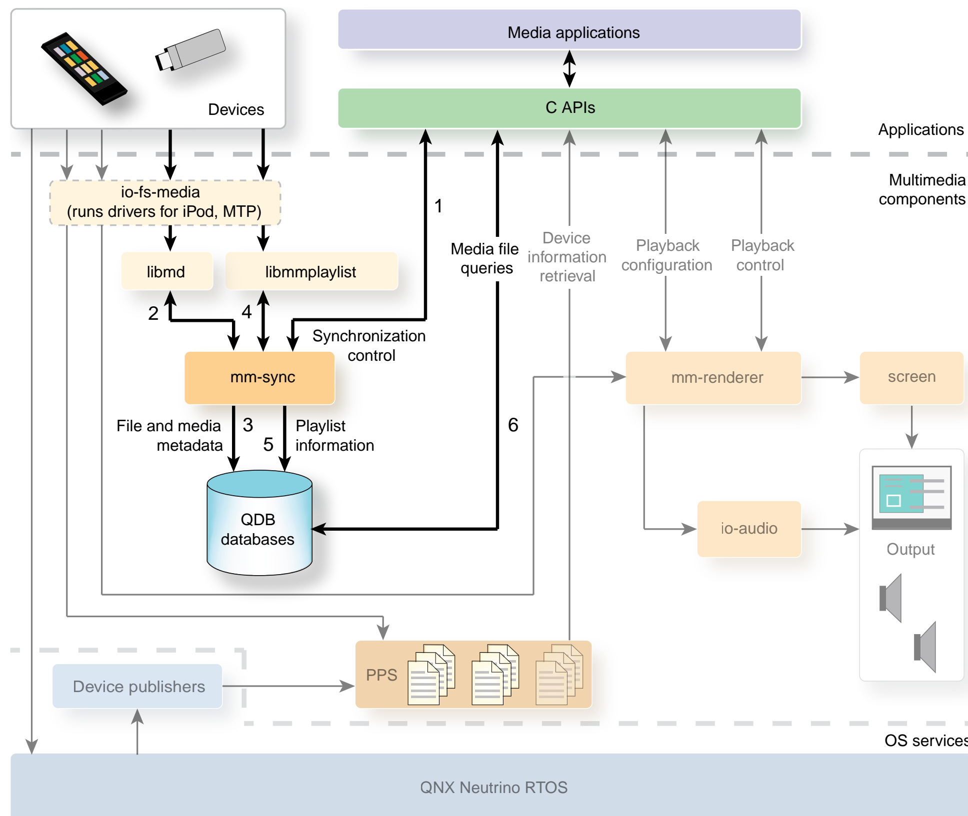


Figure 3: Metadata Synchronization and Retrieval

1. **Starting a synchronization**

   After learning through PPS about which devices (mediastores) are attached to the system and then exploring their contents, applications can invoke mm-sync to upload metadata from those mediastores into QDB databases. Metadata includes creation and playback information such as the album name, artist name, track title, and track duration. The command to start a synchronization must contain the mediastore path of the files to synchronize (e.g, **/tunes/queen/**) as well as the device entry for the QDB database (e.g, **/dev/qdb/cd0_db**) that will store the metadata.

   An application must create and load the appropriate QDB database before it starts a synchronization. We recommend using the device's unique ID in the database name; for information on how to do

this, refer to the “Maintaining database persistence” section in the *Multimedia Synchronizer Developer's Guide*.

2. **Extracting file and media metadata**

   The `mm-sync` service uses the `libmd` library to read metadata fields (types) from mediastore files. The files named in the metadata requests sent to `libmd` are always located within the mediastore path given to `mm-sync`. For information on `libmd`, see the *Metadata Provider Library Reference*.

3. **Storing file and media metadata**

   The `mm-sync` service must store the extracted metadata in the database tables and fields (columns) specified in the configuration. The `mm-sync` configuration file defines the mapping of metadata fields to database fields for audio, video, and photo files. You can modify this mapping to change which metadata fields get uploaded to QDB databases and which database tables and fields the various metadata fields get stored in. For information on how to do this, see the **<Configuration>/<Database>/<Synchronization>/<ConfigurableMetadata>** element in the `mm-sync` configuration description.

4. **Extracting playlist entries**

   After the media metadata has been uploaded, `mm-sync` extracts the entries for all playlists found within the path specified at the start of synchronization. Playlist entries, which are track URLs referenced in playlist files, are retrieved by using the `libmmplaylist` library. For information on `libmmplaylist`, see the *Multimedia Playlist Library Reference*.

5. **Storing playlist entries**

   The `mm-sync` service then uploads the playlist entries to the device's database. This action fills in empty fields in the tables that store playlist information. In the `mm-sync` configuration file, you can define filename pattern matches and even an alternative configuration for `libmmplaylist` for greater control over what playlist entries get synchronized. For information on how to do this, refer to the **<Configuration>/<Database>/<Synchronization>/<PLSS>** element in the `mm-sync` configuration description.

6. **Retrieving synchronized metadata**

   Media applications can issue SQL queries to a database through the QDB API to retrieve up-to-date track and playlist information. This information lets you show details of the currently playing track and the tracks in the playlist window, support browsing of mediastore files and directories, and display album artwork. To know when the synchronization of a mediastore has completed, meaning the tables in the device's database are as accurate as possible, your applications must monitor `mm-sync` events and wait for the MMSYNC_EVENT_MS_SYNCCOMPLETE event.

Copyright © 2015, QNX Software Systems Limited

Chapter 4
Media playback

Media applications use `mm-renderer` to play audio and video tracks. The applications attach media files or playlists as the `mm-renderer` input and attach one or more hardware devices as the outputs. During playback, `mm-renderer` manages the media flow between the input and the outputs.

The interaction between these components proceeds as shown here:

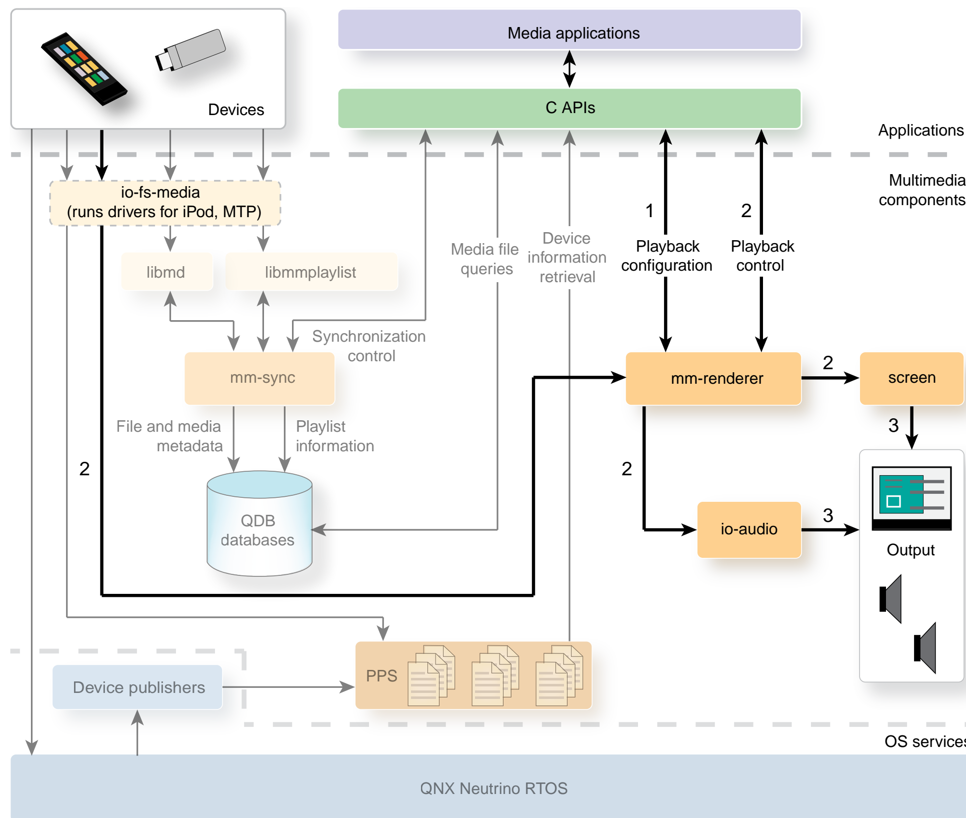


Figure 4: Media Playback

1. **Configuring the rendering service**

   To play media content, an application must configure the `mm-renderer` service by defining a context and then attaching an input and one or more outputs to that context. For the input, the application must provide the URL of a track or playlist stored on an accessible mediastore. For the output, it must provide a URL that names an output device and lists the device configuration options. For an overview of the API calls required to set up playback, see the "Playing media" section in the *Multimedia Renderer Developer's Guide*.

2. **Controlling playback**

   When an application issues the command to start playback, `mm-renderer` initiates the media flow between the input and the outputs. Note that the rendering service doesn't parse the media

files itself but instead uses lower-level mechanisms (e.g., HTTP streamers, file readers) to read and forward their contents, which it then directs to other utilities that send the audio and video components to the appropriate drivers. The main purpose of `mm-renderer` during playback is to process playback commands. These commands allow applications to change the playback speed, skip to a new track position, and stop playback.

3. **Outputting audio and video**

   Plugins within `mm-renderer` communicate (through intermediate libraries, which aren't shown) with the Screen Graphics Subsystem for outputting video and with the `io-audio` utility for outputting audio. Screen is the windowing system that `mm-renderer` uses to render video to the display. The `io-audio` utility is a resource manager that dynamically loads and configures audio drivers; `mm-renderer` uses it to deliver audio to the appropriate output devices (e.g., speakers).

Copyright © 2015, QNX Software Systems Limited

Index

D

device monitoring and information retrieval process *13*
devices *13*
    monitoring *13*
    retrieving information on *13*

M

media *17*
    playing *17*
media playback process *17*
metadata *15*
    retrieving *15*
    synchronizing *15*
metadata synchronization and retrieval process *15*
multimedia *9–12*
    application requirements *10*
    architecture *9*
    communication between components *9*
    components *11*
    OS services for detecting media devices *12*
    sample utilities *10*

T

Technical support *8*
Typographical conventions *6*

U

USB *14*
    default mountpoint *14*

Copyright © 2015, QNX Software Systems Limited

Copyright © 2015, QNX Software Systems Limited