QNX CAR Multimedia Architecture Guide

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

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

Electronic edition published: Thursday, April 17, 2014

Table of Contents

About This Guide

The *QNX CAR Multimedia Architecture Guide* provides an overview of the multimedia components in the QNX CAR Platform for Infotainment 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	*QNX CAR Multimedia Architecture* (p. 9)
The components used to detect mediastores and synchronize their metadata	*Media device detection and synchronization* (p. 12)
The components used to browse media	*Media browsing* (p. 14)
The components used to play media	*Media playback* (p. 16)

Typographical conventions

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

The following table summarizes our conventions:

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

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

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

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

Notes point out something important or useful.

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

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

Note to Windows users

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

Technical support

Technical assistance is available for all supported products.

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

Chapter 1
QNX CAR Multimedia Architecture

The QNX CAR Platform brings together many different components to support the media tasks of synchronizing metadata with databases, browsing mediastore contents, and playing audio and video files.

To carry out user-requested media actions, the Media Player app in the HMI sends commands to the `mm-player` server. This program uses several components, shipped with either the CAR platform or the QNX SDK for Apps and Media to access media filesystems and to browse and play media files. In addition, `mm-player` synchronizes the media information on devices with the contents of databases managed by the QNX Database (QDB) server. The information flow between these components looks like this:

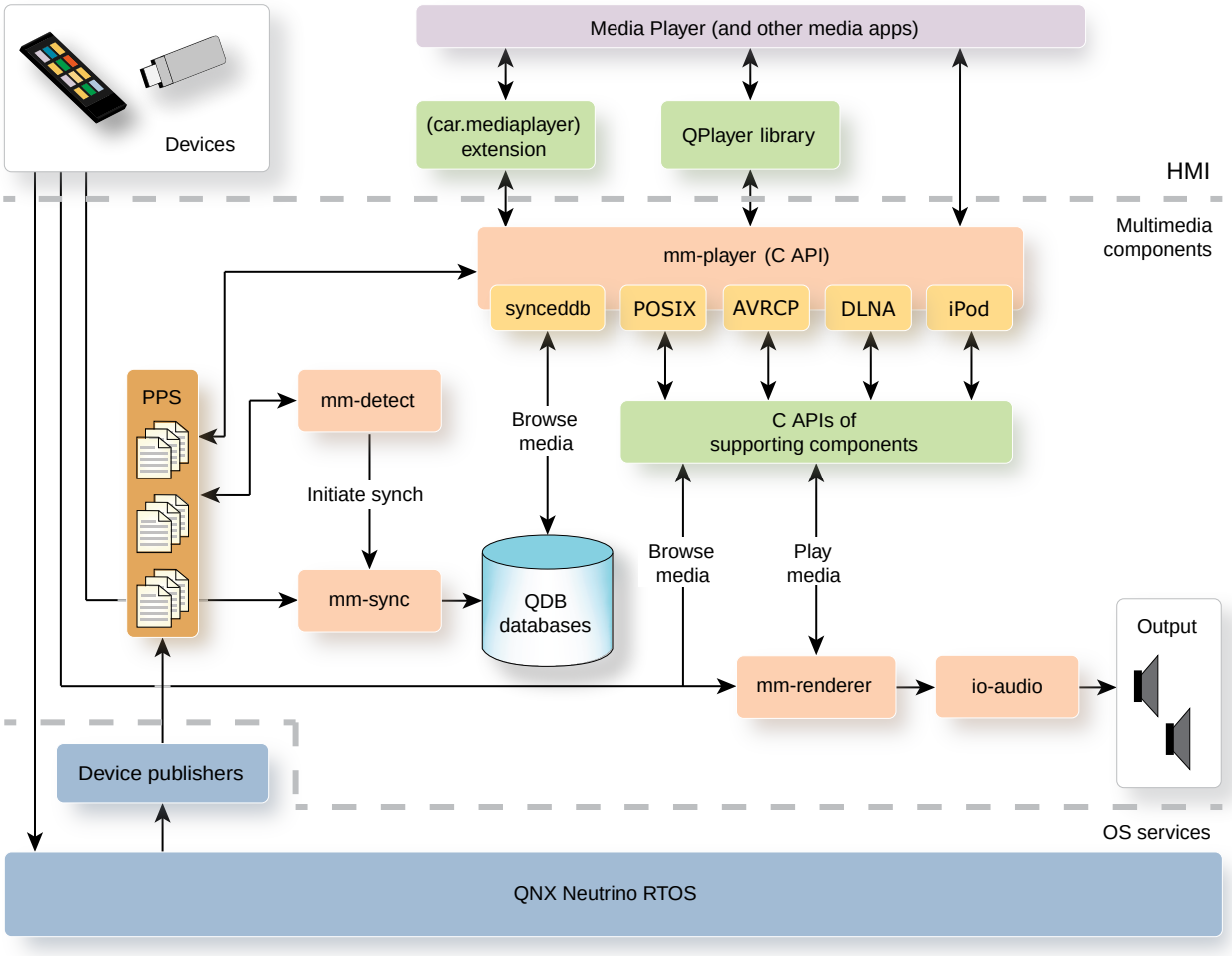

Figure 1: QNX CAR Multimedia Architecture

Communication

As *Figure 1: QNX CAR Multimedia Architecture* (p. 9) shows, communication between and within the HMI, multimedia components, and OS services layers is handled by native extensions, C APIs, QDB databases, and the Persistent Publish/Subscribe (PPS) service. On CAR systems, the PPS and QDB services are running by default. This way, the CAR user can initiate media actions as soon as the HMI loads. The components in the various layers will then forward the appropriate commands and data amongst themselves to perform the requested operations.

HMI

This release includes both an HTML5 and a Qt5 version of the HMI. The HTML5 HMI includes the `car.mediaplayer` WebWorks extension, which provides a JavaScript API that the Media Player app uses to send media requests to `mm-player`. In the Qt5 HMI, Media Player communicates with `mm-player` through the `QPlayer` library, which provides a Qt5 API for issuing media requests.

To customize media browsing and playback, you can modify the reference media player shipped with the platform or write your own media apps. In addition to the WebWorks extension and Qt library, your apps can directly call the C API of `mm-player`. All three interfaces—`car.mediaplayer`, `QPlayer`, and the `mm-player` C API—define functions for:

- retrieving a list of accessible media sources
- retrieving track metadata (e.g., artist name, album name, track title)
- starting and stopping playback
- jumping to a specific track during playback
- handling updates in playback state, media sources, and track postion

The functions defined in these interfaces aren't specific to any device type, which allows your apps to work with a wide variety of media hardware and also simplifies their design because most of the application logic is handled by `mm-player`.

Multimedia components

The `mm-player` service is a media browsing and playback engine that processes commands sent by HMI apps through the `car.mediaplayer` extension. To support different device types, `mm-player` uses plugins. Each plugin interfaces to a particular device type to carry out media operations. For example, suppose you insert an SD card. The POSIX plugin supports this type of device, so it detects the SD card insertion and informs `mm-player` of the newly connected media source. This same plugin will then be used for all media operations on the SD card. For more information about `mm-player` plugins, see the Media Player Plugins chapter in the *Multimedia Player Developer's Guide*.

The following services support the operations of `mm-player`:

mm-detect

Discovers media devices and initiates synchronization of metadata.

mm-sync

Synchronizes metadata from tracks and playlists on media devices into QDB databases.

mm-renderer

Plays audio and video tracks, and reports playback state.

io-audio

Starts audio drivers to enable outputting of audio streams through hardware.

OS services

Device publishers provide information about attached devices through PPS. When the user attaches a device to their in-car system, the appropriate OS services inform the device publishers, which then write device information (e.g., mountpoints) to PPS objects in a directory monitored by `mm-detect`. The `mm-detect` service uses this information to start synchronizing the new device's metadata.

Other OS-layer components, such as the filesystem mounting service and various resource managers, support specific `mm-player` plugins.

Media device detection and synchronization

QNX CAR automates media device detection and synchronization. When the user attaches a device for the first time, the `mm-detect` and `mm-sync` services coordinate to upload the device's metadata to its QDB database.

The interaction between these components proceeds like this:

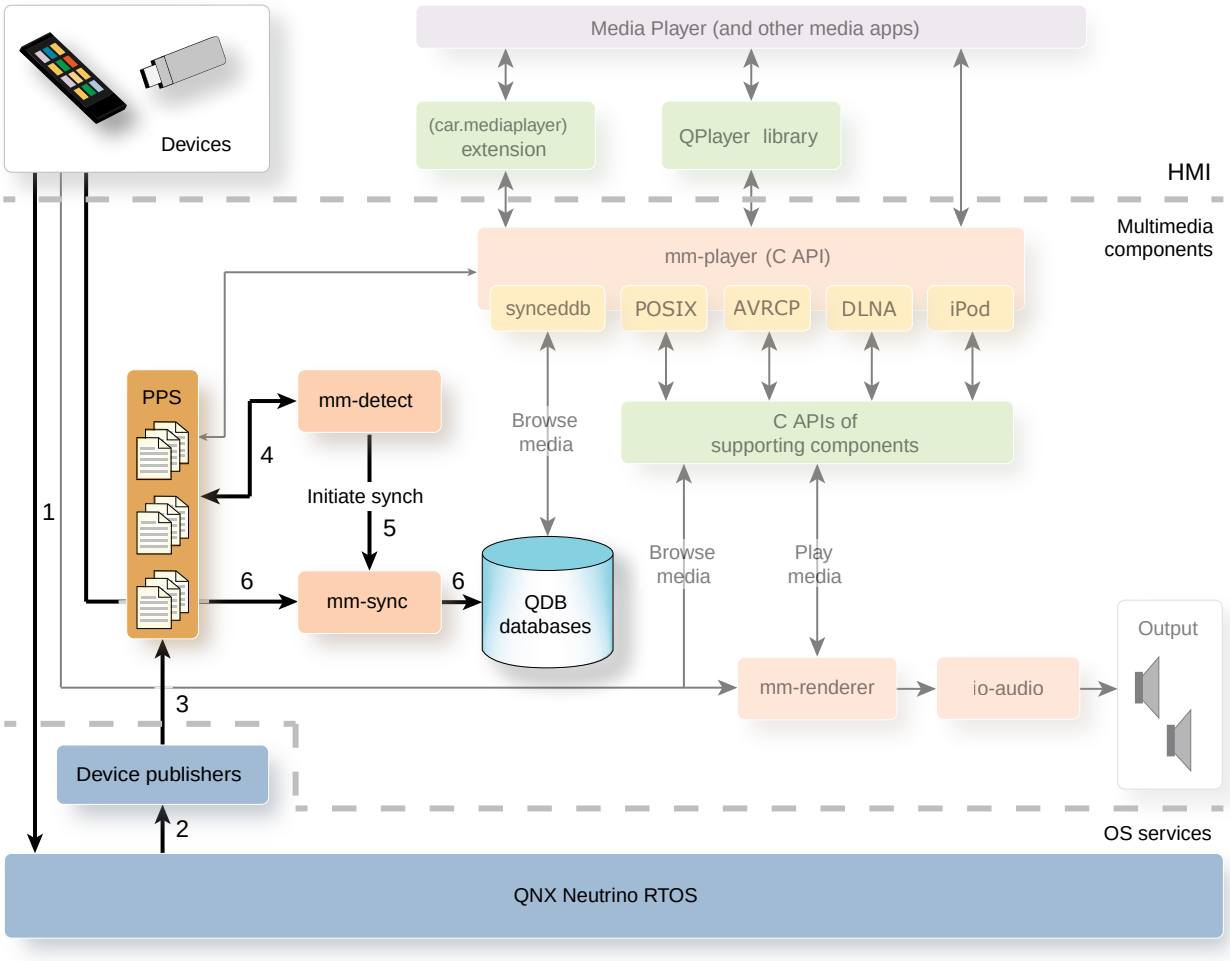

Figure 2: Media device detection and synchronization

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 `io-usb` process for device information (for details, see "The usblauncher Service" in the *Device Publishers Programmer's Guide*). The `mmcsdpub` and `cdpub` publishers monitor specific `/dev` paths and when they notice new or updated entries, they communicate with the drivers to obtain device information (for details, see "Role of device drivers and mcd").

3. **Publishing device 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. **Reading device information from PPS**

   The `mm-detect` service keeps track of which devices are attached to your in-car system by monitoring the filesystem information objects in PPS. When a new object is created following a device attachment, `mm-detect` reads the new device's information from the object, and stores this information.

5. **Initiating synchronization**

   To start synchronizing the device's media metadata, `mm-detect` loads the device's QDB database into memory (see the "Loading QDB databases" section in the *QDB Developer's Guide* for information on how this is done). Next, `mm-detect` passes the device's mountpoint and the name of its database to `mm-sync`. The database name is constructed from the device's unique ID (UID), which was read (along with the mountpoint) from the PPS object. The synchronization path given to `mm-sync` is the root directory of the device's filesystem, which means all media files on the device have their metadata synchronized.

6. **Extracting and storing media metadata**

   The `mm-sync` service uses media libraries (not shown) to read filepaths and other information from media tracks found on the device. Based on its internal mapping of metadata fields to database fields, `mm-sync` copies the extracted metadata into the appropriate tables and columns. At this point, applications can query the QDB database to obtain creation and runtime information on all audio and video content found on the device.

Media browsing

The Media Player and other HMI apps can browse the contents of mediastores through `mm-player`. The `mm-player` plugins can explore filesystems and retrieve metadata from different device types and databases.

The interaction between these components proceeds like this:

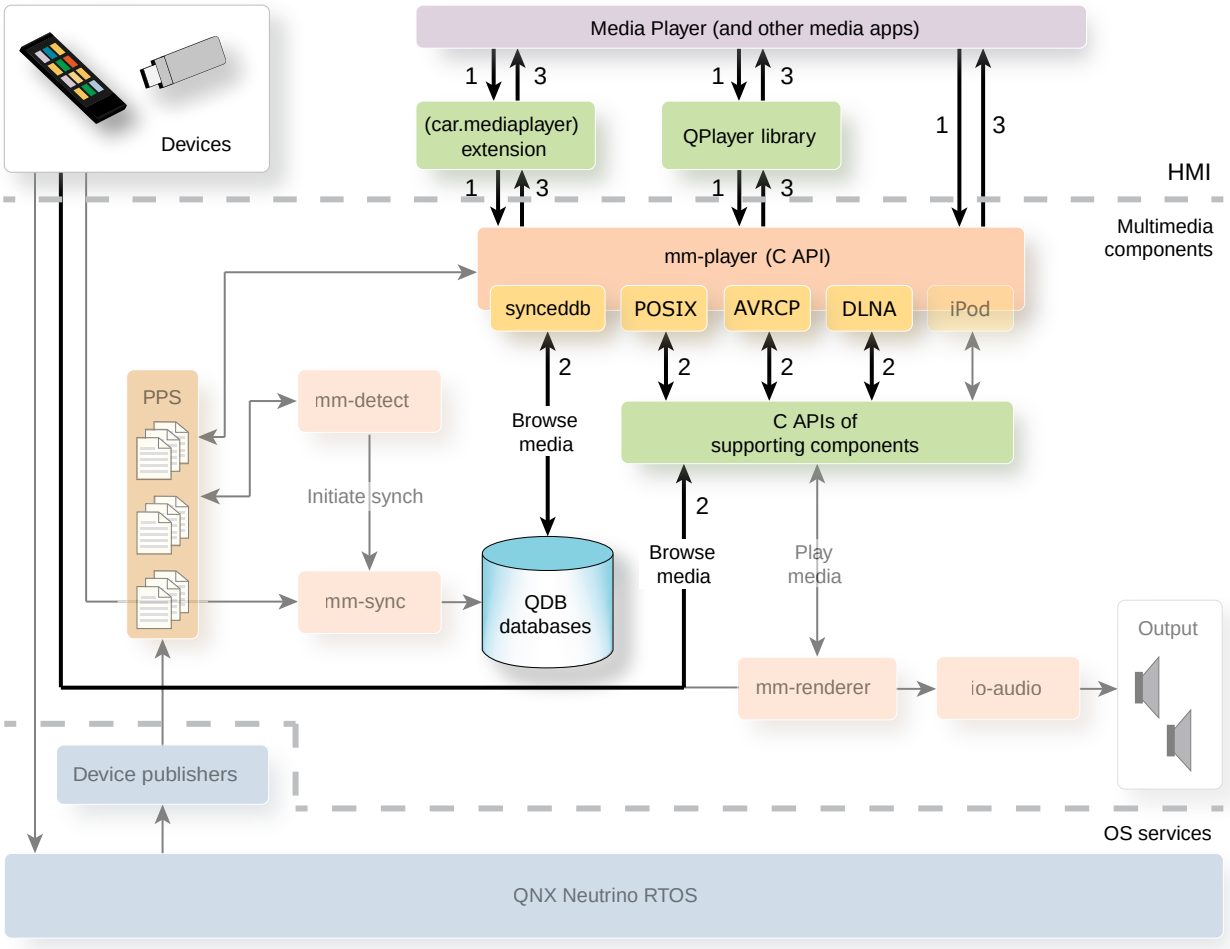

Figure 3: Media browsing

1. **Sending browsing requests**

   To send browsing requests to `mm-player`, HMI apps call functions in either the `car.mediaplayer` WebWorks extension (for the HTML5 HMI) or the `QtQnxCar2` library (for the Qt5 HMI), which then forwards the requests to the C API of `mm-player`.

   Each of these components exposes the following browsing operations:

   - *browse*

- *search*
- *getMetadata*

2. **Navigating filesystems and retrieving metadata**

   The `mm-player` service uses plugins to explore mediastore filesystems and extract metadata from their files. Each plugin uses a different mechanism to carry out the *browse*, *search*, and *getMetadata* operations on a particular device type. For instance, the `synceddb` plugin queries QDB databases for file information whereas the POSIX plugin uses the `mmbrowse` library to navigate directories and read information from locally mounted files. For details on how individual plugins browse media content, refer to the Multimedia Player Plugins chapter in the *Multimedia Player Developer's Guide*.

3. **Delivering media information**

   After the appropriate plugin retrieves the file information or track metadata requested by the browse operation, `mm-player` delivers the results data to the HMI through either the `car.mediaplayer` extension or the `QtQnxCar2` library, depending on which HMI version is in use. For examples of reading results data after issuing function calls or REST requests to the first of these mechanisms, see the "car.mediaplayer.Mediaplayer" section of the *HTML5 Developer's Guide*.

   The *browse* and *search* operations return file information that includes but is not limited to:

   - filepath
   - media item type (e.g., audio, video, folder)

   The *getMetadata* operation returns track metadata that may include but is not limited to:

   - artist name
   - album name
   - track title
   - duration (runtime)

   The plugins also notify `mm-player` when any serious browsing error occurs, so that the service can inform HMI apps when a browse operation fails.

Media playback

The Media Player and other HMI apps can play media files through `mm-player`. The `mm-player` plugins manage their own tracksessions (track sequences) and support playback on different device types by using `mm-renderer` and device-specific components to control media streams.

The interaction between these components proceeds like this:

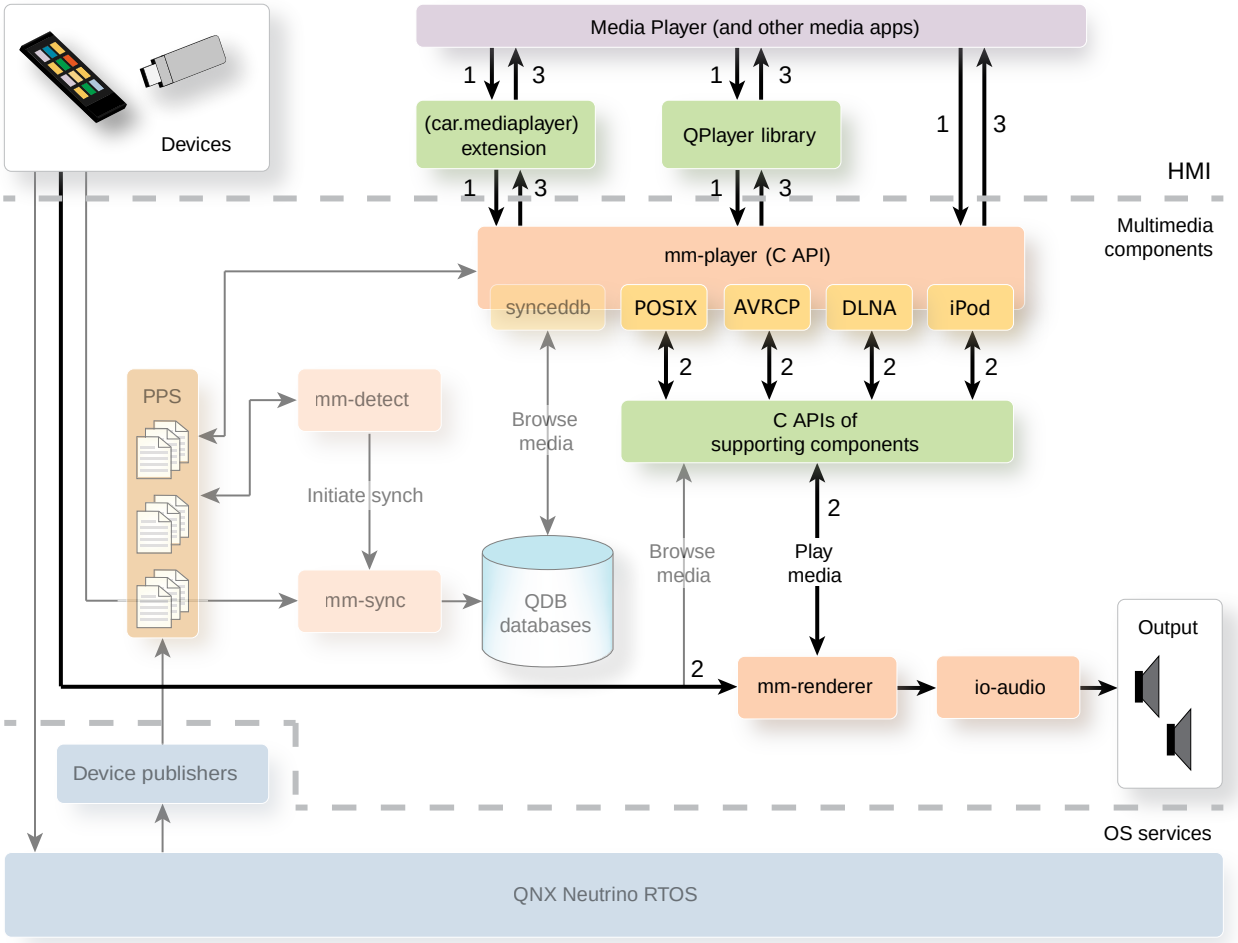

Figure 4: Media playback

1. **Sending tracksession and playback requests**

   To send tracksession and playback requests to `mm-player`, HMI apps call functions in either the `car.mediaplayer` WebWorks extension (for the HTML5 HMI) or the `QtQnxCar2` library (for the Qt5 HMI), which then forwards the requests to the C API of `mm-player`.

   The tracksession operations exposed by each of these components include but are not limited to:

- *createTrackSession*
- *destroyTrackSession*
- *getCurrentTrack*
- *getTrackSessionItems*

The playback operations exposed include but are not limited to:

- *play*
- *stop*
- *setPlaybackRate*
- *seek*
- *next*
- *previous*
- *shuffle*

2. **Managing tracksessions and playing media**

   The `mm-player` service uses tracksessions to store playback sequences of tracks. For commands related solely to tracksession management (e.g., *createTrackSession*, *deleteTrackSession*), `mm-player` creates, deletes, or updates its internal data structures that store tracksession information, but then skips the rest of this step.

   For commands affecting playback activity (e.g., *play*, *setPlaybackRate*), `mm-player` reads its tracksession data structures to determine which track to play next based on the requested operation. For instance, when processing the *next* command, `mm-player`:

   a. **Retrieves the index of the next track**

      Each tracksession stores a sequential track list, which lists tracks in their relative order defined at the media source, and a randomized track list, which lists track in a random order generated when an app issues the *shuffle* command. The list that `mm-player` reads the index of the next track from depends on the shuffle setting.

   b. **Retrieves the metadata and the URL of the next track**

      Using the index of the next track, `mm-player` looks up the track's URL and metadata, which are also stored in the tracksession. The URL indicates the location of the media file that will become the new input to `mm-renderer`. The metadata contains track creation and runtime information that `mm-player` can deliver to HMI apps so that they can refresh their display to show the newly selected track.

   To manage playback, `mm-player` uses device-specific components to initiate and terminate media streams from devices and also uses `mm-renderer` to direct the media content to the designated output.

Each plugin interacts with different libraries and drivers to direct playback commands to the devices that it supports. For example, the iPod plugin sends commands to the `ipodlib` library, which communicates with the driver that manages the audio device that the iPod is connected to. The plugin also provides `mm-renderer` with the URL of the device path (i.e., `/dev` entry) for this same device, to attach it as the input. When the library instructs the device driver to set its media sampling rate above `0`, the media content starts flowing from the iPod, through the audio device, to `mm-renderer`. As it does with all plugins, the `mm-renderer` service then directs the media flow to the `io-audio` utility, which outputs it over the designated hardware (e.g., speakers).

For a more detailed explanation of playback, see the "Media playback" section in the *Multimedia Architecture Guide* for the QNX SDK for Apps and Media.

In this release, `mm-player` doesn't support video playback (only audio). To play videos, your apps must use the HTML5 video features.

3. **Reporting tracksession and playback state**

   After the necessary tracksession and playback actions are carried out by a plugin, the `mm-player` service delivers any results data to the HMI through either the `car.mediaplayer` extension or the `QtQnxCar2` library, depending on which HMI version is in use. For examples of retrieving results data after issuing function calls or REST requests to the first of these mechanisms, see the "car.mediaplayer.Mediaplayer" section.

   The *getTrackSessionItems* and *getCurrentTrack* operations return information for tracksession items that includes but is not limited to:

   - filepath
   - media item type (e.g., audio, video, folder)

   The *getCurrentTrack* operation also returns metadata, which includes but is not limited to:

   - artist name
   - album name
   - track title
   - duration (runtime)

   The playback operations listed in *Step 1* (p. 16) don't produce any new playback data. For these operations, no new information needs to be communicated to the HMI, so the WebWorks extension or the Qt library returns only a Boolean field indicating if the operation succeeded or failed.

   The plugins also notify `mm-player` when any serious playback error occurs, so that the service can inform HMI apps when a playback operation fails.

Index