Multimedia Controller Configuration 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: Tuesday, February 25, 2014

Table of Contents

About This Guide

The *Multimedia Controller Configuration Guide* provides QNX CAR users with a reference for commands they can issue to the `mm-control` service to perform multimedia tasks such as playback, output device definition, and tracksession management. Understanding the capabilities and PPS interface of `mm-control` is necessary if you want to write your own media applications that use this service or to customize the Media Player application shipped with the QNX CAR platform.

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

To find out about:	Go to:
The role of `mm-control` in car infotainment systems and the multimedia components it interacts with	*Multimedia Controller Overview* (p. 9)
Starting `mm-control` with specific command options during bootup	*Running mm-control* (p. 13)
The syntax of the command line for starting `mm-control`	*mm-control command line* (p. 19)
The procedure used by `mm-control` to set up its PPS interface and start listening for media commands	*Setup and Command Monitoring* (p. 23)

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
Multimedia Controller Overview

The multimedia controller service, mm-control, listens through the Persistent Publish/Subscribe (PPS) service for commands sent by media applications and then invokes the appropriate platform components to carry out the requested media operations.

The `mm-control` service provides a middle layer between the car system's HMI and the lower-level multimedia services that access media content on devices and manage playback. By providing this middle layer, the QNX CAR platform abstracts several components into one controller that supports many types of media operations.

HMI applications can write media commands (encoded as JSON objects) into the PPS control object monitored by `mm-control` and then read the results of those commands through the same PPS object, after `mm-control` has processed those commands.

The QNX CAR platform includes the following components (multimedia services) that continuously listen for and execute commands forwarded by `mm-control`:

- **Digital Media Controller client (dmcclient)**: one of many DLNA services, this component processes commands (e.g., `play`, `pause`, `seek`) that control playback on DMS devices paired with the in-car system
- **mm-trkmgr**: the multimedia tracksession manager, which creates and deletes tracksessions, and updates information on the tracks within a tracksession
- **mm-renderer**: the low-level media engine, which starts and stops the media flow, and controls the playback speed, position, and repeat mode

Deciding whether to run mm-control

You can choose whether or not to run `mm-control` in your car system. The `mm-control` service is a convenience feature that allows you to issue a default set of media commands through PPS. But you can replace `mm-control` with your own PPS-based service or write applications that access the lower-level multimedia services.

By default, `mm-control` is launched during bootup by the System Launch Monitor (SLM) utility. To prevent `mm-control` from being started, you can edit the SLM configuration file (see the *Preventing mm-control from running* (p. 16) section).

When to use mm-control

You should use `mm-control` if you want to:

- simplify your applications by using a built-in service that processes commands for many types of media operations; this way, you don't have to write your own command-processing service or learn the APIs of multiple special-purpose components
- send declarative commands, compactly encoded in JSON, through PPS to perform media operations in one step
- read the results of media operations consistently and easily, by examining the `id` and `err` attributes in the JSON data sent through PPS

When not to use mm-control

You shouldn't use `mm-control`, but instead use an alternative mechanism for handling media operations if you want to:

- perform media operations not supported by `mm-control`, such as adjusting the playback volume or changing playlists without interrupting playback
- reduce your system load by running fewer multimedia services; for example, if your applications are designed to use only `mm-renderer`'s features to play content from only USB devices, you don't need to run `mm-trkmgr` or any of the DLNA services

There are three alternatives to using `mm-control`. You can:

- write a replacement service that interprets PPS commands and invokes the lower-level multimedia services to execute those commands
- write applications in HTML5 and use the HTML5 audio and video extensions to manage playback
- write applications in C and directly call the APIs of individual multimedia services

To use Media Player when you're not running `mm-control`, you must modify the application so it uses one of the mechanisms listed in the first two alternatives.

Writing a replacement for `mm-control`

You can write your own middle layer for processing media commands sent by HMI applications written in HTML5.

A middle-layer component with a PPS interface is an easy design option because the QNX CAR platform includes the JNEXT PPS extension, which provides JavaScript functions for accessing PPS objects. Using this provided extension saves you from writing your own extension that wraps the API calls to native (i.e., nonweb) applications in JavaScript.

For an example of accessing PPS by using JNEXT, see the Media Player code, located in `/apps/MediaPlayer.testDev_MediaPlayer49ba23c5/native/`.

Writing applications that use the HTML5 media extensions

You can write HMI applications that use the standard HTML5 audio and video extensions for media operations. This lets you avoid using a middle layer or writing any JNEXT extensions.

The web browser in the HMI uses the HTML5 audio and video extensions to play media files. These extensions are part of a webkit that transparently invokes `mm-renderer` to manage media playback.

Writing applications that directly access multimedia services

If your system design requires that you program in C, you can write HMI applications that directly access individual multimedia services instead of sending PPS commands through a middle layer. With this design, your code can call the API functions of the `mm-trkmgr`, `mm-renderer`, and DLNA services, and then read the return codes and error information sent back by these APIs to interpret the results of media operations.

mm-control components

The `mm-control` service exposes several media-management features in one collection of PPS commands. Applications specify commands by stating the operation name followed by the data needed for the operation.

The operation name consists of multiple words: the first word is the command category and the remaining words refer to the specific action to perform. The command category determines which `mm-control` object type the command affects. Separate subsystems process commands targetted to each of the following object types:

- **outputs**: outputs provide destinations for media playback; destinations can be audio or video devices
- **zones**: zones group outputs together for convenience; for example, you could define one zone for outputting video content and another zone for outputting audio content
- **tracksessions**: tracksessions manage ordered track collections that you can provide to a player; each tracksession stores a start index and an end index as well as the URLs to the source files of its tracks
- **players**: players control media flows from an input to one or many outputs; each player operates independently of the others, meaning the playback commands sent to it affect only its own media flow

All information on outputs and zones is maintained in internal data structures, so the commands that configure these types of objects don't entail communicating with other multimedia services. But the commands affecting tracksessions and players do require communicating with `mm-trkmgr`, `mm-renderer`, and possibly other multimedia services (depending on the device type). When processing commands in either of these last two categories, `mm-control` uses either PPS objects or API calls to invoke the multimedia services necessary to carry out the command.

See the `/pps/services/mm-control/control` entry in the *PPS Objects Reference* for full details on the command message format and the commands supported by each of the `mm-control` components.

Chapter 2
Running mm-control

Client applications don't need to explicitly start the mm-control service before performing media operations. The QNX CAR platform uses the System Launch Monitor (SLM) service to start mm-control during bootup; this ensures mm-control is running when the HMI loads and users begin interacting with its applications. For recovery purposes, applications can start mm-control manually.

Starting mm-control with specific command options during bootup

SLM automates process management by launching processes in an order that respects their interprocess dependencies. The list of processes to launch and their properties, including their command-line arguments and interprocess dependencies, is written in a configuration file. During bootup, SLM reads this file and carries out its instructions for starting processes.

Using SLM to start `mm-control` ensures that the system is ready to process media commands when the HMI finishes loading. For more information on SLM, refer to "System Launch and Monitor (SLM)" in the *System Services Reference*.

SLM is preconfigured to start `mm-control` with specific command options, but you can specify whatever command options you want.

To change the command options passed by SLM to `mm-control`:

1. From a command console connected to your car system, navigate to and open the SLM configuration file, whose default path is: `/etc/slm-config-all.xml`.
2. In the configuration file, locate the component that specifies the properties for `mm-control`.

   This component is the `<SLM:component>` XML object with the name `"mmcontrol"`.
3. Change the value of the `<SLM:args>` tag in the `"mmcontrol"` component to hold the new set of command-line options to pass to `mm-control` at startup.

   For the full list of command-line options, see the *mm-control command line* (p. 19) section.
4. Save the changes to the SLM configuration file and return to the console.
5. If you want the new configuration to take effect immediately, enter `reboot` in the console.

   The system reboots and the SLM utility relaunches all the processes, including `mm-control`, with the command options specified in the configuration file. When the system finishes reloading, `mm-control` is running with the new configuration.

If you don't reboot after changing the configuration file, `mm-control` continues to run with its previous configuration until you shut down the system and restart, at which point the new command options take effect.

Restarting mm-control manually

An application can restart `mm-control` with an explicit command if the service has unexpectedly stopped and if the application can't continue without support for media operations.

The exact conditions that require a client application to restart `mm-control` are:

- the `mm-control` process has terminated abnormally;
- the client must perform media management tasks that require the use of `mm-control`; and
- the car infotainment system can't be rebooted to restart `mm-control` with SLM because doing so would be too disruptive to the user.

To restart `mm-control` manually, your application must:

1. Confirm that `mm-control` isn't already running by checking the list of active processes with the `pidin` or `ps` command.
2. Confirm that the `pps`, `io-audio_mcbsp`, `audioman`, `mm-renderer`, and `mm-trkmgr` processes are already running by checking the same list of active processes.

   The `mm-control` process depends on all five of these other services, so if any one is *not* running, that service must be started; otherwise, `mm-control` won't run properly.
3. Send the command line for running `mm-control` with the desired command options to the OS, using the *system()* or *spawn()* system call.
   The OS tries to run `mm-control` and reports the operation outcome to `sloginfo`.

   For details on the system calls that send commands to the OS, see the *system()* or *spawn()* section in the *C Library Reference*. For the full list of command-line options, see the *mm-control command line* (p. 19) section.

If `sloginfo` shows no error, `mm-control` is running again, so your application can resume sending PPS commands to the service to perform media operations.

Your car infotainment system should run only one instance of `mm-control`, so separate media applications must coordinate with each other to avoid redundantly starting `mm-control`.

Preventing mm-control from running

By default, `mm-control` is running on QNX CAR systems. If you decide to use a different mechanism for processing media commands, you can prevent SLM from launching `mm-control` during bootup by editing the SLM configuration file.

The version of the Media Player application shipped with the QNX CAR platform won't function properly if you don't run `mm-control`. To use Media Player without `mm-control`, you must modify the application to use either a replacement service or the HTML5 media extensions to execute media operations. Both of these scenarios are explained in the *Deciding whether to run mm-control* (p. 10) section.

To prevent SLM from launching `mm-control`:

1. From a command console connected to your car system, navigate to and open the SLM configuration file, whose default path is: `/etc/slm-config-all.xml`.
2. In the configuration file, locate the component that specifies the properties for `mm-control`.

   This component is the `<SLM:component>` XML object with the name `"mmcontrol"`.
3. Disable the component by commenting it out (using the XML syntax of <!-- and -->) or by deleting it from the file.
4. Save the changes to the SLM configuration file.

In all subsequent bootups of the car system, `mm-control` won't be started.

mm-control SLM specification

SLM uses an XML configuration file to store the list of processes to be automatically started and their properties. In QNX CAR systems, `mm-control` and its prerequisite and dependant programs are listed in the SLM configuration file. You can change this configuration file to run `mm-control` with different command options.

The following excerpt from the SLM configuration file shows some of the property settings for `mm-control`:

```
<SLM:component name="mmcontrol">
    <SLM:command>mm-control</SLM:command>
    <SLM:args>-v -c /etc/mm-control.cfg</SLM:args>
    <SLM:waitfor wait="pathname">
            /pps/services/mm-control/control</SLM:waitfor>
    <SLM:depend>mmtrkmgr</SLM:depend>
    <SLM:depend>mmrenderer</SLM:depend>
</SLM:component>
```

Here, the name `"mmcontrol"` assigned to the `<SLM:component>` XML object is just an internal label used within the configuration file. This label differs from the actual process name of `mm-control`, which is provided in the `<SLM:command>` tag.

Command-line arguments

The `<SLM:args>` tag lists the command-line arguments. By default, one level of verbosity (-v) and the path for the included configuration file are specified, but you can change the value of `<SLM:args>` to use different command-line arguments. The new settings take effect after you reboot the system and SLM relaunches the service. See *Running mm-control* (p. 13) for instructions on changing the command settings for `mm-control`.

Workflow

The `<SLM:depend>` objects list which processes must be running before `mm-control` can be started. Some programs that are prerequisites to `mm-control` have their own prerequisites. For instance, `mm-renderer` requires `audioman` to be running because the former service uses the audio manager library. The interprocess dependencies between `mm-control` and the other programs it uses make up a complex workflow of processes.

The `<SLM:waitfor>` object specifies that the `mm-control` service must create a PPS object at the path matching the `<SLM:waitfor>` tag value before SLM can start any processes that depend on `mm-control`. Because the PPS service must be running for `mm-control` to create its PPS object, there's an indirect process dependency of `mm-control` on `pps`.

The following is an illustration of the process workflow subsection closest to `mm-control`:

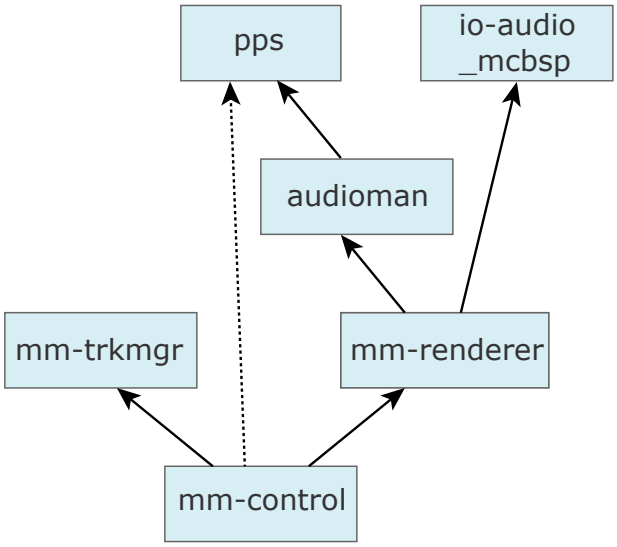

Figure 1: Workflow of mm-control and related processes

In *Figure 1* (p. 18), the solid lines represent direct process dependencies, which are specified in <SLM:depend> objects, and the dotted line represents an indirect process dependency, which is specified in a <SLM:waitfor> object.

mm-control command line

Start multimedia controller service

Synopsis:

```
mm-control [-b] [-p priority] [-c config_file] [-v]
```

Options:

-b

Run the `mm-control` process in the foreground. This option is handy for debugging because it makes `mm-control` send log messages to the standard error in addition to `sloginfo`.

By default, `mm-control` runs in the background.

-p *priority*

Set the priority of the `mm-control` process. When your system is busy running many applications, the priority level can have a considerable impact on the turnaround time for servicing media operation requests.

The valid range for priority settings is 1 to 63; the default setting is 15.

-c *config_file*

The path of the configuration file, which defines the initial outputs and zones.

A configuration file is included with `mm-control`, located at `/etc/mm-control.cfg`.

-v

Increase output verbosity. Messages are written to `sloginfo`. The -v option is cumulative, so you can add several v's to increase verbosity, up to five levels.

Output verbosity is handy when you're trying to understand the operation of `mm-control`. However, when lots of -v arguments are used, the logging becomes quite significant and can change timing noticeably. The verbosity setting is good for systems under development but probably shouldn't be used in production systems or during performance testing.

Description:

The `mm-control` command starts the multimedia controller service, which listens for commands sent through the PPS service and then invokes the necessary platform components to carry out the requested media operations. Each time a command is written to the PPS control object (`/pps/services/mm-control/control`), `mm-control` verifies the syntax of the command and if it's valid, writes instructions to PPS or makes API calls to invoke the necessary multimedia services to execute the command.

The outcomes (results) of media operations are written to the same PPS object so that client applications can read that object to get those results.

Through `mm-control` command options, you can name the configuration file used for defining initial outputs and zones, enable logging, and set process options such as the priority level. To change the command-line arguments passed to `mm-control`, you must modify the SLM configuration file.

mm-control configuration file

The `mm-control` configuration file lets you define a default set of outputs and zones. This data is specified in JSON format, which provides a compact way of expressing lists of available playback destinations. At startup, `mm-control` reads the configuration file and stores its information in internal data structures.

Defining an initial set of outputs and zones ensures media applications such as Media Player have default destinations for outputting media content during playback.

QNX CAR systems include an `mm-control` configuration file, whose full path is: `/etc/mm-control.cfg`. You can change the initial set of outputs and zones either by modifying the included configuration file or by creating another configuration file. When using a different file, you must direct `mm-control` to that file by using the -c command option, as explained in the *mm-control command line* (p. 19) section.

The full contents of the included file look like this:

```
{
    "outputs":[
        {
            "cabin":{
                "url":"audio:default",
                "type":"audio"
            }
        },
        {
            "rear":{
                "url":"snd:pcmPreferredpb",
                "type":"audio"
            }
        },
        {
            "output2":{
                "url":"screen:?dstx=320&dsty=85&zorder=100\
                              &dstw=460&dsth=259",
```

```
                "type":"video"
            }
        }
    ],
    "zones":[
        {
            "audio":[
                "cabin"
            ]
        },
        {
            "video":[
                "cabin",
                "output2"
            ]
        }
    ]
}
```

This configuration defines separate audio outputs for the front and rear speakers of the car, and a video output with the specified screen location and dimensions. Separate zones for outputting audio content and video content are also defined. Although the rear speakers aren't used in any of the default zones, you could use them as a destination in media playback.

Chapter 3
Setup and Command Monitoring

At startup, mm-control parses its command-line options and a configuration file (if specified), which lists the default outputs and zones for media playback. The mm-control service then creates and begins monitoring a Persistent Publish/Subscribe (PPS) object, to accept commands from client applications for performing media operations.

These setup tasks are performed in an automated, ongoing procedure. Understanding this procedure is necessary if you want to write a replacement service for `mm-control` or write your own media applications that use `mm-control`.

The `mm-control` process performs the following setup steps:

1. **Command-line and configuration file parsing**

   At startup, `mm-control` parses its command-line options, which may define:

   - the path to the configuration file, which defines the default outputs and zones available for media playback
   - process metrics, including the scheduling priority level and a flag for running in the background
   - the verbosity level, which is useful for debugging

   When `mm-control` learns the path of the configuration file, the service immediately parses that file, verifying that its contents are in proper JSON format. If so, `mm-control` builds up lists of default outputs and default zones by copying the data from the entries in the `outputs` and `zones` list objects found in the configuration file into internal arrays.

   If the file contents are not properly formatted, `mm-control` reports an error in `sloginfo` and skips the step of setting up a default set of outputs and zones.

2. **PPS interface setup**

   The `mm-control` process calls *open()* to create a PPS control object (`/pps/services/mm-control/control`) that provides a mechanism for client applications to issue media commands. This PPS object is opened in *delta mode*, which allows any process reading the object (`mm-control` or an HMI application monitoring media operations) to receive only the changed object content when the object is updated. The `mm-control` process then calls *read()* on the PPS control object; this call blocks until the object is updated by another process.

   For information on opening a PPS object with the `delta` and `wait` options set, which is necessary to achieve this behavior, see the "Pathname open options" section in the *QNX PPS Developer's Guide*.

3. **Media command processing**

   When a client application writes a command into the PPS control object, the *read()* call unblocks and returns the new object content, which should contain a syntactically valid command. If so, the appropriate `mm-control` subsystem then processes the command and writes its outcome to the PPS control object. The client that issued the command can then read the outcome from this object.

   For details on how `mm-control` parses, executes, and reports the outcome of a command, see *Media command processing* (p. 25).

   After processing a command, `mm-control` issues another call to *read()* to resume monitoring the PPS control object for new commands. Thus, this third step is repeated each time a command is issued.

The `mm-control` service runs continuously, so client applications can issue media commands and read their outcomes through PPS at any time. If the `mm-control` process terminates unexpectedly, an application can restart the service manually to restore support for media operations.

Media command processing

The `mm-control` service monitors its PPS control object constantly. When a client writes a command to this PPS object, `mm-control` checks the command syntax and if it's valid, invokes other multimedia services to perform the requested media operation. When these services report that the operation is complete, `mm-control` writes the operation's outcome to the same PPS object.

We recommend that any PPS-based service that you write to replace `mm-control` follow the procedure and design described here.

The `mm-control` service consists of a PPS communications layer and several subsystems that each manage a different set of media operations. When a client application writes a new command to the PPS control object, these `mm-control` components process the command by performing the following steps:

1. **Reading new PPS object content**

   The *read()* system call returns the new content of the PPS control object to the `mm-control` service. The service's PPS layer searches this content for the `msg` and `dat` attributes; if either one is missing, the service logs an error and stops processing the command.

2. **Dispatching appropriate subsystem**

   If the command written to PPS has the necessary attributes, `mm-control` reads the operation name stored in the `msg` attribute. The first word in the operation name refers to the command category, which must be one of the following:

   - `output`
   - `zone`
   - `trksession`
   - `player`

   The commands in each of these categories operate on different object types, as explained in *mm-control components* (p. 12). Based on the command category, `mm-control` invokes the appropriate subsystem to examine the remaining words in the operation name.

   If the operation name doesn't begin with any of the four words listed above or if its remaining words don't refer to any action supported by the selected subsystem, `mm-control` logs an error and stops processing the command.

3. **Data and state validation**

   The `mm-control` subsystem that's processing the command examines the `dat` attribute to verify that the client provided all the necessary data parameters, and then verifies that the parameter values and the current playback state are valid for

the requested operation. For example, when issuing the "player_play" command, a client must provide a player name that matches one of the entries in the mm-control players list. Also, the playback state, which is also stored by mm-control, must be IDLE, PAUSED, or STOPPED.

If the validity check for the data and the playback state fails, mm-control logs an error and stops processing the command.

4. **Invoking lower-level multimedia services**

   The lower-level multimedia services aren't needed to carry out any commands in the output and zone categories; for these commands, mm-control skips this step.

   When processing trksession or player commands, mm-control either writes to PPS objects or makes API calls to invoke the lower-level multimedia services to carry out the requested media operation. For instance, in response to the "player_set_params" command, the player subsystem examines the type attribute found in the content read from the PPS control object. Based on this attribute's value (e.g., input, track), the subsystem then calls the mm-renderer API function that sets the specified type of parameters (e.g., *mmr_input_parameters()*, *mmr_track_parameters()*).

   For the full list of supported media commands, shown by category, see the /pps/services/mm-control/control entry in the *PPS Objects Reference*.

5. **Updating internal data structures**

   The mm-control subsystem that processes the command checks the return codes of all API function calls. After each successful call, the subsystem updates the internal data structures as necessary. For example, the *mmr_set_speed()* returns 0 after the "player_set_speed" command has been successfully executed. In response, the player subsystem updates the speed setting in the data structure that holds information on the player indicated by the command's data.

   For commands that don't require using any lower-level multimedia services (e.g., "output_create" and "zone_attach_outputs", the relevant subsystem simply updates the lists of media resources available for playback, based on the requested operation.

6. **Reporting operation outcome**

   The last command-processing activity done by mm-control is to write the outcome (result) of the media operation into the PPS control object.

   The operation name is echoed in the res attribute. Any data describing the new playback state is encoded in JSON format and stored in the data attribute; for instance, after executing the "player_next_track" command, mm-control reports the track ID, file ID, and URL for the newly selected (and now playing) track. The error code is written to the err attribute. If this code is 0, then no error

has occurred. Otherwise, the `errstr` attribute contains a brief description of the error encountered.

Clients can retrieve the results as soon as `mm-control` updates the PPS control object by using delta mode and blocking *read()* calls, identical to how `mm-control` reads this same PPS object.

Index