QNX CAR™ Platform for Infotainment 2.1

PPS Objects Reference

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: Monday, October 6, 2014

Table of Contents

About This Reference

The *PPS Objects Reference* describes each PPS object supplied with the QNX CAR platform. The following table may help you find information quickly:

To find out about:	Go to:
Format of PPS objects	Overview of the PPS Service
Custom objects	Setting Up Your Own Objects
Details of each PPS object	PPS Objects Reference Pages
Objects used internally by system processes	List of Objects Used Internally

For more information about the PPS service itself, see:

- The "PPS" chapter in the QNX Neutrino *System Architecture* guide
- Persistent Publish/Subscribe Developer's Guide

Using this reference

In this reference, each PPS object in the system has its own page. The title of each page is the object's filename (e.g., `/pps/services/bluetooth/control`). Pages are listed in alphabetical order.

The following groupings may help you locate one or more related PPS objects:

App Launcher

- `/pps/services/app-launcher`
- `/pps/services/launcher/control`

Audio

- `/pps/services/audio/audio_router_control`
- `/pps/services/audio/audio_router_status`
- `/pps/services/audio/control`
- `/pps/services/audio/devices/`
- `/pps/services/audio/mixer`
- `/pps/services/audio/status`
- `/pps/services/audio/types/`

- `/pps/services/audio/voice_status`

Authorization

- `/pps/accounts`

Bluetooth

- `/pps/services/bluetooth/control`
- `/pps/services/bluetooth/handsfree/control`
- `/pps/services/bluetooth/handsfree/status`
- `/pps/services/bluetooth/messages/control`
- `/pps/services/bluetooth/messages/notification`
- `/pps/services/bluetooth/messages/status`
- `/pps/services/bluetooth/paired_devices/<mac_addr>`
- `/pps/services/bluetooth/phonebook/control`
- `/pps/services/bluetooth/phonebook/status`
- `/pps/services/bluetooth/remote_devices/<mac_addr>`
- `/pps/services/bluetooth/services`
- `/pps/services/bluetooth/settings`
- `/pps/services/bluetooth/spp/spp`
- `/pps/services/bluetooth/status`

Demo applications

- `/pps/applications/weathernetwork/`
- `/pps/services/gears/control`
- `/pps/services/gears/status`

Devices

- `/pps/qnx/device/<device>`
- `/pps/qnx/device/<device>_ctrl`
- `/pps/qnx/driver/<pid>`
- `/pps/qnx/mount/<device>`
- `/pps/services/clock/control`
- `/pps/services/clock/status`

Geolocation

- /pps/services/geolocation/control
- /pps/services/geolocation/status

Keyboard

- /pps/system/keyboard/control
- /pps/system/keyboard/status

MirrorLink

- /pps/services/mirrorlink/applications
- /pps/services/mirrorlink/entities
- /pps/services/mirrorlink/rtp
- /pps/services/vnc/discovery/

Multimedia

- /pps/applications/mediaplayer
- /pps/services/mm-control/control
- /pps/services/mm-control/<playername>/status
- /pps/services/multimedia/renderer/component
- /pps/services/multimedia/renderer/context/<contextname>/
- /pps/services/multimedia/renderer/context/<contextname>/metadata
- /pps/services/multimedia/renderer/context/<contextname>/output#
- /pps/services/multimedia/renderer/context/<contextname>/p#
- /pps/services/multimedia/renderer/context/<contextname>/param
- /pps/services/multimedia/renderer/context/<contextname>/play-queue
- /pps/services/multimedia/renderer/context/<contextname>/q#
- /pps/services/multimedia/renderer/context/<contextname>/status
- /pps/services/multimedia/renderer/context/<contextname>/state
- /pps/services/multimedia/renderer/control

Navigation (turn-by-turn)

- /pps/qnxcar/navigation/control
- /pps/qnxcar/navigation/geolocation
- /pps/qnxcar/navigation/options
- /pps/qnxcar/navigation/status

Navigator (Applications Window Manager)

- `/pps/system/navigator/appdata`
- `/pps/system/navigator/applications/applications`
- `/pps/system/navigator/command`
- `/pps/system/navigator/windowgroup`
- `/pps/system/navigator/windowparams`

Networking

- `/pps/services/networking/all/interfaces/<interface>`
- `/pps/services/networking/all/proxy`
- `/pps/services/networking/all/status_public`
- `/pps/services/networking/control`
- `/pps/services/networking/proxy`
- `/pps/services/networking/status`
- `/pps/services/networking/status_public`

Now Playing

- `/pps/services/multimedia/mediacontroller/control`
- `/pps/services/multimedia/mediaplayer/control`
- `/pps/services/multimedia/mediaplayer/phone`
- `/pps/services/multimedia/mediaplayer/status`

Profiles

- `/pps/qnxcar/profile/theme`
- `/pps/qnxcar/profile/user`
- `/pps/qnxcar/themes`

QDB

- `/pps/qnx/dbnotify/dbs`
- `/pps/qnx/qdb/config/<dbname>`
- `/pps/qnx/qdb/status/<dbname>`

Radio

- `/pps/radio/command`

- `/pps/radio/status`
- `/pps/radio/ti_control`
- `/pps/radio/ti_rds`
- `/pps/radio/ti_status`
- `/pps/radio/tuners`

Software Update

- `/pps/services/update/control`
- `/pps/services/update/settings`
- `/pps/services/update/status`
- `/pps/services/update/target`

Speech

- `/pps/services/asr/control`

System

- `/pps/qnxcar/system/info`
- `/pps/services/appinst-mgr/control`
- `/pps/services/appinst-mgr/status`
- `/pps/services/bootmgr/`
- `/pps/services/hmi-notification/control`
- `/pps/services/hmi-notification/Messaging`
- `/pps/services/hmi-notification/Status`

Vehicle settings

- `/pps/qnxcar/hvac`
- `/pps/qnxcar/locale`
- `/pps/qnxcar/sensors`
- `/pps/qnxcar/system/settings`

Wi-Fi

- `/pps/services/tethering/control`
- `/pps/services/tethering/status`
- `/pps/services/wifi/control`
- `/pps/services/wifi/status`

- `/pps/system/navigator/status/mobile_hotspot`
- `/pps/system/navigator/status/tethering`

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.

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 typically use a forward slash (/) as a delimiter in 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
Overview of the PPS Service

The services layer of the QNX CAR Platform for Infotainment is built on the QNX *Persistent Publish/Subscribe* (PPS) service, a simple filesystem-based facility that provides persistence across reboots. Small and extensible, PPS allows interfacing from almost any higher-level language that can support open, read, write, and close operations on files.

For a more in-depth description of PPS, see the *Persistent Publish/Subscribe Developer's Guide*.

Key concepts

Objects

Objects are implemented as files under the `/pps` directory. Your applications use objects to communicate with each other. There can be many objects in the system, but never more than one instance of the same object.

Applications often use a *control* object for sending commands and a corresponding *status* object for publishing responses.

Applications can read a single special object (`.all`) to get notifications of changes to *all* the objects in a directory. Apps can use the special `.notify` object to get changes for a certain set of objects.

Attributes

Objects contain attributes or properties that apps can modify. Each attribute appears on a single line in the object file.

Publishers

As publishers, apps can modify objects and their attributes so that other interested apps can receive updates. Publishing is asynchronous—apps don't have to wait for the publisher.

To publish to an object, the publisher calls *open()* for that object and then *write()* to modify it. Multiple publishers can publish to the same object. When a publisher changes an object, the PPS service informs all subscribers of the change.

Subscribers

As subscribers, apps receive updates for objects and attributes that publishers have modified. To get updates for an object, a subscriber calls *open()* for that object and then *read()* to query it. Note that reads are *nonblocking* by default. Multiple subscribers can subscribe to the same object.

The same app can be a publisher, a subscriber, or both.

Full subscription mode

In full mode (the default), the subscriber gets a “snapshot” of the entire object as it exists *when the request is made*. Note that if a publisher changes the object many times, the subscriber may miss some of the changes. Full mode is useful, for instance, for high-bandwidth objects that have numerous and frequent changes.

Delta subscription mode

In delta mode, the subscriber gets only the changes made to an object. On first read, the subscriber will get *all* the object's attributes (because the subscriber knows nothing yet about the object's state); subsequent reads will return only the changes since the previous read. Delta mode is useful, for instance, when you want to receive all the warnings or error messages that might be published to an object.

Persistence

PPS maintains objects in memory while it's running and can save them to persistent storage (either at shutdown or on demand) on any reliable filesystem, such as flash or hard disk. Objects can be restored immediately on startup or on first access.

Server objects

A publisher can designate itself as a *server* for a particular object. When an app writes to a server object, only the publisher will get the message. PPS appends a unique identifier to the object name so that the publisher knows which client app is sending the message. For details, see “Server objects” in the *Persistent Publish/Subscribe Developer's Guide*.

Command-line options for the PPS service

```
pps [-A file][-b][-C][-d backlog][-l argument][-m mount][-p dir]
    [-P prio][-t period][-T tolerance][-U uid:gid][-v]
```

-A ***file***

Set path to ACL configuration file. For details, see “Access Control List configuration file” in the *Persistent Publish/Subscribe Developer's Guide*.

-b

Don't run in the background (useful for debugging).

-C

Convert between -U and non-U persistence formats.

-d ***backlog***

Set size of delta backlog (default is 256 kilobytes).

-l ***argument***

Set load behavior:

- 0 — load directory names and objects on demand (default).
- 1 — load directories at startup, but objects on demand.
- 2 — load directories and objects at startup.

-m ***mount***

Specify the mountpoint for PPS (default is `/pps`).

-p ***dir***

Specify the directory for persistent storage (default is `/var/pps`).

-P ***prio***

Set the priority of the persistence thread.

-t ***period***

Set the time period (in milliseconds) for writing to persistent storage (default is off).

-T ***tolerance***

Set the tolerance (in milliseconds) for writing to persistent storage (default is off).

-U ***uid*:*gid***

Downgrade from `root` to the specified UID and GID.

-v

Run in verbose mode (use multiple `v`'s to increase verbosity).

You can also use SIGUSR1 to increase verbosity.

Pathname options

PPS lets you use various pathname options when opening objects. An option must follow a question mark (?). Use a comma to separate multiple options. For example, opening the `playlist` object like this:

```
/pps/media/playlist?wait,delta
```

will open the object with the `wait` and `delta` options.

`backlog`

Total delta size to keep before flushing this OCB (Open Control Block).

`cred`

Output the credentials for this object.

`critical`

Designate the publisher as critical to the object.

`crypt`

Set the crypto domain for this object.

`delta`

Open the object in delta mode.

`deltadir`

Return the names of all objects in the `.all` object in a directory.

`f=<`*attrspec*`>{+<`*attrspec*`>}...`

Filter notifications based on changes to the names and/or values of specified attributes, where *attrspec* can be either an attribute's name or an expression specifying an attribute's value. Here's the syntax for a value expression:

`<attr_name><operator><value>`

- Operators for integers (which must be in the range of a long long) are: `<`, `<=`, `>`, `>=`, `=`, `==`, and `!=`
- Operators for strings are: `=`, `==`, and `!=` (you can use `+` if escaped with `\`)

`flow`

Treat the object as a *server object*, with purge and overflow notifications.

`hiwater`

Flow high-water mark as percent of client backlog. If this tag isn't specified, the default (`100`) is used.

`nopersist`

Make the object nonpersistent.

`notify=`*id*:*value*

Associate the object with the notification group specified by *id*:*value*, where:

- *id* is the string returned by the first read from the `.notify` object
- *value* is any arbitrary string

`opens`

Update an *_opens::rd,wr* attribute when the open count changes.

`reflect`

Reflect attribute changes made on this object back to itself.

`server`

Designate the publisher as a *server* for the object.

`verbose`

Set the verbosity level for this object.

`wait`

Clear the `O_NONBLOCK` flag so that *read()* calls will wait until the object changes or a delta appears.

Object format

Objects appear as files and directories in the PPS filesystem. For example, to view the contents of an object called `AA:BA:19:B2:AA:70` (in this case the filename is a device's MAC address) under the `/pps/services/bluetooth/remote_devices/` directory, you can simply use `cat` at the command line:

```
cat /pps/services/bluetooth/remote_devices/AA:BA:19:B2:AA:70
```

The object's contents might look like this:

```
@AA:BA:19:B2:AA:70
[n]cod::0x007a020c
```

```
[n]name::My mobile
[n]paired:b:false
[n]rssi::0x00
```

The first line always begins with an at sign (`@`), immediately followed by the object's name. Each line after that can begin with a qualifier, followed by an attribute name, followed by its encoding, followed by its value. For example, this line:

```
[n]paired:b:false
```

means that the nonpersistence qualifier (`[n]`) has been set and that the attribute `paired` has the Boolean value of `false`.

For details on encodings and on qualifiers, see these sections in the *Persistent Publish/Subscribe Developer's Guide*:

- “ Attribute syntax”
- “Object and attribute qualifiers”

Format for messages to server objects

msg::*command_string*\nid::*ID_number*\ndat:json:{*JSON_data*}

where:

command_string

Name of the command being sent to the object.

ID_number

Any ID that identifies this instance of the message. The server always reflects the ID back in the response.

JSON_data

The `dat` is usually JSON encoded, because it may contain more than a simple string.

Format for responses

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

res::*command_string*\nid::*ID_number*\ndat:json:{*JSON_data*}\nerr::*errno_number*\nerrstr::*error_description*

Changing the directory for persistent storage

The root PPS object tree (`/pps` by default) may look something like this:

```
# pwd
/pps
# ls -1F
accounts/
applications/
qnx/
qnxcar/
services/
system/
#
```

PPS populates its root object tree from the *persistence* tree (`/var/pps` by default), where the objects and attributes that you want to persist are stored.

To specify a different directory for persistent storage:

1. Create your own persistence directory (e.g., `mkdir /myobjects`).
2. Start the PPS service from a different mountpoint (e.g., `/fs/pps`) and specify your new persistence directory:

   ```
   pps -m /fs/pps -p /myobjects
   ```

You may want to run PPS with the `-t` option, which lets you specify the time period (in milliseconds) that the service will use to write to persistent storage. Without the `-t`, you won't see any changes in your persistence directory until PPS exits.

Chapter 2 Setting Up Your Own Objects

Overview

Creating a PPS object is as easy as making an *open()* call on a file under `/pps` with the `O_CREAT` flag, which will create the PPS object if it doesn't already exist. Opening, closing, reading from, and writing to PPS objects will use the exact same mechanisms as opening, closing, reading from, and writing to files on the filesystem. As shown in "Overview of the PPS Service" in this guide, as long as the data you write conforms to the format PPS expects, you can write anything to your PPS objects.

We recommend that you use the `libpps` API for encoding/decoding PPS data. These library functions make handling data easier, faster, and more reliable than using standard `libc` functions. For more information, see "PPS API reference" in the *Persistent Publish/Subscribe Developer's Guide*.

Guidelines

You could design your program to interact with PPS objects in any variety of ways. Your design will include decisions such as whether to read objects in delta mode, how frequently to read, what data to write, whether or not you receive notifications in the form of pulses, and so on. Even more decisions come into play if you're designing a system that communicates through PPS using *server objects*.

Here are the basic steps for setting up your own PPS objects, whether you're designing a program that interacts with PPS objects or adding that capability to an existing program:

1. Make sure your program includes the `<fcntl.h>` and `<sys/pps.h>` header files.
2. Open the PPS object as if it were a file. For example, to make an open call on an existing object:

   ```
   open("/pps/myobject", O_RDWR);
   ```

   This will open `myobject` with read and write privileges.

   If you're creating a PPS object that doesn't already exist, include the `O_CREAT` flag:

   ```
   open("/pps/an-object", O_RDWR | O_CREAT);
   ```

   Here we're including both `O_RDWR` and `O_CREAT` in one field with the bitwise OR operation.

3. If you need to make a new directory, you can use the *mkdir()* function. For example, to create a directory called `myservice` under `/pps/services/`:

   ```
   mkdir("/pps/services/myservice", S_IWUSR | S_IWGRP | S_IWOTH |
   S_IRUSR | S_IRGRP | S_IROTH);
   ```

   This will make your new directory with read and write privileges for all users.

4. Now you probably want to perform a read or write. Remember to use the *pps_encoder_*()* and *pps_decoder_*()* functions for handling your data.
5. Eventually you'll need to close the PPS object before your program terminates.

Interacting with your PPS objects

The basic "building blocks" you'll use for interacting with PPS objects are relatively few:

- *open()*
- *read()*
- *write()*
- *close()*
- *pps_encoder_*()*
- *pps_decoder_*()*
- `delta` mode
- `wait` mode

But you'll find many possibilities of combining these together, combining them with synchronization techniques (mutual exclusion locks, condition variables, etc.), and employing various ways to perform the same tasks. Again, see the *Persistent Publish/Subscribe Developer's Guide* for guidance.

How you'll use mutexes and other synchronization tools is up to you and depends on the needs of your program. As you'll see in the "gears" example below, mutexes are used to ensure coherency between two parallel threads: one is reading new data from PPS while the other is using existing data to draw the gears. In this case, mutexes are used so that one thread doesn't try to change attributes that the other thread is trying to use. Naturally, the synchronization needs of your programs may be different.

OpenGL "gears" example

The QNX CAR platform includes the popular OpenGL ES 2.0 "gears" demo, which runs as a sample app in the HMI. Besides providing a useful example of a 3D graphics program on an embedded system, our version of the gears demo also supports PPS. The source code is available, so you can use it as a reference for building your own PPS objects.

We have modified the gears program to create two PPS objects:

- `/pps/services/gears/control`
- `/pps/services/gears/status`

The control object lets you change the size of the window, the speed of the gears, and other properties, while the status object publishes the animation frame rate.

SVN repository for demo source

The source code for the gears demo projects is available here:

`http://community.qnx.com/svn/repos/qnxcar2_platform/2.0/src/sample/`

Under the `sample` directory you'll find three subdirectories for different versions of the gears program:

Directory	Description
`gles2-gears`	The classic OpenGL gears demo for embedded systems. The demo draws the animated gears on the screen and prints the frame rate.
`gles2-gears-pps`	The same gears demo, but modified for PPS. The program creates and uses the `/pps/services/gears/control` and `/pps/services/gears/status` objects.
`gles2-gears-pps-html`	The same PPS version of the demo, but with HTML5 support. The HTML5 elements show how your HMI can write to the control object and see the changes on the screen.

Note that the `gles2-gears-pps-html` program contains a C-language part as well as a WebWorks part, which is located here:

`http://community.qnx.com/svn/repos/qnxcar2_platform/2.0/html5/webworks/apps/HTMLGears/`

The `gears.h` header for `gles2-gears-pps`

Let's start with `gears.h`, a header file for the `gles2-gears-pps` demo. We set up this header for these key operations:

1. Include `<sys/pps.h>` to provide access to functions in the `libpps` library.
2. Define `PPS_DIR` for the directory for the status and control objects (`/pps/services/gears/`).
3. Define `PPS_STATUS_PATH` for the status object (`/pps/services/gears/status`). The path to the control object (`/pps/services/gears/control`) is defined as a macro in the `pps.c` file. Note that the file descriptors for the status and control objects are saved as global

variables after the files are opened. The status object is opened in the *pps_init()* function; the control object is opened in the *ctrl_pps_monitor_thread()* function.

4. Create a macro for writing to the status and control objects:

```
#define write_pps_buf(buf, strFmt, fd, val)\
 flushall();\
 sprintf(buf, strFmt, val);\
 write(fd, buf, strlen(buf));\
 flushall();
```

5. List the attributes used in the control object:

```
enum {
 PPS_GEARS_ATTR_OBJNAME,
 PPS_GEARS_ATTR_INTERVAL,
 PPS_GEARS_ATTR_PAUSE,
 PPS_GEARS_ATTR_SCREEN_GROUP,
 PPS_GEARS_ATTR_ACTIVATED,
 PPS_GEARS_ATTR_X,
 PPS_GEARS_ATTR_Y,
 PPS_GEARS_ATTR_W,
 PPS_GEARS_ATTR_H,
 PPS_GEARS_ATTR_Z,
 PPS_GEARS_ATTR_LAST
}
```

6. Set up a mutex to coordinate access to the *interval* variable (for the swap interval):

```
pthread_mutex_t pps_mutex;
```

7. Set up condition variables:

```
//For waiting for valid screen group
 pthread_cond_t  screenGroup_cond;

//For waiting to be activated
 pthread_cond_t  activated_cond;

//For pause/unpause
 pthread_cond_t  pause_cond;
```

8. Set up file handles:

```
int fd_pps_status;
int fd_pps_ctrl;
```

9. Set up these functions (implemented in `pps.c`) for access by other files:

```
// pps.c
extern void *ctrl_pps_monitor_thread(void * arg);
extern void pps_init(gears_t *gr);
```

The `pps.c` file for `gles2-gears-pps`

As mentioned previously, the `pps.c` file sets up the path to the control object:

```
#define PPS_CTRL_PATH "/pps/services/gears/control?wait,delta"
```

The `pps.c` file also sets up these functions:

pps_init()

Creates and opens the status object for publishing the frame rate. If the object doesn't already exist, PPS creates it before the *open()* call returns. The *pps_init()* function also spawns a thread to run the *ctrl_pps_monitor_thread()* function.

ctrl_pps_monitor_thread()

Forever looping, this function opens the `/pps/services/gears/control` object, reads from it, and changes the demo's global settings variables in response to changes in the object. If the control object doesn't already exist, PPS creates it before the *open()* call returns. Note that the control object is opened with the `?wait` and `?delta` flags (together as `?wait,delta`). This is done so that the *read()* call will block until a change is made to the object since the last read, thus returning only the attributes that have changed.

psparse()

A custom function to tokenize attributes read from a PPS object.

The `libpps` library includes a set of encoder/decoder functions, so you won't need to write your own parsing functions. For details, see “PPS encoding and decoding API” in the *Persistent Publish/Subscribe Developer's Guide*.

The `gles2-gears.c` file for `gles2-gears-pps`

We modified the original `gles2-gears.c` file for PPS support as follows:

1. The `main(int argc, char *argv[])` function calls the *pps_init()* function. The main thread then loops over the functionality outlined in the following steps. Note that this loop isn't strictly due to PPS (a draw loop is needed in any case).
2. The main thread waits for a window group to be set in the *ctrl_pps_monitor_thread()* function with a value read from the control object.
3. The main thread attempts to join the window group. If this fails, it publishes a value of `""` (the empty string) for the *screenGroup* attribute in the control object.

4. If the main thread detects that the *pause* variable is set to true (`1`), the thread waits on a condvar until *ctrl_pps_monitor_thread()* signals and updates *pause* to false.
5. The main thread continues to read global parameters that could have been changed by *ctrl_pps_monitor_thread()* and then updates its own working variables based on them. If the main thread encounters an error while assigning updated global variables to its local copies, it will publish the attributes and values that it's using (instead of the new values) to the control object.
6. The main thread then draws the gears (which has nothing to do with PPS).
7. The main thread swaps draw buffers. If there's an error, the thread publishes a value of `""` (the empty string) for *screenGroup* in the control object.
8. Finally, the main thread publishes the frame rate to the status object, using the rate calculated from the last draw cycle.

HTMLGears

The HTMLGears app is the HTML5 version of our gears demo. HTMLGears includes these JavaScript files:

- `client.js`—the client interface
- `gears.js`—the abstraction layer
- `index.js`—the extension interface

The `gears.js` file deals with the PPS activities of the HTMLGears demo. The code is as follows:

```
var _pps = require("lib/pps/ppsUtils"),
    _gearsCtrlPPS;

/**
 * Exports are the publicly accessible functions
 */
module.exports = {

    init: function () {
        _gearsCtrlPPS = _pps.createObject();
        _gearsCtrlPPS.init();
        _gearsCtrlPPS.open("/pps/services/gears/control", JNEXT.PPS_WRONLY);
    },

    /**
     * Sets the parameters for the Gears application
     */
    setParams: function(args) {
        // otherwise uses application defaults
        _gearsCtrlPPS.write({x:args.x, y:args.y, w:args.w, h:args.h});

        // otherwise uses no screen group and appears fullscreen,
        // on top of all other windows
        if (typeof args.screenGroup != 'undefined') {
            _gearsCtrlPPS.write({screenGroup:args.screenGroup});
        } else {
            rc = false;
        }
        return rc;
    },

    /**
     * Writes the activation command to the pps object
     *
     */
    start: function() {
        _gearsCtrlPPS.write({activated:1});
    },

    /**
     * Writes the pause command to the pps object
     *
     */
    stop: function() {
        _gearsCtrlPPS.write({activated:0});
    }
};
```

Let's look at the key parts:

- The file links to the PPS utilities file `ppsUtils.js` (which resides under the `Framework/lib/` directory). This file defines the PPS-related functions for the JavaScript interface.
- These lines create and initialize a JavaScript object that can then interact with the PPS control object:

  ```
  _gearsCtrlPPS = _pps.createObject();
  _gearsCtrlPPS.init();
  ```

- The JavaScript object opens `/pps/services/gears/control` as a write-only object.
- The `setParams: function(args)` function sets the window parameters (*x* dimension, *y* dimension, width, height, and *screenGroup*). When the HTMLGears app runs, the **Window Group** field in the display will show the *screenGroup* value (e.g., `1-2412607-BlackBerry-window-group`).
- These lines:

  ```
  _gearsCtrlPPS.write({activated:1});
  _gearsCtrlPPS.write({activated:0});
  ```

  will write the values for the *activated* attribute to the PPS control object. In the HTMLGears app, the **Start** and **Stop** buttons reflect the `activated:n:1` and `activated:n:0` values shown in the control object. Here's a sample control object:

  ```
  # cat /pps/services/gears/control
  @control
  activated:n:1
  h:n:395
  screenGroup::1-2412607-BlackBerry-window-group
  w:n:800
  x:n:0
  ```

```
  y:n:0
  #
```

Here's the basic pattern for creating a PPS object in JavaScript:

```
this.applicationPPS = new JNEXT.PPS();
this.applicationPPS.init();
this.applicationPPS.open(APPLICATION_PPS, "4");
```

Chapter 3
PPS Objects Reference Pages

PPS objects in alphabetical order

Each PPS object has its own reference page. The title of each page is the object's filename (e.g., `/pps/qnxcar/hvac/`).

/pps/accounts/

Directory that third-party applications use as their sandbox

This directory serves as a sandbox for third-party applications. When an app is launched, PPS will create these subdirectories:

- `/pps/accounts/1000/`*`vendor`*
- `/pps/accounts/1000-corp/`*`vendor`*

/pps/applications/mediaplayer

Object to control multimedia playback

Publishers

Any app

Subscribers

Media Player

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Overview

You can control media playback from PPS objects by using the `/pps/applications/mediaplayer` object. This will be the entry point for voice control, external physical buttons (e.g., on the steering wheel), or other means to remotely control the mediaplayer.

Request format

Commands sent to the `/pps/applications/mediaplayer` object are of the form:

```
req:json:{"id":ID_number, "cmd":"command_string"}
```

The *ID_number* is a unique identifier that will be reflected in the response from the PPS service to your request. You can set the ID to any number you wish.

Commands

The multimedia app must be running for these commands to work.

```
req:json:{"id":1, "cmd":"stop"}
```

```
req:json:{"id":1, "cmd":"pause"}
```

```
req:json:{"id":1, "cmd":"next"}
```

```
req:json:{"id":1, "cmd":"prev"}
```

```
req:json:{"id":1, "cmd":"play"}
```

```
req:json:{"id":1, "cmd":"play", "data": {"type":"fid", "id":1}}
```

```
req:json:{"id":1, "cmd":"play", "data": {"type":"artist", "id":1}}
```

```
req:json:{"id":1, "cmd":"play", "data": {"type":"album", "id":1}}
```

The data-level ID field can be the *fid*, *artist_id*, or *album_id* from the `mmlibrary.db` database. You can obtain these by using the queries in the `MMLibrary` class included with the mediaplayer app.

Examples

Stop the currently selected song:

```
echo 'req:json:{"id":1, "cmd":"stop"}' >> /pps/applications/medi
aplayer
```

Play the next song in the list:

```
echo 'req:json:{"id":1, "cmd":"next"}' >> /pps/applications/medi
aplayer
```

/pps/applications/weathernetwork/

Directory for The Weather Network (TWN) application

This directory contains a sample object (`demo-commands`) for your reference.

Publishers

TWN

Subscribers

Any app

Sample demo-commands object

Attribute	Data type	Description
drivingDirection	String	Current direction (0 - 360^o)
lat	String	Latitude
long	String	Longitude

Examples

Change the location to New York City:

```
# echo lat::40.71 >> /pps/applications/weathernetwork/demo-commands
```

```
# echo long::-74.00 >> /pps/applications/weathernetwork/demo-com
mands
```

/pps/qnx/dbnotify/dbs

Object for media database notification

Publishers

QDB

Subscribers

Any app

Overview

This object is used for database change notifications (e.g., artwork sync). For example, when a new song is selected, an artwork sync component wakes up and fetches the appropriate artwork for the selected song.

Here's a sample object:

```
# pwd
/pps/qnx/dbnotify
# ls -a
dbs
# cat dbs
@dbs
[n]db_mme::1
#
```

/pps/qnx/device/<device>

Directory of status objects for devices attached to the head unit

Publishers

Device publishers (e.g., `usblauncher`)

For more information about all the device publishers and how they interact with PPS, see the *Device Publishers Developer's Guide*.

Subscribers

Any app

Overview

When USB sticks are connected to the computer, PPS objects appear under the `/pps/qnx/device/` directory to provide status information and control. For USB devices, object names are of the form:

`usb-`*`bus_number`*`.`*`device_number`*

If `usblauncher` is called with the `-S` option, then the object name will also include the *stackno* attribute before the *bus_number* (e.g., `usb-0.0.3`).

For device control objects, see `/pps/qnx/device/<device>_ctrl`.

Sample USB object

```
[n]@usb-0.0.3
bus::USB
busno::0x00
configuration::1
configurations::1
device_class::0xff
device_protocol::0x00
device_subclass::0xff
devno::0x03
drivers_matched::1
drivers_running::1
manufacturer::D-Link Corporation
max_packet_size0::64
product::DUB-E100
product_id::0x3c05
serial_number::000001
stackno::0
topology::(1,3),(0,0)
upstream_device_address::1
```

```
upstream_host_controller::0
upstream_port::3
upstream_port_speed::High
vendor_id::0x2001
```

Attributes for USB objects

Attribute	Description
bus	Type of bus (`USB`).
busno	Bus number (in hex).
configuration	The device's USB configuration that is selected before launching suitable drivers.
configurations	Number of configurations.
device_class	Device class ID (in hex).
device_protocol	Device protocol (present only if *device_class* is not `0x00`).
device_subclass	Device subclass ID (present only if *device_class* is not `0x00`).
devno	Device number.
drivers_matched	Number of drivers that match the device, based on rules in the `usblauncher` configuration file (`0` means the device is unsupported). For details about these rules, see "Configuration files" in the *Device Publishers Developer's Guide*.
drivers_running	Number of drivers launched for the device, which can be less than *drivers_matched* if some drivers haven't been started yet or have terminated (possibly in error).
manufacturer	Device manufacturer.
max_packet_size0	Maximum packet size.
product	Product name (e.g., DUB-E100).
product_id	OEM product ID (in hex).
serial_number	Product serial number.
stackno	The USB stack number as specified with the `-S` command-line option for `usblauncher` (e.g., `usblauncher -S 0`).
status	Device status. Normally, this field is present only when the device is being reset; the field is deleted when the reset completes. For "bad" devices (i.e., devices that `io-usb` couldn't assign a device number), this field will contain: `48 (Not supported)`. When an overcurrent condition is detected, this field will contain: `-1 (Overcurrent)`.
topology	Duplicates the immediate upstream device and port numbers (*`devno,upstream_port`*) and also provides the upstream information for the hub chain. For example, `topology::(2,1),(0,2)` indicates the device is attached to hub device 2 port 1, which in turn is connected to root port 2.

Attribute	Description
upstream_device_address	USB address where the device is connected. When connected to a host controller, this is `0`; when connected to a USB hub, this is the hub's device address.
upstream_host_controller	Host controller number (usually `0`). If you have multiple USB controllers, this is the number of the controller that detected the device.
upstream_port	Port number (hub port number or `0` if connected to host controller).
upstream_port_speed	Port speed: • `Full` • `High` • `Low`
vendor_id	Manufacturer ID (in hex).

/pps/qnx/device/<device>_ctrl

Control object for devices

Publishers

Any app

Subscribers

Device publishers (e.g., `usblauncher`)

For more information about all the device publishers and how they interact with PPS, see the *Device Publishers Developer's Guide*.

Overview

When you start `usblauncher`, the following PPS object is created:

`/pps/qnx/device/usb_ctrl`

This object allows apps to perform actions on the USB hardware. Note that this type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

The control object's name also includes the *stackno* if the `usblauncher` service is started with the `-S` option (which allows for multiple server objects, one for each instance of `usblauncher`). For example, if you issue the following command:

```
usblauncher -S 1
```

the USB service creates a PPS object named `/pps/qnx/device/usb-1_ctrl`.

Commands for USB control object

Applications can send the following commands to the `/pps/qnx/device/usb_ctrl` object:

Command	Description
`port_power`	Set a hub's power state. The command takes the form: `port_power::`*`busno,devno,power_state`*`[,`*`portno`*`]`

Command	Description
	If the optional *portno* is omitted, then all ports on that hub are controlled by the power level. The *power_state* can be one of `0` ("off") or `1` ("on").
`toggle_port_power`	Turn a port's power off and back on after a fixed delay. The command takes the form: `toggle_port_power::`*`busno,devno,portno`* The numbers for *busno*, *devno*, *portno* can be decimal, hex, or mixed. For example, all three of these forms are valid: `toggle_port_power::0,10,3` `toggle_port_power::0x0,0xa,0x3` `toggle_port_power::0x0,0xa,15`

Note that the `port_power` command is useful to repower the port if the upstream hub has disabled it (e.g., as a result of an overcurrent condition and you wish to see if the overcurrent condition still applies).

Disabling the port's power doesn't cause a "removal" event of the downstream device as you might expect, but when it's reenabled, you'll then see a removal event immediately followed by an insertion event.

Responses from `usblauncher`

When `usblauncher` executes commands, it publishes these attributes to the control object:

Attribute	Description
`port_power`	Latest power setting for the port. Values: • `1` ("on") • `0` ("off") • `-1` ("unknown")
`cmd_status`	String giving the *errno_number* and error condition (see *below* (p. 44)).

USB control examples

After starting the `usblauncher` process as usual, enter this command from a terminal:

```
cat /pps/qnx/device/usb_ctrl?wait
```

Then from a second terminal, enter these commands:

```
sloginfo -w &
```

```
echo toggle_port_power::xx,y,z >>
/ramdisk/pps/qnx/device/usb_ctrl
```

The first terminal (`cat usb_ctrl?wait`) will show the command status and the power result of the command for the specified bus, device, and port.

For example:

```
# cat usb_ctrl?wait,delta
@usb_ctrl
port_power::0
@usb_ctrl
port_power::1
@usb_ctrl
cmd_status::0
```

A value of `0` for `cmd_status` means no errors.

Possible error conditions

Here is a subset of possible errors:

Command status	Meaning
`cmd_status::19`	No such device.
`cmd_status::5`	Could not set feature. For example, the specified port number is greater than the maximum number of available ports.
`cmd_status::48`	Not supported. For example, the specified device isn't a hub.

If a command isn't recognized, an error message such as the following will be published to *stdout* or to `sloginfo` (if `usblauncher` was started with the `-l` option):

```
CMD: unknown cmd_name
```

/pps/qnx/driver/<pid>

Directory for device driver objects

Publishers

Device publishers (e.g., `usblauncher`)

For more information about all the device publishers and how they interact with PPS, see the *Device Publishers Developer's Guide*.

Subscribers

Any app

Overview

When a USB stick or other device is connected to the computer, a PPS status object appears under the `/pps/qnx/driver/` directory to report details of the connected device's driver. The object name is the driver's process ID.

Here's a sample object:

```
[n]@2818054
PPS_DEVICE_ID::/pps/qnx/device/usb-1.4
arguments::cam quiet blk cache=1m,vnode=384,auto=none,delwri=2:2,
rmvto=none,noatime disk name=umass cdrom name=umasscd
dos exe=all umass priority=21,
vid=0x0951,did=0x1625,busno=0x01,
devno=0x04,iface=00,ign_remove
interface::0
interface_class::0x08
interface_protocol::0x50
interface_subclass::0x06
name::devb-umass
pid::2818054
```

Attributes

Attribute	Description
arguments	A copy of the command-line arguments that were given to the device driver.
interface	Interface number.
interface_class	Class ID (in hex).
interface_protocol	Protocol number (in hex).

Attribute	Description
interface_subclass	Subclass ID (in hex).
name	Process name of the driver.
pid	The driver's process ID.
PPS_DEVICE_ID	Path to the device object.

/pps/qnx/mount/<device>

Directory for mounted devices

Publishers

Device publishers (e.g., `usblauncher`)

For more information about all the device publishers and how they interact with PPS, see the *Device Publishers Developer's Guide*.

Subscribers

Any app

Overview

The `/pps/qnx/mount/` directory contains objects that hold information for all devices mounted in the system. Object names are of the form:

rawdevice[*.partition#*]

For example, for a USB stick (`/dev/umass0`) with a DOS partition, the PPS objects would be named:

`/pps/qnx/mount/umass0`

`/pps/qnx/mount/umass0.0`

Here's a sample USB object:

```
[n]@umass0.0
PPS_DRIVER_ID::/pps/qnx/driver/2052107
PPS_RAWMOUNT_ID::/pps/qnx/mount/umass0
blocks_size::512
blocks_total::7830408
fs_type::dos (fat32)
id::6485a02e-4cd0-4ed6-80a1-a0bce5acde3e
label::KINGSTON
mnt_status::0 (No error)
mount::/fs/usb0
name::KINGSTON
partition::/dev/umass0t11
partition_order::0
plugin_name::generic
raw::/dev/umass0
read_only::0
```

Attributes for USB objects

Attribute	Description
blocks_size	Size of each block (in bytes).
blocks_total	Total number of blocks.
fs_type	Filesystem type: • `cd` • `dos (fat32)` • `ipod` • `iso9660` • `joliet` • `pfs` (e.g., for MTP devices) • `qnx` • `udf` • `unknown` The specific values for *fs_type* depend on the relevant filesystem driver. For details about each driver, see the fs-* entries in the OS *Utilities Reference*.
id	Device identifier.
label	Partition label.
mnt_status	Error string indicating the outcome of the mount operation (e.g., `48 (Not supported)`).
mount	Filesystem mountpoint of the device (e.g., `/fs/usb0`).
name	Name given to the device.
partition	Device partition (e.g., `/dev/umass0t11`).
partition_count	Number of partitions on this raw device.
partition_order	Order in the partition table.
plugin_name	Name of the plugin used for this device.
PPS_DRIVER_ID	Name of the PPS object that contains information about the device driver.
PPS_RAWMOUNT_ID	Name of the PPS object that contains information about the raw device.
raw	Mountpoint of the raw device.
read_only	Indicates (`1` for true, `0` for false) whether this is a read-only device.
status	Error string (present only if an error occurred).

The Juke Box app happens to create the following object under the `/pps/qnx/mount/` directory:

```
@mme
device_type::hdd
fs_type::qnx
id::mme
mount::/accounts/1000/shared/
name::Juke Box
```

Note that the *device_type* attribute and `hdd` value here are specific to the QNX CAR platform.

/pps/qnx/qdb/config/<dbname>

QDB parses this object to set up a database

Publishers

Any app

Subscribers

QDB

Overview

For every loaded database, the `/pps/qnx/qdb/config` directory contains a PPS object with the same name as the database (e.g., `/pps/qnx/qdb/config/bluetoothdb`).

Configuration parameters

For more information about these parameters, see "Database configuration objects" in the *QDB Developer's Guide*.

Parameter	Description
AutoAttach	Specifies other databases to attach to the current one (using the SQL `ATTACH DATABASE` statement).
BackupAttached	Sets this operation as the default for attached databases when a command is issued to the main database.
BackupDir	Specifies the directories for storing database backups.
BackupVia	Specifies an interim directory to copy a database as part of the backup.
ClientSchemaFile	Names the client schema file (with an absolute path) that contains the SQL commands to run whenever a client calls *qdb_connect()*.
Collation	Installs user-provided collation (sorting) routines.
Compression <option>	Specifies the compression algorithm to apply to backups. Options: • none

Parameter	Description
	• `lzo` • `bzip` • `diocopy` (direct I/O copy uisng DMA)
CompressionVia `true|false`	Used with *BackupVia* and any *Compression* setting specified. Default is `false`. Set this to `true` if you want *BackupVia* to use compression during its first step.
DataSchemaFile	If *SchemaFile* is set, names the file (with an absolute path) that contains the SQL commands to populate a database when it's created.
Filename	Sets the name of the database (raw SQLite) file. This must be an absolute path.
Function	Installs user scalar/aggregate functions.
SchemaFile	Names the file (with an absolute path) that contains the SQL commands to create the initial schema of tables, indexes, and views of a new database.
SizeAttached	Sets this operation as the default for attached databases when a command is issued to the main database.
VacuumAttached	Sets this operation as the default for attached databases when a command is issued to the main database.

/pps/qnx/qdb/status/<dbname>

QDB publishes database status to this object

Publishers

QDB

Subscribers

Any app

Overview

For every loaded database, the `/pps/qnx/qdb/status` directory contains a PPS object with the same name as the database. The status object indicates the state of the database after the loading attempt.

Status values

The status file contains a *Status* attribute that can have one of these values:

Value	Description
`AttachWait`	QDB is waiting for an attached database (specified with the *AutoAttach* configuration parameter) to become available.
`Error`	The configuration contained an error.
`Initializing`	QDB has seen the configuration object and is now initializing the database.
`Valid`	The database has been configured and can be accessed.

/pps/qnxcar/hvac

This object contains HVAC settings such as fan speed, heated seat levels, etc.

Publishers

Climate Control

Subscribers

Climate Control; any app

Attributes

Attribute	Data type	Values	Default
airConditioning_all	Boolean	`true` | `false`	`false`
airRecirculation_all	Boolean	`true` (recirculate internal air) | `false` (use external air)	`false`
defrost_all	Boolean	`true` | `false`	`true`
fanDirection_row1left	String	`defrost` | `defrostAndfeet` | `face` | `faceAndFeet` | `feet`	`faceAndFeet`
fanDirection_row1right	String	`defrost` | `defrostAndfeet` | `face` | `faceAndFeet` | `feet`	`face`
fanSpeed_row1left	Number	`0` (off) to `6` (full speed)	`1`
fanSpeed_row1right	Number	`0` (off) to `6` (full speed)	`3`
heatedSeat_row1left	Number	`0` to `3`	2
heatedSeat_row1right	Number	`0` to `3`	0
temperature_row1left	Number	`15` to `26` (^{o}C)	`21`
temperature_row1right	Number	`15` to `26` (^{o}C)	`19`
zoneLink_all	Boolean	`true` | `false`	`false`

Zone Link

When enabled, the Zone Link feature links the fan temperature and speed values for both the left and right climate zones.

The left climate zone takes priority when enabling this feature, that is, the right zone will have its fan temperature and speed set to that of the left. Disabling Zone Link will once again allow both zones to be managed independently.

Examples

Set the left-zone temperature to 20 ^{o}C:

```
echo temperature_row1left:n:20 >> /pps/qnxcar/hvac
```

Set the right-zone fan speed to 4:

```
echo fanSpeed_row1right:n:4 >> /pps/qnxcar/hvac
```

Turn rear defrost off:

```
echo defrost_all:b:false >> /pps/qnxcar/hvac
```

/pps/qnxcar/locale

Object for locale settings

Publishers

HMI

Subscribers

Any app

Attributes

Attribute	Data type	Values	Description
locale	String	en (English)	Language of the UI.

/pps/qnxcar/navigation/control

Control object for the navigation service

Publishers

Any app; navigation service

Subscribers

Navigation service; any app

Message/response format

Commands sent to the `/pps/qnxcar/navigation/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}`

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}\nerr::`*`errno_number`*`\nerrstr::`*`error_description`*

Commands

msg::	id::	dat:json:
`getPOIs`	Number	• `category` (contains several fields—see table below). • `count` • `location` (contains several fields—see table below).
`cancelNavigation`	Number	Stop the current navigation session.
`navigateTo`	Number	`location` (contains several fields—see table below).
`panMap`	Number	• `deltaX` (number of pixels to move the map horizontally) • `deltaY` (number of pixels to move the map vertically)
`search`	Number	• `country`

msg::	id::	dat:json:
		• `province` • `city`
`zoomMap`	Number	`scale` (in meters/pixel)

Fields for `category`

id	name	parentId	type
`1`	Accommodations + Amenities	`0`	`accommodation`
`2`	Attractions	`0`	`attraction`
`3`	Public Places + Services	`0`	`public`
`4`	Restaurants + Entertainment	`0`	`restaurant`
`5`	Transportation	`0`	`transportation`
`6`	Hotel or Motel	`1`	`accommodation`
`7`	Airport	`5`	`transportation`
`8`	Golf Course	`4`	`sports`
`9`	Ferry	`5`	`transportation`
`10`	Restaurant	`4`	`restaurant`
`11`	Nightlife	`4`	`restaurant`
`12`	Casino	`4`	`restaurant`
`13`	Movie Theatre	`4`	`restaurant`
`14`	Community Centre	`3`	`public`
`15`	City Hall	`3`	`public`
`16`	Sports Centre	`3`	`sports`
`17`	Amusement Park	`2`	`attraction`
`18`	Museum	`2`	`museum`
`19`	Historical Monument	`2`	`attraction`
`20`	Tourist Office	`2`	`attraction`
`21`	Parking Garage	`5`	`parking`

id	name	parentId	type
22	Park & Ride	5	`parking`
23	Automobile Dealer	5	`transportation`
24	Rest Area	5	`park`
25	Hospital	1	`hospital`
26	School	1	`accommodation`
27	Police Station	3	`public`
28	Place of Worship	1	`church`

Fields for `location`

Field	Description
`city`	The destination's city.
`country`	The city's country.
`distance`	Distance from current location to destination.
`id`	Destination identifier.
`latitude`	The destination's latitude.
`longitude`	The destination's longitude.
`name`	The destination's name (e.g., Toronto City Hall).
`number`	Street address.
`postalCode`	The postal code's first three characters, which indicate the Forward Sortation Area (FSA).
`province`	The destination's province.
`street`	Name of the destination's street.
`type`	Point-of-interest (POI) type: • `accommodation` • `attraction` • `church` • `hospital` • `museum` • `park` • `parking` • `public`

Field	Description
	• `restaurant` • `sports` • `transportation`
`xindex`	This attribute is a byproduct of using the Sencha framework and is ignored when present.

/pps/qnxcar/navigation/geolocation

Geolocation object for the navigation service

Publishers

Any app

Subscribers

Navigation service

Attributes

Attribute	Data type	Description
`heading`	Number	Current direction (0 - 360^o).
`latitude`	Number	The current latitude.
`longitude`	Number	The current longitude.

/pps/qnxcar/navigation/options

Set options for the navigation service

Publishers

Any app

Subscribers

Navigation service

Attributes

Attribute	Data type	Default	Description
avoid_construction	Boolean	`false`	Indicates whether to avoid known construction areas.
avoid_ferries	Boolean	`false`	Indicates whether to avoid routes that have ferries.
avoid_motorways	Boolean	`false`	Indicates whether to avoid motorways (controlled-access highways).
avoid_seasonal_roads	Boolean	`false`	Indicates whether to avoid seasonal roads.
avoid_time_restricted	Boolean	`false`	Indicates whether to avoid time-restricted roads.
avoid_tolls	Boolean	`false`	Indicates whether to avoid toll roads.
avoid_tunnels	Boolean	`false`	Indicates whether to avoid roads that have tunnels.
avoid_unpaved	Boolean	`false`	Indicates whether to avoid unpaved roads.
daynight	String	`both`	Set map display to day mode, night mode, or both (not currently implemented).
language	String	`en_US`	Set the language.
routing	String	`fastest`	Set routing method (not currently implemented).
show_turn_restrictions	Boolean	`true`	Show turn restrictions.
units	String	`metric`	Set the units for distance (not currently implemented).
voice	String	`Rupert`	Set the voice persona (not currently implemented).

/pps/qnxcar/navigation/status

The navigation service publishes status information to this object

Publishers

Navigation service

Subscribers

Any app

Attributes

Attribute	Data type	Values	Description
destination	JSON	Contains several fields (see table below).	Data for the destination (e.g., city, country, distance).
maneuvers	JSON	Contains several fields (see table below).	Data while navigating (e.g., current street, turns).
navigating	Boolean	`true` | `false`	Indicates whether the system is currently navigating.
total_distance_remaining	Number	`>=0`	Distance to destination (in meters).
total_time_remaining	Number	`>=0`	Estimated time of arrival (in minutes).

Fields for *destination*

Field	Description
`city`	The destination's city.
`country`	The city's country.
`distance`	Distance from current location to destination.
`id`	Destination identifier
`latitude`	The destination's latitude.
`longitude`	The destination's longitude.
`name`	The destination's name (e.g., Toronto City Hall).
`number`	Street address.

Field	Description
`postalCode`	The postal code's first three characters, which indicate the Forward Sortation Area (FSA).
`province`	The destination's province.
`street`	Name of the destination's street.
`type`	Point-of-interest (POI) type (e.g., restaurant, public building).
`xindex`	This attribute is a byproduct of using the Sencha framework and is ignored when present.

Fields for *maneuvers*

Field	Description
`street`	Street currently traveling on.
`command`	Directional command. Values: • `dt` (arrived at destination) • `dt-l` (destination is on the left) • `dt-r` (destination is on the right) • `lht-rx` (follow the roundabout on the left) • `lht-ut` (make a U-turn on the left) • `nc` (no change; follow the street) • `rx` (follow the roundabout on the right) • `tr-l` (turn left) • `tr-r` (turn right) • `ut` (make a U-turn on the right)
`distance`	Distance to travel on current street (in meters).

/pps/qnxcar/profile/theme

View or modify the settings for the Personalization **Theme** field

Publishers

Personalization

Subscribers

Any app

Attributes

Attribute	Data type	Values	Default	Description
nightMode	Boolean	`true` | `false`	`false`	This feature isn't fully implemented yet. The vehicle sensor responsible for the headlights modifies this attribute. When the headlights are turned on, the current theme will be rejigged to display higher-contrast graphics for better readability at nighttime.
theme	String	The possible values for this release are as follows: • `default` • `midnightblue` • `titanium` Note that a related PPS object (`/pps/qnxcar/themes`) holds the attributes of the Personalization theme packages available on the QNX CAR platform.	`de fault`	The *theme* attribute affects the graphics assets of all registered applications. Structural changes are minimal; most changes are cosmetic (image swaps, color changes, font changes, etc.). The attribute is controlled by the **Active Theme** field in the Personalization application. The *theme* attribute can be modified in two ways: • The user changes the theme of the current profile. • The user changes to a new profile that has a different theme defined.

/pps/qnxcar/profile/user

View or change settings in the Personalization application

Publishers

Personalization; any app

Subscribers

Personalization; any app

Attributes

Attribute	Data type	Values	Default	Description
avatar	String	• `male1` • `male2` • `female1` • `female2`	`male1`	Image of the currently selected profile in the list of the Personalization application. Changes made to the *avatar* field are propagated to PPS.
bluetoothDeviceId	String	Device's MAC.	`0`	Preferred Bluetooth device.
fullName	String	User-specified name.	"Default"	The user name appears on the status bar (far-left label) in the Personalization application. Changes made to this field are propagated to PPS.
id	Number	`>=0`	`1`	Represents the currently selected profile in the Personalization application. The number changes only when a different profile is selected.
theme	String	• `default` • `midnightblue` • `titanium`	`de fault`	Preferred theme, which is selected in the Personalization **Theme** field (see `/pps/qnxcar/profile/theme` for details).

/pps/qnxcar/sensors

Get the status of various components, such as fuel level, tire pressure, etc.

Publishers

Virtual Mechanic

Subscribers

Climate Control; Virtual Mechanic

This is the only PPS object that Virtual Mechanic uses. Status values for all components are read from this object. The only case where Virtual Mechanic writes to `/pps/qnxcar/sensors` is when turning the ABS brakes setting (*brakeAbsEnabled*) on or off.

Attributes

Attribute	Data type	Values	Default	Caution status condition	Alert status condition
brakeAbsEnabled	Boolean	`true` | `false`	`true`	n/a	n/a
brakeAbsFrontLeft	Boolean	`true` | `false`	`true`	`false`	n/a
brakeAbsFrontRight	Boolean	`true` | `false`	`true`	`false`	n/a
brakeAbsRearLeft	Boolean	`true` | `false`	`true`	`false`	n/a
brakeAbsRearRight	Boolean	`true` | `false`	`true`	`false`	n/a
brakeFluidLevel	Number	0 - 100%	90	<=80	<=70
brakePadWearFrontLeft	Number	0 - 100%	100	<=40	<=20
brakePadWearFrontRight	Number	0 - 100%	100	<=40	<=20
brakePadWearRearLeft	Number	0 - 100%	50	<=40	<=20
brakePadWearRearRight	Number	0 - 100%	65	<=40	<=20
cameraRearviewActive	Boolean	`true` | `false`	`false`	n/a	n/a
coolantLevel	Number	0 - 100%	100	<=80	<=70
engineOilLevel	Number	0 - 100%	95	<=85	<=75
engineOilPressure	Number	>=0 (PSI)	100	<=85	<=75
fuelLevel	Number	0 - 100%	75	<=25	<=10

Attribute	Data type	Values	Default	Caution status condition	Alert status condition
lightHeadLeft	Boolean	`true` | `false`	`true`	`false`	n/a
lightHeadRight	Boolean	`true` | `false`	`true`	`false`	n/a
lightTailLeft	Boolean	`true` | `false`	`false`	`false`	n/a
lightTailRight	Boolean	`true` | `false`	`true`	`false`	n/a
rpm	Number	>=0	2800	>=6250	>=7000
speed	Number	>=0	0	n/a	n/a
tirePressureFrontLeft	Number	>=0 (PSI)	31	<=26, >=36	<=24, >=38
tirePressureFrontRight	Number	>=0 (PSI)	31	<=26, >=36	<=24, >=38
tirePressureRearLeft	Number	>=0 (PSI)	25	<=26, >=36	<=24, >=38
tirePressureRearRight	Number	>=0 (PSI)	32	<=26, >=36	<=24, >=38
tireWearFrontLeft	Number	0 - 100%	90	<=30	<=20
tireWearFrontRight	Number	0 - 100%	90	<=30	<=20
tireWearRearLeft	Number	0 - 100%	70	<=30	<=20
tireWearRearRight	Number	0 - 100%	70	<=30	<=20
transmissionClutchWear	Number	0 - 100%	70	<=60	<=40
transmissionFluidLevel	Number	0 - 100%	85	<=80	<=70
transmissionFluidTemperature	Number	-273.15 - 1000 (degrees F)	185	>=215	>=240
transmissionGear	String	p,r,n,d,1,2,3,4,5,6,7	p	n/a	n/a
washerFluidLevel	Number	0 - 100%	20	<=20	<=10

Examples

Set the fuel level to 15%:

```
echo fuelLevel:n:15 >> /pps/qnxcar/sensors
```

Turn ABS off:

```
echo brakeAbsEnabled:b:false >> /pps/qnxcar/sensors
```

/pps/qnxcar/system/info

Holds software build information

Publishers

coreServices

Subscribers

Any app

Attributes

Attribute	Data type	Description
buildHost	String	Build host.
buildID	String	Identifier showing the date and time of the build.
buildNum	String	Build number.
car2Branch	String	Branch in the repository.
car2Rev	String	Revision number.
date	String	Date and time stamp of the build.
platform	String	Target hardware platform (e.g., `omap5uevm`).
project	String	Project name for this build.
variant	String	Variant name.

/pps/qnxcar/system/settings

Configure HMI application displays

Publishers

Any app

Subscribers

HMI

Attributes

Attribute	Data type	Values	Default	Impact

appSection_profile	String	`high`, `mid`	`high`	`high`
				• Application carousel is free-dragging and animates between categories.
				`mid`
				• Application carousel supports swipe gestures but doesn't animate between categories.

carControl_profile	String	`high`, `mid`	`high`	`high`
				• Climate Control fan dials are free-dragging.
				• Audio slider controls update their display while dragging.
				• Virtual Mechanic dialogs fade in and out when appearing and disappearing.
				`mid`
				• Climate Control fan dials use a tap gesture on the top- and bottom-third of the control to change settings.
				• Audio slider controls don't update their display while dragging.
				• Virtual Mechanic dialogs appear and disappear instantly.

communication_profile	String	`high`, `mid`	`high`	`high`
				• Pulldown submenus slide in and out from the right.
				`mid`

Attribute	Data type	Values	Default	Impact
				• Pulldown submenus don't animate when appearing and disappearing.

home_profile	String	`mmcontrol`, `mmplayer`	`mmplayer`	`mmcontrol`
				• Specify `mm-control` as the media service that the Home application should use to display Now Playing information.
				`mmplayer`
				• Specify the new `mm-player` as the media service to use.

mediaPlayer_profile	String	`high`, `mid`	`high`	`high`
				• Pulldown submenus slide in and out from the right.
				• Radio view uses a free-dragging dial to change stations.
				• Coverflow component offers enhanced visuals and interactivity.
				`mid`
				• Pulldown submenus don't animate when appearing and disappearing.
				• Radio view uses a horizontal slider to change stations.
				• Coverflow component shows only the current track's album art, but still supports swipe gestures to change tracks.

navigationProvider	String	`elektrobit`	`elektrobit`	Specify *Elektrobit (EB) street director* as the navigation engine to use as the default provider. Note that this setting will cause the JavaScript extension framework to load the appropriate provider bundle and to communicate with the appropriate PPS objects.

/pps/qnxcar/themes

View the attributes of Personalization theme packages

Publishers

Personalization

Subscribers

Any app

Overview

The `/pps/qnxcar/themes` object holds the attributes of the Personalization theme packages available on the QNX CAR platform. This object is updated to notify the system whenever a new theme has been added or removed.

These themes are currently available:

- `default`
- `midnightblue`
- `titanium`

A related PPS object (`/pps/qnxcar/profile/theme`) has a *theme* attribute that can be set to one of the available themes for the desired appearance (image swaps, color changes, font changes, etc.) of all the HMI apps. The *theme* attribute is controlled by the **Theme** field in the Personalization application.

Here's a sample `/pps/qnxcar/themes` object:

```
default:json:{"ppsName":"default","title":"Default","themePackageName":"default","packageDate":"2013-05-13"}
midnightblue:json:{"ppsName":"midnightblue","title":"Midnight Blue","themePackageName":"midnightblue",
     "packageDate":"2013-05-13"}
titanium:json:{"ppsName":"titanium","title":"Titanium","themePackageName":"titanium","packageDate":"2013-05-13"}
```

Each line gives the theme, followed by the `json` data type, followed by the attributes and values for that theme.

Attributes

Attribute	Description
ppsName	Value of the *theme* attribute in the `/pps/qnxcar/profile/theme` object.
title	Name that appears in the title field in the Personalization app.

Attribute	Description
themePackageName	Name of the theme's package file.
packageDate	Date of the theme's package file.

/pps/radio/command

The radio app listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

Radio

Commands

The control object accepts the following commands:

Command	Description
`seek` *direction*	Seek either `up` or `down`.
`seek stop`	Stop seeking.
`tune`	Tune to the specified station.
`tuner` *tuner*	Set the tuner to `am` or `fm`.

Examples

Use the FM tuner:

```
echo tuner::fm >> /pps/radio/command
```

Seek down:

```
echo seek::down >> /pps/radio/command
```

Stop seeking:

```
echo seek::stop >> /pps/radio/command
```

/pps/radio/status

View the status of radio attributes such as artist, song, etc.

Publishers

Radio

Subscribers

Any app

The object's format looks like this:

```
@status
am:json:{"presets":[880,910,950,1020,1220,1430],"station":1680}
artist::Bjork
fm:json:{"presets":[87.5,88.5,99.9,105.3,106.9,107.1],"station":96.5}
genre::News & Entertainment
hd:b:true
song::All is Full of Love
station::CBC Radio 2
tuner::fm
```

Attributes

Attribute	Data type	Description
am	JSON	Indicates the AM band and contains: • `presets`, a set of six preset AM stations • `station`, the AM station currently tuned in
artist	String	Name of the artist currently playing.
fm	JSON	Indicates the FM band and contains: • `presets`, a set of six preset FM stations • `station`, the FM station currently tuned in
genre	String	Category of the station's format.
hd	Boolean	Indicates whether HD radio is enabled.
song	String	Title of the song currently playing.
station	String	Name identifying the station.
tuner	String	Indicates the type of band (AM or FM).

/pps/radio/ti_control

Radio control object for TI Jacinto hardware

This PPS object is supplied as a reference design for your convenience.

/pps/radio/ti_rds

Radio object for TI Jacinto hardware

This PPS object is supplied as a reference design for your convenience.

/pps/radio/ti_status

Radio status object for TI Jacinto hardware

This PPS object is supplied as a reference design for your convenience.

/pps/radio/tuners

Get the status of attributes of the radio tuners

Publishers

Radio

Subscribers

Any app

The object's format looks like this:

```
@tuners
am:json:{"type":"analog","rangeMin":880,"rangeMax":1710,"rangeStep":10}
fm:json:{"type":"analog","rangeMin":87.5,"rangeMax":107.1,"rangeStep":0.2}
```

Attributes

Attribute	Data type	Description
am	JSON	Indicates the AM tuner and contains: • `type` (analog or digital) • `rangeMin` (the minimum frequency in the AM tuner's range, e.g., 880 kHz) • `rangeMax` (the maximum frequency in the AM tuner's range, e.g., 1710 kHz) • `rangeStep` (the size of the intervals between the mimimum and maximum values in the range, e.g., 10 kHz)
fm	JSON	Indicates the FM tuner and contains: • `type` (analog or digital) • `rangeMin` (the minimum frequency in the FM tuner's range, e.g., 87.5 MHz) • `rangeMax` (the maximum frequency in the FM tuner's range, e.g., 107.1 MHz) • `rangeStep` (the size of the intervals between the mimimum and maximum values in the range, e.g., 0.2 MHz)

/pps/servicedata/schedule

Sample object for the **Schedule Maintenance** feature in Virtual Mechanic

Publishers

Virtual Mechanic

Subscribers

Any app

This directory contains a sample `schedule` object used by the Virtual Mechanic application.

Attributes

Attribute	Data type	Description
Days	Number	Days since last service.
Odometer	Number	Odometer reading.
Recommended	JSON	Contains: • `service` (`recommended`) • `booked` (`false`) • `km` • `time` • `duration` (in minutes) • `cost` (in dollars) • `date` (hour, minute, day, month, year) • `type` (0 to 7)

/pps/services/app-launcher

Object to control the Applications Navigator

Publishers

Applications Navigator; any app

Subscribers

Applications Navigator; any app

Overview

You can control the Applications Navigator by sending commands to the `/pps/services/app-launcher` PPS object.

The `/pps/services/app-launcher` object must exist before the Applications Navigator starts.

Command format

Commands sent to the `/pps/services/app-launcher` object are of the form:

```
req:json:{"id":ID_number,"cmd":"com
mand_string","app":"app_string,"dat":""}
```

The *ID_number* is a unique identifier that will be reflected in the response from the PPS service to your request. You can set the ID to any number you wish.

The `dat` attribute is responsible for setting parameters that will be sent to the application. Parameters can be either strings or JSON objects.

Launching and stopping applications

At start time, the Applications Navigator publishes a list of existing applications in the PPS object's `app_list` attribute:

```
app_list:json:{apps[app_string,app_string],...]}
```

You can use PPS to launch any application given in `app_list`. For example, to launch the application named “MediaPlayer”, issue the following command:

```
echo 'req:json:{"id":1,"cmd":"launch app","app":"MediaPlay
er","dat":""}' >> /pps/services/app-launcher
```

To stop the MediaPlayer application, use the `"close app"` command:

```
echo 'req:json:{"id":1,"cmd":"close app","app":"MediaPlay
er","dat":""}' >> /pps/services/app-launcher
```

Responses

Each command issued will receive a status response. The `status` attribute will contain the ID number that you used in your command as well as any errors that may have occurred. For example:

```
status:json:{"error":"OK","id":1}
```

This response indicates that the command with ID `1` was executed successfully.

The Applications Navigator creates the PPS object `/pps/system/navigator/command` for publishing all necessary data. Applications that subscribe to this object should open it using the `?wait` option and *delta mode* so as to receive all relevant changes.

For more information on `?wait` and on delta mode, see “Subscribing” in the *Persistent Publish/Subscribe Developer's Guide*.

/pps/services/appinst-mgr/control

The `appinst-mgr` service listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

`appinst-mgr`; any app

The `appinst-mgr` service is provided as a reference design for your convenience.

Overview

The `appinst-mgr` service is used by the *QNX App Portal* client to install/uninstall apps. QNX App Portal is an application showcase where developers can submit their apps for users of the QNX CAR platform to download and evaluate on their targets.

The `appinst-mgr` service subscribes to the `/pps/services/appinst-mgr/control` object for commands that apps want to issue, and then publishes results to the `/pps/services/appinst-mgr/status` object.

Commands

Command	Description
`install`	Download and install the specified `.bar` file from the specified URL. Note that you must also specify the app's 32-character `auth token`.
`uninstall`	Uninstall the installed app specified in the `appname` field.

Examples

Install the `HelloWorld.bar` file that resides at `http://10.222.98.197` and has an authentication token of `85vD2COZBmBjDVGlR3mooOttcqysfLFb`:

```
echo "command::install\nurl::http://10.222.98.197/HelloWorld.bar\n
authtoken::85vD2COZBmBjDVGlR3mooOttcqysfLFb\n" >> /pps/services/ap
pinst-mgr/control
```

Uninstall the HelloWorld app:

```
echo "command::uninstall\nappname::HelloWorld\n" >> /pps/ser
vices/appinst-mgr/control
```

/pps/services/appinst-mgr/status

Status object for reporting the results of commands sent to the `appinst-mgr` service

Publishers

`appinst-mgr`

Subscribers

Any app

The `appinst-mgr` service is provided as a reference design for your convenience.

Overview

The `appinst-mgr` service uses this status object to publish the results of commands sent to the `/pps/services/appinst-mgr/control` object.

Attributes

Attribute	Values
msg	Error or information message.
progress	Number (percent) indicating the progress of the install/uninstall operation.
state	`install|uninstall`
status	`OK|ERROR`

Sample status objects

Successful installation:

```
@status
msg::HelloWorld.testRel_HelloWorld_1a2fa200
progress:n:100
state::install
status::OK
```

Successful *un*installation:

```
@status
msg::successfully removed
progress:n:100
state::uninstall
status::OK
```

Attempt to install a `.bar` file that doesn't exist:

```
@status
msg::unzip: cannot find zipfile directory in /tmp/temp.bar
progress:n:100
state::install
status::ERROR
```

Attempt to uninstall an invalid app (e.g., the app named "my_test" doesn't exist):

```
@status
msg::Error: Unable to determine installed application: [my_test]
progress:n:100
state::uninstall
status::ERROR
```

/pps/services/asr/control

The `io-asr` manager uses this object to communicate with the HMI

Publishers

`io-asr`; any app

Subscribers

`io-asr`; any app

Overview

When running, the `io-asr` manager updates its `state` on this object, responds to any `strobe` commands, and then goes into `listening` mode.

Attributes

Attribute	Description
result	JSON object that contains the `"confidence"` level of the speech input. Range is from 0 (input not recognized at all) to 1000 (full confidence—no other possible results for the input).
speech	Indicates input is being handled. Values: • `processing` • `handled`
state	Indicates `io-asr`'s current state. Values: • `held` • `idle` • `listening` • `processing` • `prompting` • `running`
strobe	Controls speech sessions. Values: • `barge-in` • `cancel` • `hold` • `mic-off`

Attribute	Description
	• `off` • `on` • `on?audio_log_dir=`*path* (directory to use to capture session logs) • `on?audio_source_dir=`*path* (directory of stored logs to use for recognition sessions) • `on?repeat=[true|false]` (if set to `true`, speech sessions will auto-trigger as long as another application isn't launched) If a third-party app initiated the current speech session, it will be interrupted and a new system-level session will be started. The results of a third-party session are delivered only to the requesting app and won't be used to carry out system-level speech commands. • `release` (releases a `held` speech session)

How `io-asr` responds to `strobe` commands

If `io-asr` is:	Then this command:	Has this effect:
idle	`strobe::on`	Begins a speech session.
running	`barge-in`	Interrupts any currently playing prompts, allowing the speech session to advance to the next state (usually `listening`).
running	`strobe::cancel`	Cancels the current speech session.
running	`strobe::hold`	Interrupts the current speech turn and suspends the session, maintaining its state.
running	`strobe::mic-off`	Interrupts capturing of audio so that the session can advance to `processing`; this is useful if the auto end-pointing isn't detecting silence because of ambient noise levels.
running	`strobe::on`	n/a
stopped	`strobe::off`	n/a

If io-asr is:	Then this command:	Has this effect:
running and listening	`strobe::off`	Ends the speech session. If the `repeat` modifier is in effect, it will be canceled.
running and prompting	`strobe::off`	Silences the prompt.
held	`strobe::release`	Releases a held speech session, allowing it to return to the `running` state.

Examples

Start a speech session and then record each utterance to the `speech_logs` directory:

```
echo strobe::on?audio_log_dir=speech_logs >>/pps/services/asr/control
```

Play back the recorded session:

```
echo strobe::on?audio_source_dir=speech_logs >>/pps/services/asr/control
```

/pps/services/audio/audio_router_control

The Audio Manager listens for routing commands on this control object

Publishers

Audio Manager; any app

Subscribers

Audio Manager; any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/audio/audio_router_control` object are of the form:

```
msg::command_string\nid::ID\ndat:json:{JSON_data}
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID\ndat:json:{JSON_data}\n error::error_de
scription
```

Commands

Commands	Parameters	Data type	Description
`BT_A2DP_capability`	*supported*	Boolean	Indicates whether the paired device supports A2DP.
	connected	Boolean	Indicates whether the A2DP stream is connected.
`BT_SCO_capability`	*supported*	Boolean	Indicates whether the paired device supports SCO.
	connected	Boolean	Indicates whether the A2DP stream is connected.

Commands	Parameters	Data type	Description
	volumecontrol	String	Type of volume control supported by the paired device: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc.)
	audioprocessing	Boolean	Indicates whether the paired device supports audio processing.
`free_handle`	*audioman_handle*	Number	Handle returned by `get_handle`.
`get_alias_handle`	*target*	Number	The Audio Manager returns a unique handle given the target handle. Audio on the new handle will be impacted by audio ducking whenever the target handle is impacted.
	audioman_handle	Number	The returned Audio Manager handle that the client should use for all other actions.
`get_handle`	*type*	String	The audio source type: • `alert` • `default` • `inputfeedback` • `multimedia` • `pushtotalk` • `ringtone` • `soundeffect` • `texttospeech` • `videochat` • `voice` • `voicerecognition` • `voicerecording` • `voicetones`
	pid	Number	The process ID of the caller (returned by the *getpid()* call). Note that Audio Manager will get this automatically if not filled.

Commands	Parameters	Data type	Description
	suspended	Boolean	Indicates whether the audio handle is activated right away. Default is `true`.
	audioman_handle	Number	The returned Audio Manager handle that the client should use for all other actions.
`get_handle_concurrency_status`	*audioman_handle*	Number	Handle returned by `get_handle`.
	attenuated	Boolean	Indicates whether the given handle's audio type is being attenuated.
	muted	Boolean	Indicates whether the given handle's audio type is being muted.
	muted_by	String	Name of the audio type that mutes the given handle.
	muted_by_pid	String	The process ID of the audio source that mutes the given handle.
`get_type_concurrency_status`	*type*	String	Name of the audio type whose audio concurrency policy is being returned.
	attenuated	Boolean	Indicates whether the given handle's audio type is being attenuated.
	muted	Boolean	Indicates whether the given handle's audio type is being muted.
	muted_by	String	Name of the audio type that mutes the given handle.
	muted_by_pid	String	The process ID of the audio source that mutes the given handle.
`get_voice_enhanced_audio_option`	*source*	String	Name of the voice source: • `cellular` (default) • `voip`
	output	String	Name of the specific output device for this enhanced audio option. Default is `handset`.
	option	String	Name of the enhanced audio option: • `normal` • `boost_bass` • `boost_treble`

Commands	Parameters	Data type	Description
`get_voice_mode`	*source*	String	Name of the voice source: • `cellular` (default) • `voip`
	mode	String	The voice mode: • `ringer` • `on` • `off`
`numchans`	*number*	Number	Number of channels (`1` or `2`).
`pcm_input_closed` This command notifies apps that a PCM channel for input has been closed/suspended by the `libasound` library.	*audioman_handle*	Number	Handle returned by `get_handle`.
`pcm_input_opened` This command notifies apps that a PCM channel for input has been opened through the `libasound` library. The Audio Manager will default the type of this source to `generic`.	*audioman_handle*	Number	Handle returned by `get_handle`.
`pcm_output_closed`	*audioman_handle*	Number	Handle returned by `get_handle`.
`pcm_output_opened`	*audioman_handle*	Number	Handle returned by `get_handle`.
`print_audio_srcs`	n/a	n/a	This command causes the Audio Manager to log all the active audio sources.
`set_audio_src`	*audioman_handle*	Number	Handle returned by `get_handle`.
	type	String	The audio source type: • `alert` • `generic` • `multimedia`

Commands	Parameters	Data type	Description
			• soundeffect • ringtone • texttospeech • videochat • voice • voicerecognition • voicerecording
	input	String	The input device name overridden by the audio source. (See `/pps/services/audio/devices/` for the supported devices.) The `default` device clears the input setting.
	output	String	The output device name overridden by the audio source. A client normally shouldn't need to set the *input* and *output* parameters. Use these fields *only* to override the default routing path. For example, if the user has a headset in during a phone call, the default routing that the Audio Manager picks would be through the headset. However, the user could force the device to send output through the loud speaker and get input from the handset, in which case the phone app would have to tell `phone-pps` to override the output to speaker(3) and input to handset(1).
`set_voice_enhanced_audio_option`	*source*	String	Name of the voice source: • `cellular` (default) • `voip`

Commands	Parameters	Data type	Description
	output	String	Name of the specific output device for this enhanced audio option. Default is `handset`.
	option	String	Name of the enhanced audio option: • `normal` • `boost_bass` • `boost_treble`
`set_voice_mode`	*source*	String	Name of the voice source: • `cellular` (default) • `voip`
	mode	String	The voice mode: • `ringer` • `on` • `off`

/pps/services/audio/audio_router_status

The Audio Manager uses this object to reflect the status of voice routing

Publishers

Audio Manager

Subscribers

Any app

The `/pps/services/audio/audio_router_status` object contains telephony settings (`cellular` and `voip`) for voice enhancement for the supported devices. The object's format looks like this:

```
@audio_router_status
voiceservices.cellular.a2dp.audio_option::normal
voiceservices.cellular.btsco.audio_option::normal
voiceservices.cellular.hac.audio_option::normal
voiceservices.cellular.handset.audio_option::normal
voiceservices.cellular.hdmi.audio_option::normal
voiceservices.cellular.headphone.audio_option::normal
voiceservices.cellular.headset.audio_option::normal
voiceservices.cellular.lineout.audio_option::normal
voiceservices.cellular.speaker.audio_option::normal
voiceservices.cellular.status::off
voiceservices.cellular.tones.audio_option::normal
voiceservices.cellular.toslink.audio_option::normal
voiceservices.cellular.tty.audio_option::normal
voiceservices.cellular.usb.audio_option::normal
voiceservices.cellular.voice.audio_option::normal
voiceservices.voip.a2dp.audio_option::normal
voiceservices.voip.btsco.audio_option::normal
voiceservices.voip.hac.audio_option::normal
voiceservices.voip.handset.audio_option::normal
voiceservices.voip.hdmi.audio_option::normal
voiceservices.voip.headphone.audio_option::normal
voiceservices.voip.headset.audio_option::normal
voiceservices.voip.lineout.audio_option::normal
voiceservices.voip.speaker.audio_option::normal
voiceservices.voip.status::off
voiceservices.voip.tones.audio_option::normal
voiceservices.voip.toslink.audio_option::normal
voiceservices.voip.tty.audio_option::normal
voiceservices.voip.usb.audio_option::normal
voiceservices.voip.voice.audio_option::normal
```

The possible values for `audio_option` are:

- `normal`
- `boost_bass`

- `boost_treble`

The `status` field gives the current status of the voice call. The possible values are:

- `ringer`
- `on`
- `off`

For more information on the audio devices, see the `/pps/services/audio/devices/` entry.

/pps/services/audio/control

The Audio Manager listens for commands on this control object

Publishers

Any app

Subscribers

Audio Manager

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see “Server objects” in the *Persistent Publish/Subscribe Developer's Guide*.

Audio Manager library

Besides reading from and writing to the PPS audio objects directly, you may also choose to use the Audio Manager library, which allows your apps to set up and process events from audio devices. This library provides a C/C++ interface to audio devices that are accessible through the underlying PPS framework. Using this library, you can get and set properties for audio device status, volume, routing, and concurrency. You can also add and remove device events to notify clients that are using audio devices. For more information, see the *Audio Manager Library Reference*.

Message/response format

Commands sent to the `/pps/services/audio/control` object are of the form:

```
msg::command_string\nid::ID_number\ndat:json:{JSON_data}
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID_number\ndat:json:{JSON_data}\n er
ror::error_description
```

Commands

Command	Parameters	Data type	Description
`adjust_input_level`	*name*	String	The device name, which is listed under `/pps/services/audio/devices/` (e.g., `headset`).

Command	Parameters	Data type	Description
	level	Double	Percentage representing the desired adjustment of the input volume level.
`adjust_output_level`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
	level	Double	Percentage representing the adjustment of the output volume level.
`adjust_voice_output_level`	*name*	String	Name of the device to adjust the output volume level of when a voice call is activated.
	level	Double	Percentage representing the change of the output volume level.
`decrease_output_level`	*name*	String	The device name. The device's *volumecontrol* can be `percentage` or `simple`.
`get_a2dp_status`	*supported*	Boolean	Indicates whether the paired device supports A2DP.
	connected	Boolean	Indicates whether the A2DP stream is connected.
	numchans	Number	Number of channels that the A2DP stream supports (`1` or `2`).
	volumecontrol	String	Type of volume control that the current device supports for A2DP: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc.) Default is `unavailable`.
	state	String	State of the A2DP stream: • `closed` • `idle` • `open` • `streaming`
`get_device_property`	*name*	String	The device name.
	property	String	Name of the device's attribute.
	value	String	Value of the given attribute.

Command	Parameters	Data type	Description
`get_hdmi_info`	*numchans*	Number	Number of audio channels supported by the connected HDMI device.
	hdmiorder	String	Audio channel order of the connected HDMI device (e.g., `FL,FR`).
	audioconfig	String	Configuration of the audio output channels (e.g., "2.0", "5.1"). Note that this is used only by the `hdmi` device.
	mirror	Boolean	Indicates whether mirror mode is enabled.
	keep_alive	Boolean	Indicates whether the `hdmi` driver is to be kept alive when no audio stream is active.
	enabled	Boolean	Indicates whether the HDMI device is enabled.
	volumecontrol	String	Type of volume control supported by the attached HDMI device. Values: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc.)
`get_headphone_enable`	*name*	String	The device name.
`get_hfpg_sco_state`	*supported*	Boolean	Indicates whether the paired device supports HFP.
	connected	Boolean	Indicates whether the handsfree connection can be established.
	volumecontrol	String	Type of volume control that the current device supports for A2DP: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc.) Default is `unavailable`.
	audioprocessing	Boolean	Indicates whether the paired device needs audio processing by the handset or modem.
	state	String	State of the handsfree connection: • `closed`

Command	Parameters	Data type	Description
			• `idle` • `open` • `streaming`
	suspended	Boolean	Whether HFP is suspended as requested by the device. Default is `false`.
	remoteVAD	Boolean	Whether HFP also supports VAD and voice recording.
`get_input_level`	*name*	String	The device name. If an invalid name is given, the level of the currently selected input device is returned. If the given device is output only, the level of the corresponding input device when the output device is selected is returned.
`get_input_mute`	*name*	String	The device name.
`get_output_level`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
`get_output_mute`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
`get_voice_output_level`	*name*	String	The device to get the output volume level of when a voice call is activated.
	level	Double	Percentage representing the current output volume level.
`get_voice_output_mute`	*name*	String	The device name.
	muted	Boolean	Indicates whether the device output is muted.
`increase_output_level`	*name*	String	The device name. The device's *volumecontrol* can be `percentage` or `simple`.
`keep_output_alive`	*name*	String	The device name.
	keep_alive	Boolean	Indicates whether the given output device is to be kept alive when no audio stream is active.
`set_audio_mode` (may be deprecated)	*audiomode*	String	Tunings to apply to the hardware codec: • `audio` • `video`

Command	Parameters	Data type	Description
			• `voice` • `record`
`set_hac_enabled`	*enabled*	Boolean	Indicates whether HAC (Hearing Aid Compatibility) is enabled for `voice` handset mode.
`set_hdmi_enable`	*enabled*	Boolean	Indicates whether an HDMI device is connected. The HDMI device is activated only when these conditions are met: **1.** No higher-priority device is active. **2.** Mirror mode is set for the HDMI device. **3.** The *numchans*, *hdmiorder*, and *audioconfig* parameters for `set_hdmi_info` are set correctly.
`set_hdmi_info`	*numchans*	Number	Number of audio channels supported by the connected HDMI device.
	hdmiorder	String	Audio channel order of the connected HDMI device (e.g., `FL,FR`).
	audioconfig	String	Configuration of the audio output channels (e.g., "2.0", "5.1"). Note that this is used only by the `hdmi` device.
	volumecontrol	String	Type of volume control supported by the attached HDMI device: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc.) The `hdmi` audio driver sends HDMI info to the Audio Manager before the HDMI handler calls `set_hdmi_enable`. This is to prevent `set_hdmi_enable` from publishing inaccurate HDMI info.
`set_hdmi_mode`	*mirror*	Boolean	Indicates whether mirror mode is enabled.

Command	Parameters	Data type	Description
`set_headphone_enable` (may be deprecated)	*name*	String	Name of the device to be activated as the default because of this event.
	enabled	Boolean	Indicates whether the headphone is enabled as the output.
`set_headphone_override`	*hpoverride*	Boolean	Indicates whether headphone volume boost is enabled. If enabled, volume >92% is allowed.
`set_input_level`	*name*	String	The device name. If an invalid name is given, the level of the currently selected input device is returned. If the given device is output only, the level of the corresponding input device when the output device is selected is returned.
	level	Double	Percentage representing the desired input volume level.
`set_input_mute`	*name*	String	The device name.
	muted	Boolean	Indicates whether the given device input is muted.
`set_output_level`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
	level	Double	Percentage representing the desired output volume level.
`set_output_mute`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
	muted	Boolean	Indicates whether the given device output is muted.
`set_toslink_enable`	*enabled*	Boolean	Indicates whether a TOSLINK connection is used.
`set_tty_enabled`	*enabled*	Boolean	Indicates whether TTY mode is selected for `voice` headset mode. The Audio Manager automatically picks TTY mode only when a headset is connected and TTY is enabled during a voice call.
`set_voice_output_level`	*name*	String	The device to set the output volume level of when a voice call is activated.
	level	Double	Percentage representing the desired output volume level.

Command	Parameters	Data type	Description
`set_voice_output_mute`	*name*	String	The device to mute/unmute.
	muted	Boolean	Indicates whether the device output is muted.
`toggle_input_mute`	*name*	String	The device name. If an invalid name is given, the level of the currently selected input device is updated. If the given device is output only, the corresponding input device when the output device is selected is updated.
`toggle_output_mute`	*name*	String	The device name. The device's *volumecontrol* must be `percentage`.
`toggle_voice_output_mute`	*name*	String	The device to toggle the mute status of when a voice call is activated.

Examples

Set the volume level of the headset to 75%:

```
msg::set_output_level\nid::1\ndat:json:{"name":"headset", "lev
el":75}
```

The `"level"` field is a double, so its value doesn't have quotes.

Mute the speaker:

```
msg::set_output_mute\nid::2\ndat:json:{"name":"speaker", "mut
ed":"true"}
```

/pps/services/audio/devices/

This directory contains all the supported audio devices

Publishers

Audio Manager

Subscribers

Any app

Supported devices

Device name	Description
`a2dp`	Bluetooth A2DP connection. When selected, the default input (onboard mic) is selected.
`btsco`	Bluetooth SCO/HFP connection.
`hac`	HAC telecoil used for hearing aids. When selected, the default input (onboard mic) is selected.
`handset`	The handset on the device phone receiver. When selected, the default input (onboard mic) is selected.
`hdmi`	HDMI audio connection. When selected, the default input (onboard mic) is selected.
`headphone`	Headphone with no mic input. When selected, the default input (onboard mic) is selected.
`headset`	Headset with mic input.
`lineout`	An output device connected through the headset jack. When selected, the default input (onboard mic) is selected.
`speaker`	The main speaker on the device (handsfree on mobile phones).
`tones`	A virtual audio device for the system tones.
`toslink`	Optical audio device (“Toshiba Link”) used by some receivers.
`tty`	Telecommunications device for the deaf (connected through the headphone jack).
`usb`	Used for USB audio devices.
`voice`	A virtual audio device for voice content.
`wifidisplay`	Wireless connection to TVs for A/V playback.

To find all the connected audio devices on your system, read the `/pps/services/audio/devices/.all` object.

The default device

The status of the default audio device is published to the `/pps/services/audio/devices/default` object.

The default device has the following attributes:

Attribute	Description
device	Name of the default audio device used for playback (e.g., `speaker`).
input.device	The current default audio device for capture (e.g. `handset`).
input.path	The mountpoint to access the input device (e.g., `/dev/snd/pcmPreferredp`).
path	Path to the actual audio interface for input and output (e.g., `/dev/snd/pcmPreferredp`).

Device attributes

Each device has the following attributes:

Attribute	Data type	Description
audioconfig	String	Configuration of the audio output channels (e.g., "2.0", "5.1"). Note that this is used only by the `hdmi` device.
audioprocessing	Boolean	Indicates whether the device can do some audio processing that the system won't need to handle (e.g., a headset may be able to do noise cancellation).
connected	Boolean	Indicates whether a given device is connected.
controlpps	String	The path of the device's PPS control object. If this object exists, then the device is controlled by a PPS interface instead of an actual audio driver.
dependency	Boolean	Indicates whether this device depends on another device (this device has no effect unless the other is also connected).
hwinchans	Number	Total number of input channels on the hardware.

Attribute	Data type	Description
inchans	Number	Default number of channels that the client should use for multimedia audio capture. For example, for a device with four microphones, the client might use two for multimedia, in which case *inchans* would be `2` (*hwinchans* would be `4`). If *inchans* is `0`, then no input is supported for this device.
input.device	String	The current default audio device for capture (e.g. `handset`). This attribute appears only in the `default` object.
input.mixer	String	Name of the mixer group implemented by the input device for volume control. Values depend on the particular audio drivers and on the Audio Manager configuration. Default names are: • `BT A2DP In` • `BT SCO In` • `HDMI In` • `Input Gain` • `Tones In` • `TosLink In` • `USB In` • `Voice In` • `WIFI In` For details about audio drivers, see the `io-audio` manager and the `deva-*` entries in the QNX Neutrino *Utilities Reference*.
input.path	String	The mountpoint to access the input device (e.g., `/dev/snd/pcmPreferredc`).
keepalive	Boolean	Indicates whether the output device is to be kept alive when no audio stream is active.
mixer	String	Name of the mixer group implemented by the output device for volume control. As for *input.mixer*, values depend on the particular audio drivers and on the Audio Manager configuration. Default names are:

Attribute	Data type	Description
		• `BT A2DP Out` • `BT SCO Out` • `HDMI Out` • `Master` • `PCM Mixer` • `Tones Out` • `TosLink Out` • `USB Out` • `Voice Out`
mutable	Boolean	Indicates whether the device can be muted. During audio transitions from one device to another, the Audio Manager may mute both devices until the transition is complete so as to avoid sound leaks.
numchans	Number	Number of audio channels supported by the device.
order	String	The channel order (e.g., `FL,FR`) for two channels. Note that this is used only by the `hdmi` device.
path	String	Path to the actual audio interface for input and output (e.g., `/dev/snd/pcmPreferredp`).
public	Boolean	Indicates whether the device can be heard publicly (e.g., the value for a speaker would be `true`).
supported	Boolean	Indicates whether the device is physically installed on the target.
suspended	Boolean	Indicates whether the device is temporarily disabled by the system.
volumecontrol	String	Type of volume control supported: • `unavailable` (no volume control) • `simple` (supports only increase/decrease) • `percentage` (supports full control, including mute, specific steps, etc.)

/pps/services/audio/mixer

Status object for audio mixer levels

Publishers

Audio Control

Subscribers

Any app

Overview

The `/pps/services/audio/mixer` object contains the mixer settings (for balance, fade, etc.) used in the Audio Control app. The values shown indicate a range from 0 to 100%. Here's a sample object:

```
@mixer
balance:n:51.1811023622047
bass::74.02597402597402
fade::42.2077922077922
mid::92.20779220779221
treble::78.57142857142857
```

Volume levels are handled through the `/pps/services/audio/control` and `/pps/services/audio/status` objects.

Attributes

Attribute	Data type	Description
balance	Number	Distribution level of the left and right channels.
bass	String	Bass level.
fade	String	Distribution level of the front and rear channels.
mid	String	Middle level.
treble	String	Treble level.

/pps/services/audio/status

The Audio Manager uses this object to reflect the status of audio devices

Publishers

Audio Manager

Subscribers

Any app

Attributes

Attribute	Data type	Description
audio.mode	String	Current audio mode: • `audio` • `record` • `video` • `voice`
hpboostlevel	Number	Headphone boost level.
hpoutput.regulated	Boolean	This is `true` if the headphone output volume is limited by regulations (e.g., the setting may be 100% by an app, but the regulation limits the volume to 90%). This field will become `true` when `headphone` is the default routing path.
hpoutput.unregulatedlevel	Number	The unregulated volume setting (e.g., 100%), which may differ from the actual volume (e.g., 90%).
hpoverride	Boolean	Indicates whether audio boost is on.
hpoverride.supported	Boolean	Indicates whether audio-boost override is supported.
hpunsafelevel	Number	The unsafe volume level for headphones.
hpunsafezone	Boolean	The unsafe volume range for headphones.
hpunsafezone.supported	Boolean	Indicates whether the unsafe volume range is supported.

Attribute	Data type	Description
input.<device>.gain	Number	The hardware codec digital gain (in percent) for this device.
input.<device>.muted	Boolean	Indicates whether the input is muted for this device.
input.gain	Number	The input gain (in percent).
input.muted	Boolean	Indicates whether the input is muted.
output.available	Boolean	Indicates whether a device is selected as the default.
output.<device>.muted	Boolean	Indicates whether the output is muted for this device.
output.<device>.volume	String	The output gain (in percent) for this device.

/pps/services/audio/types/

This directory contains all the supported audio types

Publishers

Audio Manager

Subscribers

Any app

Overview

The Audio Manager publishes the status of each audio type for concurrent audio playback to the `/pps/services/audio/types/` directory. By monitoring these objects, an application may take certain actions when an event occurs. For example, a multimedia application may decide to pause when it is being muted by a higher-priority audio source.

Supported audio types

Audio type	Description
`alert`	Notifiers, such as calendar events, email, SMS, etc.
`default`	Any unclassified audio stream.
`inputfeedback`	Used for keyboard clicks.
`multimedia`	Used by media player applications.
`pushtotalk`	Used to denote streams related to push-to-talk use cases.
`ringtone`	Used for playback of ringtones when an incoming phone call occurs.
`soundeffect`	Sound effects that can never be attenuated, such as the camera click.
`texttospeech`	Text-to-speech services.
`videochat`	Used by the video chat client, this type isn't covered by the `voice` type because of a difference in automatic routing policy.
`voice`	Voiceband-related streams and certain telephony items (cellular or VoIP).

Audio type	Description
`voicerecognition`	Voice-recognition services such as VAD (Voice-Activated Dialing).
`voicerecording`	Used for voice-recording services.
`voicetones`	DTMF and call-progress tones, but can also be used to play back nontone-based audio during phone calls.

Audio type attributes

Each audio type object has the following attributes:

Attribute	Data type	Description
active_pid	String	The process ID of the application that is currently playing. This is used only in the `multimedia` type.
attenuated	Boolean	Indicates whether the audio type is being attenuated.
muted	Boolean	Indicates whether this audio type is being muted.
muted_by	String	The audio type (e.g., `ringtone`) that causes this audio type to be muted.
muted_by_pid	String	The process ID of the application that is muting this audio type.

/pps/services/audio/voice_status

The Audio Manager uses this object to reflect the status of voice settings

Publishers

Audio Manager

Subscribers

Any app

The `/pps/services/audio/voice_status` object contains voice settings for the audio devices. The object's format looks like this:

```
@voice_status
input.muted:b:false
voice.mode::Off
voice.output.a2dp.muted:b:false
voice.output.a2dp.volume:n:100.000000
voice.output.btsco.muted:b:false
voice.output.btsco.volume:n:100.000000
voice.output.handset.muted:b:false
voice.output.handset.volume:n:60.000000
voice.output.hdmi.muted:b:false
voice.output.hdmi.volume:n:100.000000
voice.output.headphone.muted:b:false
voice.output.headphone.volume:n:60.000000
voice.output.headset.muted:b:false
voice.output.headset.volume:n:60.000000
voice.output.lineout.muted:b:false
voice.output.lineout.volume:n:60.000000
voice.output.speaker.muted:b:false
voice.output.speaker.volume:n:60.000000
voice.output.tones.muted:b:false
voice.output.tones.volume:n:100.000000
voice.output.tty.muted:b:false
voice.output.tty.volume:n:100.000000
voice.output.usb.muted:b:false
voice.output.usb.volume:n:100.000000
voice.output.voice.muted:b:false
voice.output.voice.volume:n:100.000000
```

For more information on the audio devices, see the `/pps/services/audio/devices/` entry.

/pps/services/bluetooth/control

The Bluetooth Manager listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

Bluetooth Manager

Commands

The control object accepts the following commands:

Command	Parameters	Description
`authorize`	*data*: MAC address of the device. *data2*: Boolean value (`true` | `false`) to indicate whether to authorize.	Authorize an incoming Bluetooth connection request.
cancel_device_search	n/a	Cancel an initiated device search.
`cancel_pairing`	*data*: MAC address of the device.	Cancel an initiated device pairing request. Note that this command must be issued before the pairing completes.
`connect_service`	*data*: MAC address of the device. *data2*: Service number: • `0x110B`: A2DP/AVRCP (Advanced Audio Distribution Profile / Audio/Video Remote Control Profile) • `0x111E`: HANDSFREE (Hands-Free Profile) • `0x1134`: MAP_PROFILE (Message Access Profile) • `0x1115`: PAN (Personal Area Networking Profile) • `0x1130`: PBAP_PROFILE (Phone Book Access Profile) • `0x1101`: SERIAL_PORT (Serial Port Profile)	Connect to a Bluetooth service on the remote paired device. For the SPP service, you need to specify the UUID in the *data2* parameter by putting a colon between the service number and the UUID. For example: `0x1101:5DF26DC6-8E42-8401-6D98-75C100B108B1`

Command	Parameters	Description
`device_search`	n/a	Search for remote devices. Results will be published to `/pps/services/bluetooth/status` (devices will also be published under /pps/services/bluetooth/remote_devices/<mac_addr>).
`disconnect_service`	*data*: MAC address of the device. *data2*: Service number: • `0x110B`: A2DP/AVRCP (Advanced Audio Distribution Profile / Audio/Video Remote Control Profile) • `0x111E`: HANDSFREE (Hands-Free Profile) • `0x1134`: MAP_PROFILE (Message Access Profile) • `0x1115`: PAN (Personal Area Networking Profile) • `0x1130`: PBAP_PROFILE (Phone Book Access Profile) • `0x1101`: SERIAL_PORT (Serial Port Profile)	Disconnect a connected Bluetooth service. For SPP (as with the `connect_service` command), you must specify the UUID in the *data2* parameter.
`initiate_pairing`	*data*: MAC address of the device.	Initiate pairing to a remote device.
`radio_init`	n/a	Initialize the Bluetooth system and the radio chip.
`radio_shutdown`	n/a	Shut down the radio.
`remove_device`	*data*: MAC address of the device.	Delete a paired device from the system.
`set_access`	*data*: Accessibility level (0 - 4): • `0: IOBT_NOT_ACCESSIBLE` (no discoverability and connectability) • `1: IOBT_GENERAL_ACCESSIBLE` (general discoverability and connectability) • `2: IOBT_LIMITED_ACCESSIBLE` (limited discoverability and connectability)	Set the accessibility level of the Bluetooth system.

Command	Parameters	Description
	• `3: IOBT_CONNECTABLE_ONLY` (connectable but not discoverable) • `4: IOBT_DISCOVERABLE_ONLY` (discoverable but not connectable)	
`set_legacy_pin`	*data*: MAC address of the device. *data2*: PIN of legacy device (usually a four-digit number).	Set a numeric PIN required for authentication during pairing.
`set_name`	*data*: Desired name.	Set the friendly name of the Bluetooth system.
`set_passkey`	*data*: MAC address of the device. *data2*: Pass phrase string (usually four, but not more than six, characters).	Set the pass phrase required for authentication during pairing.
`user_confirm`	*data*: MAC address of the device. *data2*: Boolean value (`true` | `false`) to indicate whether to confirm request.	Confirm an authorization request from a remote device.

Examples

Set the accessibility level so devices will be discoverable:

```
echo "command::set_access\ndata:n:1" >> /pps/services/bluetooth/con
trol
```

Allow a connection request from the specified device:

```
echo "command::authorize\ndata::BB:C3:33:AD:66:CD\ndata2::b:true"
>> /pps/services/bluetooth/control
```

Connect the specified device using the Phone Book Access Profile:

```
echo "command::connect_service\ndata::BA:C3:32:AD:55:CC\nda
ta2::0x1130" >> /pps/services/bluetooth/control
```

Connect the specified device to the specified UUID using the Serial Port Profile:

```
echo "command::connect_service\ndata::BA:C3:32:AD:55:CC\nda
ta2::0x1101:5DF26DC6-8E42-8401-6D98-75C100B108B1" >> /pps/ser
vices/bluetooth/control
```

Pair with the specified device:

```
echo "command::initiate_pairing\ndata::CC:55:AD:24:46:51" >>
/pps/services/bluetooth/control
```

```
echo "command::user_confirm\ndata::CC:55:AD:24:46:51\ndata2:b:true"
>> /pps/services/bluetooth/control
```

Remove pairing for the specified device:

```
echo "command::remove_device\ndata::CC:55:AD:24:46:51" >>
/pps/services/bluetooth/control
```

/pps/services/bluetooth/handsfree/control

Control object for accepting phone call commands from the HMI

Publishers

Any app

Subscribers

Bluetooth Manager

Overview

The results of commands sent to this control object are published to `/pps/services/bluetooth/handsfree/status`.

Commands

The control object accepts the following commands:

Command	Parameters	Description
`HFP_ACCEPT`	`cmd_data` (phone number)	Accept incoming call.
`HFP_CALL`	`cmd_data` (phone number)	Call a number.
`HFP_HANGUP`	`cmd_data` (phone number)	End the call.

/pps/services/bluetooth/handsfree/status

Status object for reporting the results of call operations and the state of the device's phone line

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The Bluetooth Manager uses this status object to publish the results of commands sent to the `/pps/services/bluetooth/handsfree/control` object.

Attributes

Attribute	Values	Description
cmd_status	• `HFP_SUCCESS` • `HFP_FAIL`	Command success or failure.
state	`HFP_*`	See table below.
state_param	Various (e.g., phone number)	State-specific data.

Values for *state*

Value	Description
`HFP_CALL_ACTIVE`	An active call.
`HFP_CALL_ACTIVE_HELD`	An active call and also one or more held calls.
`HFP_CALL_ACTIVE_HELD_WAITING`	An active call, one or more held calls, as well as an incoming waiting call.
`HFP_CALL_ACTIVE_WAITING`	An active call and also an incoming waiting call.
`HFP_CALL_HELD_OUTGOING_ALERTING`	One or more held calls and also an outgoing call (alerting remote party).
`HFP_CALL_HELD_OUTGOING_DIALING`	One or more held calls and also an outgoing call (dialing remote party's number).
`HFP_CALL_HELD`	One or more held calls.
`HFP_CALL_HELD_WAITING`	One or more held calls and also an incoming waiting call.

Value	Description
`HFP_CALL_INCOMING`	An incoming call.
`HFP_CALL_OUTGOING_ALERTING`	An outgoing call (alerting remote party).
`HFP_CALL_OUTGOING_DIALING`	An outgoing call (dialing remote party's number).
`HFP_CALL_WAITING`	A waiting call (used only for call list state info).
`HFP_CONNECTED_IDLE`	No call activity; service is connected and idle.
`HFP_ERROR`	An error occurred.
`HFP_INITIALIZED`	Profile initialized.
`HFP_INITIALIZING`	Profile initializing.

/pps/services/bluetooth/messages/control

Control object for accepting message commands from the HMI

Publishers

Any app

Subscribers

Bluetooth Manager

Overview

The results of commands sent to this control object are published to `/pps/services/bluetooth/messages/status`.

Commands

The control object accepts the following commands:

Command	Parameters	Description
`DELETE`	• *account_id* (number from the MAP database) • *message_handle* (hex number ID) • *message_status*`:b:true|false`	Deletes or undeletes a message.
`GET_FOLDER_LISTING`	• *account_id* (number from the MAP database)	Refreshes the folder listing and stores it in the DB.
`GET_MSG`	• *account_id* (number from the MAP database) • *message_handle* (hex number ID)	Gets a specific, full message from the device and stores it in the DB.
`GET_MSG_LISTING`	• *account_id* (number from the MAP database) • *message_folder*	Gets message listings (first 256 characters of the subject, the sender, etc.) for a specific folder and stores them in the DB.
`INITIALIZATION_COMPLETE`	n/a	An *internal* command used to signal the completion of the initialization.
`SET_MSG_STATUS`	• *account_id* (number from the MAP database)	Sets a specific message as either read or unread.

Command	Parameters	Description
	• *message_handle* (hex number ID) • *message_status*:`b:true|false`	
`UPDATE_INBOX`	• *account_id* (number from the MAP database)	Forces device to connect to and sync with the network, if possible.

/pps/services/bluetooth/messages/notification

Object for notifications of Bluetooth messages

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The Bluetooth Manager uses this object to publish notifications of messages.

Attributes

Attribute	Values
account_id	Number starting from zero found in the MAP database.
message_handle	Hex number ID.
message_type	• `EMAIL` • `MMS` • `SMS_CDMA` • `SMS_GSM`
status	• `DELIVERY_FAILURE` • `DELIVERY_SUCCESS` • `MEMORY_AVAILABLE` • `MEMORY_FULL` • `MESSAGE_DELETED` • `MESSAGE_SHIFT` • `NEW_MESSAGES` • `SENDING_FAILURE` • `SENDING_SUCCESS`

/pps/services/bluetooth/messages/status

Status object for reporting the results of message commands

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The Bluetooth Manager uses this status object to publish the results of commands sent to the `/pps/services/bluetooth/messages/control` object.

Attributes

Attribute	Values
device	• MAC address of the connected device
state	• `CONNECTED` • `CONNECTING` (processing the initial sync) • `DISCONNECTED` • `UNINITIALIZED` (shown if an error occurred at startup)
status	• `COMPLETE` • `ERROR_BUSY` (occurs when the Bluetooth Manager is already busy processing another command) • `ERROR_COMMAND_NOT_KNOWN` • `ERROR_NOT_CONNECTED` (occurs when a command is received and there's no connection) • `FAILED` • `PROCESSING`

/pps/services/bluetooth/paired_devices/<mac_addr>

Directory that the Bluetooth Manager uses for publishing paired devices

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The `/pps/services/bluetooth/paired_devices` directory contains one object per Bluetooth device successfully paired when issuing an `initiate_pairing` command (via `/pps/services/bluetooth/control`). The object's name is the device's MAC address.

Attributes

Attribute	Data type	Description
available_services	String	Comma-separated list of profile services available (e.g., `0x1101` for SPP).
cod	String	Class of device code.
connected_services	JSON	Comma-separated list of profile services currently connected.
in_range	Boolean	*NOTE: This attribute is not actually used.*
name	String	Friendly name of the device.
paired	Boolean	Indicates whether device has been paired.
rssi	String	Received Signal Strength Indicator (in dBm).

/pps/services/bluetooth/phonebook/control

Control object for accepting phonebook commands from the HMI

Publishers

Any app

Subscribers

Bluetooth Manager

Overview

The results of commands sent to this control object are published to `/pps/services/bluetooth/phonebook/status`.

Commands

Command	Description
`INITIALIZATION_COMPLETE`	An *internal* command used to signal the completion of the initialization.
`SYNC_START`	Issue the phonebook sync command.

/pps/services/bluetooth/phonebook/status

Status object for reporting the phonebook sync progress

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The Bluetooth Manager uses this status object to publish the results of commands sent to the `/pps/services/bluetooth/phonebook/control` object.

Attributes

Attribute	Values
device	MAC address of the connected device
state	• `CONNECTED` • `CONNECTING` (processing the initial sync) • `DISCONNECTED` • `UNINITIALIZED` (shown if an error occurred at startup)
status	• `COMPLETE` • `ERROR_BUSY` (occurs when the Bluetooth Manager is already busy processing another command) • `ERROR_COMMAND_NOT_KNOWN` • `ERROR_NOT_CONNECTED` (occurs when a command is received and there's no connection) • `FAILED` • `PROCESSING`

/pps/services/bluetooth/remote_devices/<mac_addr>

Directory that the Bluetooth Manager uses for publishing discovered devices

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

The `/pps/services/bluetooth/remote_devices` directory contains one object per Bluetooth device discovered when issuing a `device_search` command (via `/pps/services/bluetooth/control`). The object's name is the device's MAC address.

Attributes

Attribute	Data type	Description
available_services	String	Comma-separated list of profile services available (e.g., `0x1101` for SPP).
cod	String	Class of device code.
connected_services	JSON	Comma-separated list of profile services currently connected.
in_range	Boolean	*NOTE: This attribute is not actually used.*
name	String	Friendly name of the device.
paired	Boolean	Indicates whether device has been paired.
rssi	String	Received Signal Strength Indicator (in dBm).

/pps/services/bluetooth/services

Shows the profile used for a connected device

Publishers

Bluetooth Manager

Subscribers

Any app

Overview

When a remote Bluetooth device is paired with the head unit, the Bluetooth Manager uses the `/pps/services/bluetooth/services` object to indicate which profile is being used. For each profile supported on the QNX CAR platform, the information appears in the object in this form:

```
service::mac_addr
```

where *service* is the abbreviated name of the profile and *mac_addr* is the remote device's MAC address.

Here's a sample object:

```
@services
[n]avrcp::98:D8:CB:5D:71:0C
[n]hfp::98:D8:CB:5D:71:0C
[n]map::98:D8:CB:5D:71:0C
[n]pan::
[n]pbap::98:D8:CB:5D:71:0C
[n]spp::
```

This object indicates that a remote device with a MAC address of `98:D8:CB:5D:71:0C` is connected via A2DP/AVRCP, HFP, MAP, and PBAP; the other profiles (PAN and SPP) aren't currently being used.

/pps/services/bluetooth/settings

Contains information about the Bluetooth stack

Publishers:

Bluetooth Manager

Subscribers:

Any app; Bluetooth Manager

Attributes

Attributes	Data type	Description
accessibility	Number	*NOTE: This attribute is not actually used.*
active_connections	Boolean	Indicates the presence of any active connections.
btaddr	String	MAC address of the local Bluetooth chip.
enabled	Boolean	Indicates whether the Bluetooth system is enabled.
name	String	Friendly name of the local Bluetooth chip.

/pps/services/bluetooth/spp/spp

Control object for the `pps-spp` service

Publishers

`pps-spp`; any app

Subscribers

`pps-spp`; any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see “Server objects” in the *Persistent Publish/Subscribe Developer's Guide*.

Overview

The `pps-spp` service supports any HTML5 application (e.g., PandoraLink) that needs to access Bluetooth SPP data. The server object will accept commands to open and close one SPP stream for a client. Once the SPP connection is open, the service will notify the client when new data has been read from SPP. This object also supports a command to write data to SPP.

Besides using this PPS server object, an HTML5 app would also need to use the `qnx.bluetooth` and `qnx.bluetooth.spp` JavaScript extensions to start or stop the SPP service on a paired device before communicating with the `pps-spp` service. For more information, see WebWorks JavaScript Extensions (CAR 2.0—deprecated) in *HTML5 and JavaScript Framework*.

Apps may also use the `/pps/services/bluetooth/control` object to issue Bluetooth commands for setting the accessibility level for devices, initiating pairing, and so on.

Message/response format

Commands sent to the `/pps/services/bluetooth/spp/spp` object are of the form:

msg::*command_string*\nid::*ID_number*\ndat::*command_parameters*

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

res::*command_string*\nid::*ID_number*\nerr::*errno_number*

The `id` field can be omitted if there's no need to get a response back for the message.

Messages sent by the client

Command	Parameters	Description
`open_stream`	*dat* JSON-encoded object that holds two parameters: • *mac*—MAC address of the device • *uuid*—UUID of the SPP server	Open an SPP stream.
`close_stream`	n/a	Close the SPP stream.
`write_data`	*base64Data*	Write Base64-encoded data to the stream.

Responses returned by the server

Besides returning the client's message and ID, the server can also send a `new_data` response:

Response	Parameters	Description
`new_data`	*base64Data*	Indicates new data was read from the SPP stream.

Examples

These examples show how to open an SPP stream, write “Hello World!”, then close the stream.

These examples assume that you've already opened an SPP connection to a device using the `connect_service` Bluetooth command. For example:

```
echo "command::connect_service\ndata::D5:DA:8E:43:ED:68\n
data2::0x1101:453994D5-D58B-96F9-6616-B37F586BA2EC" >>
/pps/services/bluetooth/control
```

For more information on Bluetooth commands, see the entry for `/pps/services/bluetooth/control` in this reference.

1. We need to force the shell to keep the file descriptor open (because this is a server object):

```
# exec 3<> /pps/services/bluetooth/spp/spp
```

2. Issue the command to open the stream, along with a `cat` so we can see the response:

   ```
   # echo
   "msg::open_stream\nid::1\ndat:json:{\"mac\":\"D5:DA:8E:43:ED:68\",
   \"uuid\":\"453994D5-D58B-96F9-6616-B37F586BA2EC\"}" >&3; cat
   <&3
   ```

   The `spp` object should show the response:

   ```
   @spp
   res::open_stream
   id::1
   ```

3. Next we write “Hello World!” to the SPP stream. Remember that the caller will need to Base64-encode the data before sending the message:

   ```
   # echo "msg::write_data\nid::2\ndat::SGVsbG8gV29ybGQh" >&3; cat
   <&3
   ```

   Again, the `spp` object will show the response:

   ```
   @spp
   res::write_data
   id::2
   ```

4. Finally, we send the `close_stream` message to close the stream:

   ```
   # echo "msg::close_stream\nid::3\ndat::" >&3; cat <&3
   ```

   And the object should look like this:

   ```
   @spp
   res::close_stream
   id::3
   ```

/pps/services/bluetooth/status

Status object that the Bluetooth Manager uses to publish responses to requests

Publishers

Bluetooth Manager

Subscribers

Any app

Events

The Bluetooth Manager publishes these events in response to requests made via the `/pps/services/bluetooth/control` object:

Event	Description
BTMGR_EVENT_AUTHORIZE_REQUIRED	Incoming Bluetooth connection request needs to be authorized.
BTMGR_EVENT_COMMAND_FAILED	The last command sent has failed.
BTMGR_EVENT_CONFIRM_NUMERIC_REQ	A pairing request from a remote device has been confirmed.
BTMGR_EVENT_CONNECT_ALL_FAILURE	The attempt to connect all services has failed.
BTMGR_EVENT_CONNECT_ALL_SUCCESS	The attempt to connect all services has succeeded.
BTMGR_EVENT_DEVICE_ADDED	A remote device was paired.
BTMGR_EVENT_DEVICE_DELETED	The specified paired device was successfully deleted.
BTMGR_EVENT_DEVICE_DELETED_FAILED	Request to delete the specified paired device has failed.
BTMGR_EVENT_DEVICE_LIST_CHANGED	List of devices under `/pps/services/bluetooth/remote_devices/<mac_addr>` has changed.
BTMGR_EVENT_DEVICE_SEARCH_COMPLETE	Search for remote devices is done.
BTMGR_EVENT_DISCONNECT_ALL_FAILURE	The attempt to disconnect all services has failed.
BTMGR_EVENT_DISCONNECT_ALL_SUCCESS	The attempt to disconnect all services has succeeded.
BTMGR_EVENT_DISPLAY_NUMERIC_IND	Authentication number shown during pairing.
BTMGR_EVENT_INIT_PAIRING_FAILED	Request to initiate pairing to the remote device has failed.
BTMGR_EVENT_INIT_PAIRING_SUCCESS	Request to initiate pairing to the remote device has succeeded.
BTMGR_EVENT_LEGACY_PIN_REQUIRED	Legacy PIN requested during pairing.
BTMGR_EVENT_PAIRING_CANCELED	Request to cancel an initiated device pairing has succeeded.

Event	Description
BTMGR_EVENT_PAIRING_COMPLETE	Request to initiate pairing to a remote device has succeeded.
BTMGR_EVENT_PAIRING_FAILED	Request to initiate pairing to a remote device has failed.
BTMGR_EVENT_PASSKEY_REQUIRED	Passkey (alphanumeric PIN) requested during pairing.
BTMGR_EVENT_RADIO_INIT	Bluetooth system and the radio chip were initialized.
BTMGR_EVENT_RADIO_SHUTDOWN	Radio was shut down.
BTMGR_EVENT_SERVICE_CONNECTED	A service was connected.
BTMGR_EVENT_SERVICE_DISCONNECTED	A service was disconnected.
BTMGR_EVENT_SET_ACCESS_FAILURE	Request to set the accessibility level has failed.
BTMGR_EVENT_SET_ACCESS_SUCCESS	Request to set the accessibility level has succeeded.
BTMGR_EVENT_STACK_FAULT	Bluetooth stack has failed; restart required.

/pps/services/bootmgr/

Directory for objects used by the Boot Manager and HMI

Publishers

Boot Manager; HMI

Subscribers

Boot Manager; HMI

Overview

The Boot Manager and the HMI use the `/pps/services/bootmgr/` directory as a means to inform each other of which apps to launch and in what order.

The `/pps/services/bootmgr/` directory contains the following objects:

`history`

The HMI uses the `last_tab` attribute in this object to publish the name of each app selected via the tabs on the screen. The `history` object persists across reboots, so by subscribing to this object, the Boot Manager can know the last app that was selected and will start that app first when the system boots.

`modules_ready/`

The Boot Manager creates an object in this directory for every app that's specified in the `/base/etc/slm-config-modules.xml` file. The HMI reads the `.all` object in the `modules_ready/` directory and launches the apps listed there. The objects themselves are empty and nonpersistent.

Sample `history` object

```
@history
last_tab::Home
```

Sample `.all` object

```
@Home
@launcher
@MediaPlayer
@navigation
@carcontrol
@Communication
@AppSection
@ASR
```

The Boot Manager collaborates with the *System Launch and Monitor (SLM)*, a special service that automates process management. For more information, see the entry for SLM in the *System Services Reference*.

/pps/services/clock/control

Control object for setting the date and time

Publishers

Any app

Subscribers

coreServices

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/clock/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}`

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}\nerr::`*`errno_number`*`\nerrstr::`*`error_description`*

The `set` command

This object accepts only one command:

msg::	id::	dat:json:
`set`	Number	`{"`*`parameter`*`":"`*`value`*`", "`*`parameter`*`":"`*`value`*`", ...}` (see below)

Parameters and values are as follows:

Parameter	Value
year	1970-9999
month	1-12

Parameter	Value
day	1-31 The `set` command doesn't completely validate input. For example, if you enter a month and day of `"2":"31"` (February 31), the `set` command will pass this input to the `date` command-line utility, which will reject it, but `set` won't report the error back to you. Remember to sanitize your input.
hour	0-24
minute	0-59
second	0-60

On real hardware (with a hardware clock on the I2C bus), the `set` command will set the time in the hardware clock. On VMware VMs, the command will set the Neutrino system time but not the hardware clock.

Response

res::	id::	dat:json:	err::	errstr::
`set`	Number (whatever was sent in `id::`)	n/a	*errno_number*	as appropriate

If your application is concerned with changes to the system clock or time zone, you can monitor the `/pps/services/clock/status` object.

/pps/services/clock/status

Get changes to system clock or time zone

Publishers

coreServices

Subscribers

Any app

Attributes

Attribute	Data type	Description
datechange	Number	The number represents the POSIX time value (in seconds since the epoch) after the clock was changed. This is updated whenever the system clock is changed.
timezonechange	String	This is updated whenever the *_CS_TIMEZONE* system variable is updated.

Note that the *datechange* attribute must be treated as an approximation. Although the attribute is the new POSIX time value (as returned by the *time()* function) after the clock has been changed, the value might have been read from the system *after* the clock was actually changed because of scheduling delays. Furthermore, a thread that subscribes to this status object might not receive notification of the change and get a chance to run until some time after *datechange* is modified. Notifications of changes can be relied upon, but only up to a point. The changed values should be considered only a hint.

/pps/services/gears/control

Control object for the OpenGL ES 2.0 Gears demo

Publishers

Any app

Subscribers

`gles2-gears`

Attributes

Attribute	Data type	Description
activated	Number	Turn the gears on (`1`) or off (`0`).
h	Number	Height (in pixels).
interval	Number	Swap interval. A value of `0` lets the app run as fast as possible; a value of `1` or more will limit the rate to the number of vsync periods.
pause	Number	Once the gears have been activated, pause (`1`) or resume (`0`) them.
screenGroup	String	When this attribute is present on startup (or when it's changed), the demo will try to join its screen window to this one as an embedded child.
w	Number	Width (in pixels).
x	Number	The *x* dimension from the top left (in pixels).
y	Number	The *y* dimension from the top left (in pixels).
z	Number	The *z* order of the window. This should be set to `-1` so that the parent window can take advantage of its own transparency to add a useful overlay.

The `/pps/services/gears/status` object shows the demo's frame rate.

Examples

Spin the gears as fast as possible:

```
echo "interval:n:0" >> /pps/services/gears/control
```

Now get the frame rate:

```
# cat /pps/services/gears/status
@status
framerate::133.462
```

/pps/services/gears/status

Status object for the OpenGL ES 2.0 Gears demo

Publishers

`gles2-gears`

Subscribers

Any app

Attributes

Attribute	Data type	Description
framerate	String	Animation rate (frames per second).

/pps/services/geolocation/control

The Geolocation service listens for commands on this object

Publishers

Geolocation service; any app

Subscribers

Any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/geolocation/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}`

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}\nerr::`*`error_description`*

Commands

The control object accepts the following commands:

Command	Description
`location`	Query the current location based on the IP address. The format for this message is as follows: `msg::location\nid::test_1\ndat:json:{"period": 2.0, "provider":"network", "fix_type":"wifi"}` where: • *period*—specifies the periodic delay (in seconds) between the update intervals. If period is `0` then the update is provided only once. • *provider*—geolocation source (always `network`). • *fix_type*—geolocation fix type (e.g., `wifi`, but the value isn't significant because this field is for browser compatibility only).

Command	Description
`cancel`	Cancel the currently running periodic request.

As soon as the Geolocation service receives the `location` message from the client, it queries *http://www.hostip.info* to get the current location based on the IP address. The correctness of the result depends on the contents of the database that the **hostip.info** site provides. The absence of an IP address for the requesting client in the database might yield an arbitrary result (e.g., “wrong location”). Note also that the Geolocation service is multithreaded, so it can handle requests from multiple clients at the same time.

Messages sent by the Geolocation service

Besides returning the client's message and ID, the Geolocation service can also send these responses:

Response	Description
`accuracy`	A percentage value representing the accuracy of the location (e.g., `60`).
`latitude`	The latitude (e.g., `45.3333`).
`longitude`	The longitude (e.g., `-75.9`).

Examples

1. If we want to observe responses from the Geolocation service, we need to force the shell to keep the file descriptor open (because this is a server object). We also use the `?wait` option to ensure we receive all responses:

   ```
   # exec 3<> /pps/services/geolocation/control?wait &&
   ```

2. Now we'll send the `location` request:

   ```
   echo 'msg:: location\nid::test_1\ndat:json: {"period": 5.0,
   "provider": "network", "fix_type": "wifi"}' >&3 && cat <&3
   ```

The control object might now look like this:

```
@control
res::location
id::test_1
dat:json: {"accuracy":60,"latitude": 45.3333,"longitude":-75.9}
```

/pps/services/geolocation/status

Status object for the Geolocation service

Publishers

Geolocation service; any app

Subscribers

Any app

Overview

The Geolocation service populates this object at startup to enable the browser to access current geolocation information. Here's a sample object:

```
@status
status:json:{"location_manager_location_on":true,
"location_manager_location_gnss_on":true}
```

Attributes

Attribute	Description
location_manager_location_on	Indicates whether the Geolocation service is on/off for the browser.
location_manager_location_gnss_on	Indicates whether the Global Navigation Satellite System is on/off.

/pps/services/hmi-notification/control

Control object for the **generic** event-source plugin for the HMI Notification Manager

Publishers

Any app

Subscribers

HMI Notification Manager

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Overview

The **generic** event-source plugin for the HMI Notification Manager provides a PPS interface that allows applications to use the HMI policy-management facilities without implementing a custom plugin. Applications can use this PPS interface to generate events. The **generic** plugin can also be used for automated testing. Using the PPS interface, a test application can generate customized events and inspect the outcome for expected results.

PPS messages sent to the plugin's interface from clients take the form of JSON objects that describe an event. The format is similar to that used by the `pps/services/hmi-notification/Status` and `pps/services/hmi-notification/Messaging` objects, but additional attributes are included in this control object.

The `event` message

This object accepts only one command:

msg::	dat:json:
`event`	`{"parameter":"value", "parameter":"value", ...}` (see below)

Parameters and values are as follows:

Parameter	Value	Description
`name`	*event_name*	Name of the event to be passed to the HMI Notification Manager.

Parameter	Value	Description
view	*view_application*	Name of the application responsible for handling the event.
type	*event_type*	Type of event being passed. One of: • display-start • display-end
priority	*priority_number*	An integer for the priority level (range is 0 to 7).
window-type	*window_type*	Type of window associated with the event. One of: • Fullscreen • Growl • Hidden • Notification • Overlay
fall back_types	*fallbacks*	List of window types to be used as fallbacks for the event. These events must be given in order of preference and are used as fallback types if the requested window type can't be accepted. These fallbacks will be tried in order until one that can be accepted is found or the event request is rejected.

Examples

A client sends a `display-start` event with a priority of `1`, with no fallback window types:

```
# echo 'event::{"name":"test_event", "view":"TestApp",
"type":"display-start", "priority":1, "window-type":"Overlay",
"fallback_types":[]}' >> /pps/services/hmi-notification/control
```

The `Status` object should then look like this:

```
# cat /pps/services/hmi-notification/Status
[n]@Status
display:json:[{"name":"test_event","type":"Overlay","view":"TestApp"}]
```

The client sends the associated `display-end` event:

```
# echo 'event::{"name":"test_event", "view":"TestApp",
"type":"display-end", "priority":1, "window-type":"Overlay",
"fallback_types":[]}' >> /pps/services/hmi-notification/control
```

The `Status` object should now be restored, showing the previously displayed event in its display list:

```
# cat /pps/services/hmi-notification/Status
[n]@Status
display:json:[{"name":"Home","type":"Fullscreen","view":"Home"}]
```

The following example illustrates a situation where the HMI Notification Manager uses a fallback window type.

1. The client must be attached to the `Messaging` object (because it's a server object):

   ```
   # exec 3<> /pps/services/hmi-notification/Messaging
   ```

2. Next, the client sends a `test-event1` event to the **generic** plugin to ensure that an event with priority 1 or greater is active:

   ```
   # echo
   'event::{"name":"test-event1","view":"TestApp1","type":"display-start",
   "priority":1,"window-type":"Overlay","fallback_types":[]}'
   >> /pps/services/hmi-notification/control
   ```

3. The client then sends `test-event2` with priority 1 and fallback window type `Growl`:

   ```
   # echo
   'event::{"name":"test-event2","view":"TestApp2","type":"display-start",
   "priority":1,"window-type":"Overlay","fallback_types":["Growl"]}'
   >> /pps/services/hmi-notification/control
   ```

4. Now we can see that the `Status` object has been updated:

   ```
   cat /pps/services/hmi-notification/Status ; cat <&3
   [n]@Status
   display:json:[{"name":"test_event1","type":"Overlay","view":"TestApp1"}]
   ```

   And the `Messaging` object will show that a `Growl` notification event has been sent:

   ```
   @Messaging
   display:json:[{"name":"test_event2","type":"Growl"}]
   ```

/pps/services/hmi-notification/Messaging

Server object for HMI Notification Manager

Publishers

HMI Notification Manager

Subscribers

Any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Overview

Since the `Messaging` object is used to issue transient notifications, this PPS object is created in server mode. When a client reads the object, no data is returned immediately. Clients should connect to the object with the `wait` flag enabled so as to be notified when the HMI Notification Manager has issued a transient notification. For example:

```
# cat /pps/services/hmi-notification/Messaging?wait
```

When a transient notification is issued, connected clients will be notified with a message similar to that used to express the manager's internal status. This message will be a JSON-formatted object that specifies the name of a growl event. The window type is specified in the message, but the value of this attribute will always be `"Growl"`. For example:

```
@Messaging
display:json:{"name":"Fluid::Alert","type":"Growl"}
```

/pps/services/hmi-notification/Status

Status object for HMI Notification Manager

Publishers

HMI Notification Manager

Subscribers

Any app

Overview

The `Status` object is used to expose the HMI Notification Manager's internal state. This object publishes the status of the various output modalities. Note that this may not represent the *actual* state of the output modalities, but rather the manager's belief of what they are.

For this release of the QNX CAR platform, the HMI Notification Manager supports only the `display` interaction modality.

An attribute is defined within the object to publish the internal state of the display. This attribute's value is a JSON list that specifies which applications should currently be displayed. The `display` attribute has the following format:

```
@Status
display:json:[{"name":"Home","type":"Modal"}]
```

In this example, the display currently has a single modal view called `Home`. The following example shows the contents of the `display` attribute in the case where a nontransient notification is to be displayed along with the `Home` application:

```
@Status
display:json:[{"name":"Home","type":"Modal"},
{"name":"Fluid::Alert","type":"Notification"}]
```

The HMI Notification Manager doesn't mandate how the HMI lays out the display in this case.

/pps/services/launcher/control

Control object for the applications launcher

Publishers

Applications Navigator

Subscribers

Any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/launcher/control` object should be in this form:

```
msg::command_string\ndat::{application_string}\nid::ID_number
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message:

```
res::command_string\ndat::application_string\nid::ID_number
```

Commands

msg::	dat::	id::
`start`	Application to launch (string from the app's directory name and key found under `/apps`).	Number (or any other identifier).
`stop`	Application to close. This must be the identifier returned in the `dat` attribute of the `start` response.	Number (or any other identifier).

Examples

The HTML5 version of the Navigator (also called Applications Window Manager) isn't intended to work with applications that are started or stopped from the command line. As a result, commands such as the following may not work as expected.

Launch the Communication application by writing the following to the `/pps/services/launcher/control` object:

```
msg::start\ndat::Communication.testDev_mmunicationf1e9ffb6\nid::1
```

The server responds with:

```
res::start\nid::1\ndat::2015282
```

You must pass `2015282` to the `stop` command to close this application.

Close the Communication application by writing the following to the `/pps/services/launcher/control` object:

```
msg::stop\ndat::2015282\nid::1
```

/pps/services/mirrorlink/applications

Holds information about the VNC-typed MirrorLink apps that are currently available

Publishers

`mlink-daemon`

Subscribers

Any app

Overview

When the `mlink-daemon` service detects a new MirrorLink device, it publishes the device's MirrorLink apps (up to a maximum of 10) to the `/pps/services/mirrorlink/applications` object.

To enable these MirrorLink apps to be launched from the HMI, the `mlink-daemon` service also creates shortcuts for the apps (in the `/apps/` directory) and then publishes the apps to the `/pps/system/navigator/applications/applications` object.

The `mlink-daemon` service also publishes to these PPS objects:

- `/pps/services/mirrorlink/entities` (for information about the devices currently available)
- `/pps/services/mirrorlink/rtp` (for audio-streaming information)

Object format

Each line in the object has three colon-separated fields:

`VNC.`*`app_number`*`:MLD.`*`entity`*`:`*`app_name`*

where:

app_number

The app's current number (`0` to `9`), prefixed with `VNC.`.

entity

Hexadecimal number identifying the device, prefixed with `MLD.`.

app_name

The app's name as reported by the device.

Here's a sample object:

```
[n]@applications
VNC.0:MLD.111eb0:Nokia Drive
VNC.1:MLD.111eb0:Nokia Call
VNC.2:MLD.111eb0:Nokia Music
VNC.3:MLD.111eb0:Car Mode
VNC.4:MLD.111eb0:VNC server
VNC.5:MLD.111eb0:Nokia Phone
```

/pps/services/mirrorlink/entities

Holds information about the MirrorLink entities (devices) that are currently available

Publishers

`mlink-daemon`

Subscribers

Any app

Object format

This object consists of four fields:

`MLD.`*`entity`*`:`*`type`*`:`*`name`*`;uuid:`*`identifier`*

where:

entity

Hexadecimal number identifying the device, prefixed with `MLD.`.

type

Type of entity as reported by the RealVNC SDK (usually `mirrorlink`).

name

Device's reported name.

The *name* field ends with a semicolon (`;`), not a colon.

identifier

Device's reported UUID.

Here's a sample object:

```
[n]@entities
MLD.8058ed8:mirrorlink:Symbian;uuid:59295e3b-5339-f5bf-cac0-3441af27fc53
```

/pps/services/mirrorlink/rtp

Holds RTP audio-streaming information for MirrorLink apps

Publishers

`mlink-daemon`

Subscribers

`mlink-rtp`

Object format

This object consists of five fields:

`RTP`*`ip_port`*`:`*`direction`*`:`*`interval/payload`*`,`*`latency`*`,`*`size`*

where:

ip_port

IP address and port number of the device for the stream, prefixed with `RTP`.

direction

Stream direction: `in` or `out`.

interval

If the direction is `in`, this is the interval (in milliseconds) for hello messages.

payload

If the direction is `out`, this is the RTP payload type.

latency

Initial playback latency (in milliseconds).

size

Maximum length (in bytes) of the payload.

Here's a sample object:

```
[n]@rtp
[n]RTP192.168.3.100_4000:out:1000,4800,9600
```

/pps/services/mm-control/control

Control object for `mm-control` mediaplayer

The `mm-control` mediaplayer is being deprecated. The `mm-player` mediaplayer replaces it.

Publishers

`mm-renderer`; any app

Subscribers

`mm-control`; any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see “Server objects” in the *Persistent Publish/Subscribe Developer's Guide*.

Overview

The `mm-control` manager conveniently ties together media outputs, zones, tracksessions, and renderer control. It provides a simple interface for HMIs that have limited access to native calls.

The following types of commands are supported:

- *Output commands* (p. 159)
- *Zone commands* (p. 159)
- *Tracksession commands* (p. 160)
- *Player commands* (p. 161)
- *iPod Out commands* (p. 166)

Message/response format

Commands sent to the `/pps/services/mm-control/control` object are of the form:

```
msg::command_string\nid::ID_number\ndat:json:{JSON_data}
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID_number\ndat:json:{JSON_data}\nerr::er
rno_number\nerrstr::error_description
```

Output commands

msg::	id::	dat:json:
`output_create`	Number	`{"name":"output_name", "url":"output url", "type":"output type"}`
`output_destroy`	Number	`{"name":"output_name"}`

Examples of output commands

```
echo 'msg::output_create\nid::1\ndat:json:{"name":"output0",
"url":"snd:pcmPreferredp","type":"audio"}' >> /pps/services/mm-
control/control
```

```
echo 'msg::output_destroy\nid::2\ndat:json:{"name":"output0"}' >>
/pps/services/mm-control/control
```

Output responses

res::	id::	dat:json:	err::	errstr::
`output_create`	Number	n/a	EBUSY	"Output *name* already exists"
`output_destroy`	Number	n/a	ENOENT	"Couldn't find output *name*"

Zone commands

Zones are containers for collecting outputs into sets that can be atomically added to or removed from players.

Outputs must already exist before you can use the `zone_attach_outputs` or `zone_detach_outputs` commands.

msg::	id::	dat:json:
`zone_attach_outputs`	Number	`{"name":"zone_name", "outputs":["output name",…]}`
`zone_create`	Number	`{"name":"zone_name"}`
`zone_destroy`	Number	`{"name":"zone_name"}`
`zone_detach_outputs`	Number	`{"name":"zone_name", "outputs":["output name",…]}`

Examples of zone commands

```
echo 'msg::zone_create\nid::3\ndat:json:{"name":"zone0"}' >>
/pps/services/mm-control/control
```

```
echo 'msg::zone_destroy\nid::4\ndat:json:{"name":"zone0"}' >>
/pps/services/mm-control/control
```

```
echo 'msg::zone_attach_outputs\nid::5\ndat:json:{"name":"zone0",
"outputs":["output0"]}' >> /pps/services/mm-control/control
```

```
echo 'msg::zone_detach_outputs\nid::6\ndat:json:{"name":"zone0",
"outputs":["output0"]}' >> /pps/services/mm-control/control
```

Zone responses

res::	id::	dat:json:	err::	errstr::
`zone_attach_outputs`	Number	n/a	*errno_number*	as appropriate
`zone_create`	Number	n/a	*errno_number*	as appropriate
`zone_destroy`	Number	n/a	*errno_number*	as appropriate
`zone_detach_outputs`	Number	n/a	*errno_number*	as appropriate

Tracksession commands

Use tracksessions to provide a collection of tracks to a player.

Note the following:

1. The `trksession_randomize_range` command lets you randomize any portion of a tracksession. A "start" of 0 and an "end" of -1 will randomize the entire session. You may want to randomize a subrange to allow the user to enable random playback partway through a session. Randomizing the range after the current track permits randomized playback of the remaining unheard songs without mixing in some of the tracks already heard.
2. The `trksession_get_range` command lets you view a session in either sequential or playback order, which will differ if randomized playback is requested.

msg::	id::	dat:json:
`trksession_create`	Number	`{"name":"trksession_name", "media_source":"attribute name in /pps/services/mm-detect/status"}`
`trksession_delete`	Number	`{"name":"trksession_name"}`
`trksession_get_range`	Number	`{"name":"trksession_name","start":start,"end":end, "type":"random|sequential"}`
`trksession_import`	Number	`{"name":"trksession_name","url":"sql"}`
`trksession_randomize_range`	Number	`{"name": "trksession_name","start": start,"end": end}`

Examples of trksession commands

```
echo 'msg::trksession_create\nid::7\ndat:json:{"name":"mtrkses
sion", "media_source":"dbmme"}' >> /pps/services/mm-control/control
```

```
echo 'msg::trksession_import\nid::8\ndat:json:{"name":"mtrkses
sion", "url":"SELECT mediastore_metadata.mountpath || fold
ers.basepath || library.filename AS url, fid AS userdata from me
diastore_metadata, folders, file WHERE file.folderid == fold
ers.folderid ORDER BY fid;"}' >> /pps/services/mm-control/control
```

```
echo 'msg::trksession_randomize_range\nid::10\ndat:json:{"name":
"mtrksession","start":0,"end":5}' >> /pps/services/mm-control/con
trol
```

```
echo 'msg::trksession_get_range\nid::11\ndat:json:{"name": "mtrk
session","start":0,"end":5,"type":"sequential"}' >> /pps/ser
vices/mm-control/control
```

```
echo 'msg::trksession_delete\nid::9\ndat:json:{"name":"mtrkses
sion"}' >> /pps/services/mm-control/control
```

Tracksession responses

res::	id::	dat:json:	err::	errstr::
`trksession_create`	Number	n/a	*errno_number*	as appropriate
`trksession_get_range`	Number	`{"num":num, "entries": [{"fid":fid,"url":"url"},...]}`	*errno_number*	as appropriate
`trksession_import`	Number	`{"trksession_size": int session_size}`	*errno_number*	as appropriate
`trksession_randomize_range`	Number	n/a	*errno_number*	as appropriate

Player commands

Players are used to play tracks from a tracksession. Players are created and referenced by name so that the HMI can connect to players created or started before the HMI is initialized.

For every player created, `mm-control` creates a PPS status object called `/pps/services/mm-control/`*`playername`*`/status` to publish the status associated with that player.

Note the following:

1. For `player_set_position`, the exact format of the position depends on the input media type.
2. For `player_set_read_mode`, if the requested read mode for an active tracksession differs from the current read mode, the tracksession will be either shuffled or unshuffled and the `TABLE trksessionview` field inside the media source's database will be updated accordingly.
3. For `player_set_trksession`, the track specified by `idx` in the tracksession named by `trksession` is attached as the new input.
4. For `player_set_current`, the track specified by `index` is attached as the new input.
5. For `player_play`, if you don't set `position`, the default playback position is the beginning of the track.

msg::	id::	dat:json:
`player_attach_zone`	Number	`{"player":"player_name","zone":"zone name"}`
`player_create`	Number	`{"name":"player_name"}`
`player_current_track`	Number	`{"player":"player_name"}`
`player_detach_zone`	Number	`{"player":"player_name","zone":"zone name"}`
`player_next_track`	Number	`{"player":"player_name"}`
`player_play`	Number	`{"player":"player_name","position":position_ms}`
`player_previous_track`	Number	`{"player":"player_name"}`
`player_set_current`	Number	`{"player":"player_name,"index":index}`
`player_set_params`	Number	`{"player":"player_name,"type":"context|input|track","params":[{"key0":"value0"},{"key1":"value1"},…]}`
`player_set_position`	Number	`{"player":"player_name,"position":"position_ms"}`
`player_set_read_mode`	Number	`{"player":"player_name,"mode":"random|sequential"}`
`player_set_repeat_mode`	Number	`{"player":"player_name, "mode":"all|one|none"}`
`player_set_speed`	Number	`{"player":"player_name","speed":speed}`
`player_stop`	Number	`{"player":"player_name"}`
`player_set_trksession`	Number	`{"player":"player_name","trksession":"trksession_name","idx":current_idx}`

Examples of player commands

```
echo 'msg::player_create\nid::9\ndat:json:{"name":"mptest"}' >>
/pps/services/mm-control/control
```

```
echo 'msg::player_attach_zone\nid::12\ndat:json:{"player":"mptest",
"zone":"zone0"}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_detach_zone\nid::12\ndat:json:{"player":"mptest",
"zone":"zone0"}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_set_trksession\nid::13\ndat:json:{"play
er":"mptest", "trksession":"mtrksession","idx":0}' >> /pps/ser
vices/mm-control/control
```

```
echo 'msg::player_set_speed\nid::14\ndat:json:{"player":"mptest",
"speed":0}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_set_position\nid::14\ndat:json:{"play
er":"mptest", "position":"100000"}' >> /pps/services/mm-control/con
trol
```

```
echo 'msg::player_set_read_mode\nid::17\ndat:json:{"play
er":"mptest", "mode":"random"}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_set_repeat_mode\nid::17\ndat:json:{"play
er":"mptest", "repeatmode":"all"}' >> /pps/services/mm-control/con
trol
```

```
echo 'msg::player_set_current\nid::14\ndat:json:{"player":"mptest",
"index":2}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_play\nid::13\ndat:json:{"player":"mptest"}' >>
/pps/services/mm-control/control
```

```
echo 'msg::player_stop\nid::17\ndat:json:{"player":"mptest"}' >>
/pps/services/mm-control/control
```

```
echo 'msg::player_previous_track\nid::17\ndat:json:{"play
er":"mptest"}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_next_track\nid::16\ndat:json:{"player":"mptest"}'
>> /pps/services/mm-control/control
```

```
echo 'msg::player_current_track\nid::15\ndat:json:{"play
er":"mptest"}' >> /pps/services/mm-control/control
```

```
echo 'msg::player_set_params\nid::14\ndat:json:{"player":"mptest",
"type":"track","params":[{"language":"english"}]}' >> /pps/ser
vices/mm-control/control
```

Player responses

res::	id::	dat:json:	err::	errstr::
`player_attach_zone`	Number	n/a	*errno_number*	as appropriate
`player_create`	Number	`{"status_path":"path to the status object for this player"}`	*errno_number*	as appropriate
`player_current_track`	Number	`{"trk_id":idx,"fid":fid,"url":"url"}`	*errno_number*	as appropriate
`player_detach_zone`	Number	n/a	*errno_number*	as appropriate
`player_next_track`	Number	`{"trk_id":idx,"fid":fid,"url":"url"}`	*errno_number*	as appropriate
`player_play`	Number	`{"trk_id":fid}`	*errno_number*	as appropriate
`player_previous_track`	Number	`{"trk_id":idx,"fid":fid,"url":"url"}`	*errno_number*	as appropriate
`player_set_current`	Number	n/a	*errno_number*	as appropriate
`player_set_params`	Number	n/a	*errno_number*	as appropriate
`player_set_position`	Number	n/a	*errno_number*	as appropriate
`player_set_read_mode`	Number	n/a	*errno_number*	as appropriate
`player_set_repeat_mode`	Number	n/a	*errno_number*	as appropriate
`player_set_speed`	Number	n/a	*errno_number*	as appropriate
`player_set_trksession`	Number	n/a	*errno_number*	as appropriate
`player_stop`	Number	n/a	*errno_number*	as appropriate

Player states

State:	Description:
IDLE	No active trksession is set for the player.
INVALID	An error has occurred; the player thread is dead. The player cannot accept any new commands from the HMI.
PLAYING	The active trksession is playing.
PAUSED	The active trksession is playing; the speed is set to zero.
STOPPED	All other scenarios.

Attributes published to `/pps/services/mm-control/`*`playername`*`/status`

When this command is called:	These attributes are published to the status object:
`player_create`	• `state::IDLE` • `speed::1000`
`player_play`	• `trkid:n:index_of_trk` • `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata) • `state::PLAYING` • `position:n:` (current position of the track) • `duration:n:` (for iPods only, duration of the track)
`player_set_current`	• `trkid:n:index_of_trk` • `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata)
`player_set_trksession`	• `trksession::trksession_name` • `media_source::attribute name in /pps/mm/status` • `trkid:n:index_of_trk` • `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata) • `state::STOPPED`
`player_stop`	If called when player is `PLAYING` or `PAUSED`: • `state::STOPPED`

iPod Out commands

Before you can use the following commands, you must first purchase the iPod addon for the QNX SDK for Apps and Media. For details, contact your *QNX representative*.

msg::	id::	dat:json:
`ipodout_enter`	Number	`{"name":"player_name","zone":"zone name", "mount point":"path to ipod"}`
`ipodout_exit`	Number	`{"name":"player_name"}`
`ipodout_pushuibutton`	Number	`{"name":"player_name", "direction":"left"|"right"|"up"|"down"|"select"|"menu", "delay":int32, "repeat":int32}` The `"delay"` and `"repeat"` attributes inside the JSON object are optional (default is 0).

Examples of iPod Out commands

```
echo 'msg::ipodout_enter\nid::15\ndat:json:{"name":"mpipodout",
"zone":"audio","mountpoint":"/fs/ipod0"}' >> /pps/services/mm-
control/control
```

```
echo 'msg::ipodout_exit\nid::15\ndat:json:{"name":"mpipodout"}'
>> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"left"}' >> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"right"}' >> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"up"}' >> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"down"}' >> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"select"}' >> /pps/services/mm-control/control
```

```
echo 'msg::ipodout_pushuibutton\nid::15\ndat:json:{"name":"mpipod
out", "direction":"menu"}' >> /pps/services/mm-control/control
```

iPod Out responses

res::	id::	dat:json:	err::	errstr::
`ipodout_enter`	Number	n/a	*errno_number*	as appropriate
`ipodout_exit`	Number	n/a	*errno_number*	as appropriate
`ipodout_pushuibutton`	Number	n/a	*errno_number*	as appropriate

/pps/services/mm-control/<playername>/status

Mediaplayer status object

Publishers

`mm-control`

Subscribers

Any app

Overview

Players are used to play back tracks from a tracksession. Players are created and referenced by name so that the HMI can connect to players created or started before the HMI is initialized.

For every player created, `mm-control` creates a PPS object called `/pps/services/mm-control/`*`playername`*`/status` to publish the status associated with that player.

Attributes published to `/pps/services/mm-control/`*`playername`*`/status`

When this command is called:	These attributes are published to the status object:
`player_create`	• `state::IDLE` • `speed::1000`
`player_set_trksession`	• `trksession::trksession_name` • `media_source::attribute name in /pps/mm/status` • `trkid:n:index_of_trk` • `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata) • `state::STOPPED`
`player_set_current`	• `trkid:n:index_of_trk` • `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata)
`player_play`	• `trkid:n:index_of_trk`

When this command is called:	These attributes are published to the status object:
	• `fid:n:` (*fid* of the track inside the database; stored inside the trksession as userdata) • `state::PLAYING` • `position:n:` (current position of the track) • `duration:n:` (for iPods only, duration of the track)
`player_stop`	If called when player is `PLAYING` or `PAUSED`: • `state::STOPPED`

/pps/services/mm-detect/status

The `mm-detect` service publishes device information to this object

Publishers

`mm-detect`

Subscribers

HMI Media Player; any app

Overview

The `mm-detect` service uses this status object to notify the HMI of mediastore insertions and removals as well as synchronization events. During synchronization, `mm-detect` publishes device information in JSON format to the status object.

To listen for mediastore insertions and removals, `mm-detect` subscribes to objects in the `/pps/qnx/mount/` directory.

Attributes

Attribute	Description
dbpath	Path to the database (e.g., `/dev/qdb/mme`).
device_type	Type of storage device: • `hdd` • `ipod` • `mtp` • `usb`
fs_type	Filesystem type: • `cd` • `dos (fat32)` • `ipod` • `iso9660` • `joliet` • `pfs` (e.g., for MTP devices) • `qnx` • `udf` • `unknown`

Attribute	Description
	The specific values for *fs_type* depend on the relevant filesystem driver. For details about each driver, see the fs-* entries in the OS *Utilities Reference*.
image_path	Path to the mediastore's images.
mount	Filesystem mountpoint (e.g., `/fs/usb0`).
name	Name of the removable device (e.g., KINGSTON).
synched	Indicates whether the file information has been synchronized.

/pps/services/multimedia/mediacontroller/control

The Now Playing service listens for commands on this control object

Publishers

Media controllers

Subscribers

Now Playing

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/multimedia/mediacontroller/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\nparameter_data`

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\nerror::`*`error_description`*

The `id` field can be omitted if there's no need to get a response back for the message.

Commands sent by the client

Command	Parameters	Data type	Description
`app_maximize`	n/a	n/a	Request that the active player be maximized.
`b_down`	`button_id:`*`bid_value`*	JSON	Notifies Now Playing of a button **make** event, intended only for the hardware button driver. The *bid_value* parameter can be one of: • `bid_forward` • `bid_hookswitch` • `bid_minus` • `bid_next`

Command	Parameters	Data type	Description
			• bid_pause • bid_play • bid_playpause • bid_plus • bid_prev • bid_rewind • bid_stop • bid_volume_down • bid_volume_up
b_up	button_id:*bid_value*	JSON	Notifies Now Playing of a button **break** event, intended only for the hardware button driver. For the *bid_value* parameter, see the description above for b_down.
v_set	*vlevel*	String	Set the volume level to the given *vlevel* (a number from 0 to 100).
v_up	n/a	n/a	Increase the volume level by one *step*. Media volume typically has 16 steps, voice has 10. Increasing the volume by one step has the effect of a 6.25% increase.
v_down	n/a	n/a	Decrease the volume level by one step.
v_mute	*mute*	Boolean	Set the *mute* state to true or false.
v_update	n/a	n/a	Indicates that a volume or mute update was done to the audioman service; the Now Playing service should reread it and display a toast notification.
m_play	n/a	n/a	Direct the connected media player to play its content.
m_pause	n/a	String	Direct the connected media player to pause its content.
m_playpause	n/a	String	Toggle the pause state and direct the connected media player to play or pause its content accordingly.
m_stop	n/a	String	Direct the connected media player to stop its content.
m_previous	n/a	String	Direct the connected media player to jump to the start of the current track or to the previous track.
m_next	n/a	String	Direct the connected media player to jump to the start of the next track.
m_fforward	n/a	String	Direct the connected media player to enter fast-forward mode.
m_rewind	n/a	String	Direct the connected media player to enter rewind mode.

Command	Parameters	Data type	Description
`md_subscribe`	n/a	n/a	Request that the caller subscribe to updates in metadata, track change, and state for the currently playing content.
`md_unsubscribe`	n/a	n/a	Request that the caller no longer subscribe to changes for the currently playing content.

Messages sent by Now Playing

Besides returning the client's message and ID, the Now Playing service can also send these responses:

Response	Parameters	Data type	Description
`m_active_pid`	*pid*	Number	Provides the active music player's process ID. A value of `0` indicates no active player.
`m_background_player_pid`	*pid*	Number (for the number of *pid*s); JSON (for each *pid*)	Provides the process IDs of the backgrounded (interrupted) players. A value of `0` indicates no backgrounded players. The format for this message is as follows: `msg::m_background_player_pid\nnumber:n:`*`number_of_pids`*`\ndat:json:`*`pid1, pid2, ...`*
`m_bg_pid`	*pid*	Number	Provides the backgrounded (interrupted) music player's process ID. A value of `0` indicates no backgrounded player.
`m_preactive_pid`	*pid*	Number	Provides the process ID of the *preactive* phone player. This is intended to be used by the phone when screening an incoming call. A value of `0` indicates no preactive player.
`m_state`	*state*	String	This message is sent only to subscribing controllers. Reflects the active player's state in the system. The *state* parameter can be one of: • `stopped` • `paused` • `playing` • `trackchange` Note that `trackchange` only delimits tracks—it doesn't indicate a state change (from stopped, paused, or playing). The player is expected to send `msg::state\ndat::trackchange` at every change of

Response	Parameters	Data type	Description
			track to generate the `trackchange` message to the controller.
`md_update`	*`md_attribute`*`:`*`value`*	JSON	This message is sent only to subscribing controllers. Reflects the initial (or change in) metadata for the currently played track to each client controller listening for metadata changes. Strictly speaking, the metadata attributes are arbitrary because these PPS interfaces will proxy any metadata information from the active media player in the system to all subscribed controllers. Known metadata attributes include: • `artist:`*`name`*—string representing the name of the artist being played • `album:`*`name`*—string representing the name of the album being played • `track:`*`name`*—string representing the name of the track being played • `position:`*`track_pos`*—number representing the track's current playback position (in milliseconds) • `duration:`*`track_duration`*—number representing the track's current duration (in milliseconds) • `albumArtwork:`*`file_path`*—absolute path to an image file representing the album's artwork • `nextEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **next** button on the volume toast (default is `true`) • `prevEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **prev** button on the volume toast (default is `true`) The `nextEnabled` and `prevEnabled` attributes are intended to be sent by media players that support and enable the fancy overlay in the volume toast.

Examples

1. If we want to observe responses from the Now Playing service, we need to force the shell to keep the file descriptor open (because this is a server object):

```
# exec 3<> /pps/services/multimedia/mediacontroller/control
```

2. Now let's set the volume to 50%:

```
# echo "msg::v_set\nid::1\ndat::50" >&3; cat <&3
```

3. We can see the response from the server:

```
@control
res::v_set
id::1
error::ok
```

/pps/services/multimedia/mediaplayer/control

The Now Playing service listens for commands on this control object

Publishers

Media players

Subscribers

Now Playing

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/multimedia/mediaplayer/control` object are of the form:

```
msg::command_string\nid::ID_number\nparameter_data
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID_number\nerror::error_description
```

Messages sent by the client

Command	Parameters	Data type	Description
`acquire`	n/a	n/a	The **`acquire`** command asks the system to establish the calling player as the *active media player* in the system. If a currently active media player is at the same or lower priority as the calling player, the active player will have its active state revoked. Media players must send the **`acquire`** command before they begin media playback so as to allow other players to stop playback. Players should do this only on an explicit action by the user (e.g., the user has pressed the play button or has just launched the player). If the active status is revoked by a higher-priority player (such as VAD or phone), the current player will be paused (not revoked) and will be resumed (if it wasn't paused) and given active status again when the higher-priority player releases. A player is released from being active

Command	Parameters	Data type	Description
			if it sends the release message, gets preempted by another acquiring player, or disconnects from the PPS object.
`button`	`key:`*`button_name`*	JSON	The *button_name* parameter is of the form: *`bn_button_button_length`*, where *button* is one of: • `forward` • `hookswitch` • `minus` • `next` • `pause` • `play` • `playpause` • `plus` • `prev` • `rewind` • `stop` • `vdown` • `vup` and *length* is one of: • `med` (600ms) • `short` (<600ms) A time value is required. Registering both `short` and `med` will request the button event as soon as the button goes down. Adding the threshold option (`nothresh`) to a short button registration will prevent the immediate action and provide `short` or `med` events based on how long the button is held. The *button_name* parameter is designed to be flexible enough to allow any hard button to be combined with a time value and optionally with another button (for simultaneous presses). If the `med` length is also registered, the action is taken as soon as the button goes down.

Command	Parameters	Data type	Description
	`nothresh:true|false` (optional)	JSON	Setting for threshold support.
	`action:forward`	JSON	Forward button notifications to the registered app.
`metadata`	`property:`*`value`*	JSON	Updates the metadata for the currently played track. Strictly speaking, the metadata attributes are arbitrary because these PPS interfaces will proxy any metadata information from the active media player in the system to all subscribed controllers. Known metadata attributes include: • `artist:`*`name`*—string representing the name of the artist being played • `album:`*`name`*—string representing the name of the album being played • `track:`*`name`*—string representing the name of the track being played • `position:`*`track_pos`*—number representing the track's current playback position (in milliseconds) • `duration:`*`track_duration`*—number representing the track's current duration (in milliseconds) • `albumArtwork:`*`file_path`*—absolute path to an image file representing the album's artwork • `nextEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **next** button on the volume toast (default is `true`) • `prevEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **prev** button on the volume toast (default is `true`) The `nextEnabled` and `prevEnabled` attributes are intended to be sent by media players that support and enable the `fancy` overlay in the volume toast.
`register`	`name:`*`name`*	JSON	The value for *name* is a descriptive string.
	`prio:`*`priority`*	JSON	One of `low` or `high`.
	`audio`	JSON	One of `voice` or `general`.
	`overlay` (optional)	JSON	One of `fancy` (i.e., enable controls) or `plain` (i.e., disable controls).
	`audioman_handle:`*`handle`* (optional)	JSON	Handle used for the audio channel. This handle is required for correct functionality of auto-resume when the media player is interrupted

Command	Parameters	Data type	Description
			by a higher-priority player. The correct functionality that requires this handle is detecting a *private-to-public switch*. For example, if the headset is unplugged during the interrupting player, the backgrounded player won't resume automatically. The audioman_handle:*handle* parameter is required to determine this switch. Generally, only the VAD, phone, or VoIP applications should use voice or any priority but low. Media players should register with low priority and general audio. When the voice-activated dialing or phone applications need to send an **acquire** command, Now Playing will pause the current media player and resume it (if it wasn't already paused) after the dialing operation or phone call.
	recorder (optional flag)	JSON	One of true or false. Setting the recorder flag to true indicates that this player is a *capture* device. This is currently intended to be used only for video recording that shouldn't be stopped by screening of incoming calls.
	pid:*apppid* (optional)	JSON	You may use *apppid* instead of the connection PID when maximizing the active player with msg::app_maximize (see the entry for /pps/services/multimedia/mediacontroller/control).
release	n/a	n/a	Notifies Now Playing that the media player is relinquishing control. If this is a higher-priority player (such as VAD or phone), any previously active media player that was paused when this one sent an **acquire** command will be resumed (if it wasn't paused) and will be given active status.
state	dat::*state*	String	Updates the player's play state to the given *state* string, one of: • stopped • paused • playing • trackchange—a special string for players that support multiple tracks. Players can send trackchange as a way to delimit tracks when playback moves from one track to another.
unbutton	key:*button_name*	JSON	Deregister a single button at the given length. See the button command for the values for *button_name*.

Messages sent by Now Playing

Besides returning the client's message and ID, the Now Playing service can also send the following:

Message	Parameters	Description
`revoke`	n/a	Indicates that the player's state as the active media player in the system is being revoked. The player should immediately stop playback and free up multimedia resources. This message will be sent to a player to deny its request to acquire active status if a higher-priority player is currently active. The message is also sent to the active player whenever another player preempts it.
`track\ndat::`*command_string*	One of: • `forward`—enter fast-forward mode • `holdData`—stop sending metadata updates • `next`—jump to the beginning of the next track • `pause`—pause playback of the current track • `play`—play the current track • `prev`—jump to the beginning of the current or previous track • `rewind`—enter rewind mode • `sendData`—start or resume sending metadata updates • `stop`—stop playback of the current track	Asks the media player to handle the given command string.
`key\ndat::`*button_name*	See the `button` command for the values for *button_name*.	Notifies the player of the given button press.

Command-line examples

1. If we want to observe responses from the Now Playing service, we need to force the shell to keep the file descriptor open (because this is a server object):

   ```
   # exec 3<> /pps/services/multimedia/mediaplayer/control
   ```

2. Notify Now Playing that we want to be the active media player:

   ```
   # echo "msg::acquire\nid::1" >&3; cat <&3
   ```

3. We can see the response from the server:

   ```
   @control
   res::acquire
   id::1
   error::ok
   ```

Button-registration examples

1. Register for both short and medium button lengths, which will cause Now Playing to send one button event at button down:

   ```
   msg::button\nid::1\ndat:json:{"key":"bn_vup_short","action":"for
   ward"} >> /pps/services/multimedia/mediaplayer/control
   ```

   ```
   msg::button\nid::1\ndat:json:{"key":"bn_vup_med","action":"for
   ward"} >> /pps/services/multimedia/mediaplayer/control
   ```

2. Now we'll set the `nothresh` option for the short button registration. Now Playing won't send the event at button down, but rather when the button goes up, as if you registered for just one length:

   ```
   msg::button\nid::1\ndat:json:{"key":"bn_vup_short","action":"for
   ward", "nothresh":true} >> /pps/services/multimedia/mediaplay
   er/control
   ```

   ```
   msg::button\nid::1\ndat:json:{"key":"bn_vup_med","action":"for
   ward"} >> /pps/services/multimedia/mediaplayer/control
   ```

   Note that if `nothresh` is set and the button is held for less than 600ms, the event will be sent at button up. If the button is held longer than 600ms, the event will be sent at 600ms.

/pps/services/multimedia/mediaplayer/phone

The Now Playing service listens for commands on this control object

Publishers

Phone app

Subscribers

Now Playing

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/multimedia/mediaplayer/phone` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\n`*`parameter_data`*

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\n`*`parameter_data`*`\n error::`*`error_description`*

Commands sent by the phone app

Command	Parameters	Data type	Description
`acquire`	n/a	n/a	The **`acquire`** command asks the system to establish the phone app as the *active media player* in the system. Any currently active player will be paused (not revoked) and then resumed (if it wasn't paused). When the phone releases, the paused player will be given active status again.
`button`	`key:`*`button_name`*	JSON	The *button_name* parameter is of the form: *`bn_button_button_length`*, where *button* is one of: • `forward` • `hookswitch` • `minus`

Command	Parameters	Data type	Description
			• next • pause • play • playpause • plus • prev • rewind • stop • vdown • vup and *length* is one of: • med (600ms) • short (<600ms) A time value is required. Registering both short and med will request the button event as soon as the button goes down. Adding the threshold option (nothresh) to a short button registration will prevent the immediate action and provide short or med events based on how long the button is held. The *button_name* parameter is designed to be flexible enough to allow any hard button to be combined with a time value and optionally with another button (for simultaneous presses). If the med length is also registered, the action is taken as soon as the button goes down.
	nothresh:true|false (optional)	JSON	Setting for threshold support.
	action:forward	JSON	Forward button notifications to the registered app.
metadata	property:*value*	JSON	Updates the metadata for the currently played track. Strictly speaking, the metadata attributes are arbitrary because these PPS interfaces will proxy any metadata information from the active media player in the system to all subscribed controllers. Known metadata attributes include:

Command	Parameters	Data type	Description
			• `artist:`*`name`*—string representing the name of the artist being played • `album:`*`name`*—string representing the name of the album being played • `track:`*`name`*—string representing the name of the track being played • `position:`*`track_pos`*—number representing the track's current playback position (in milliseconds) • `duration:`*`track_duration`*—number representing the track's current duration (in milliseconds) • `albumArtwork:`*`file_path`*—absolute path to an image file representing the album's artwork • `nextEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **next** button on the volume toast (default is `true`) • `prevEnabled:true|false`—control from the player to enable (`true`) or disable (`false`) the **prev** button on the volume toast (default is `true`) The `nextEnabled` and `prevEnabled` attributes are intended to be sent by media players that support and enable the `fancy` overlay in the volume toast.
`phonereg`	`name:`*`name`*	String	The value for *name* is a descriptive string. The priority is set at `phone` (which is higher than `low` or `high` for players). Generally, only the phone or VoIP applications should use this interface.
`preacquire`	n/a	n/a	Asks the system to establish the phone as the active media player. If a *recorder* (a type of player flagged at registration time) is active, the phone becomes preacquired alongside the recorder. The recorder continues with no state changes. This state is intended to be used by the phone for screening incoming calls. If the call is rejected, the phone should release and any recorder should continue. If the call is accepted, the phone should send the **`acquire`** message, which will cause the recorder to be backgrounded. Any current player (not flagged as a recorder) will be paused (not revoked) and will be resumed (if it wasn't paused) and given active status again

Command	Parameters	Data type	Description
			when the phone releases. A subsequent **`acquire`** before release won't affect the state.
`release`	n/a	n/a	Notifies Now Playing that the phone app is relinquishing control. Any previously active media player that was paused when the phone sent an `acquire` command will be resumed (if it wasn't paused) and will be given active status.
`unbutton`	`key:`*`button_name`*	JSON	Deregister a single button at the given length. See the `button` command for the values for *button_name*.

/pps/services/multimedia/mediaplayer/status

The Now Playing service publishes information about media players to this object

Publishers

Now Playing

Subscribers

Now Playing; any app

Attributes

Attribute	Data type	Description
`active_name`	String	Registered name of the active player (`none` if there's no registered name or player).
`active_pid`	Number	Process ID of the active player (`0` if there's no active player).
`active_prio`	String	Registered priority of the active player (`none` if there's no registered priority or player). Values are `low` or `high` for media players, but always `phone` if the phone app is active.
`background_player_name`	Number (for number of names); string (for names)	Provides the registered names of the backgrounded (previously active) players. The format of this message is as follows: `background_player_name:`*`number_of_names`*`:`*`name1,name2,...`* The value for *number_of_names* is `0` if there are no backgrounded players.
`background_player_pid`	Number (for number of PIDs); number (for PID)	Provides the PIDs of the backgrounded (previously active) players. The format of this message is as follows: `background_player_pid:`*`number_of_pids`*`:`*`pid1,pid2,...`* The value for *number_of_pids* is `0` if there are no backgrounded players.
`background_player_prio`	Number (for number of players); string (for priority)	Provides the registered priorities of the backgrounded (previously active) players. The format of this message is as follows: `background_player_prio:`*`number_of_players`*`:`*`prio1,prio2,...`* The value for *number_of_players* is `0` if there are no backgrounded players.

Attribute	Data type	Description
bg_name (Deprecated—use background_player_name instead.)	String	Registered name of the previously active (backgrounded) player (none if there's no registered name or backgrounded player).
bg_pid	Number	Process ID of the previously active (backgrounded) player (0 if there's no backgrounded player).
bg_prio (Deprecated—use background_player_prio instead.)	String	Registered priority of the previously active (backgrounded) player (none if there's no registered priority or backgrounded player).
preactive__name	String	Registered name of the preactive player (none if there's no preactive player).
preactive__pid	Number	Process ID of the preactive player (0 if there's no preactive player).
preactive__prio	String	Registered priority of the preactive player (none if there's no registered priority or player).

Sample status objects

No active player:

```
@status
active_name::none
active_pid:n:0
active_prio::none
background_player_name:0:
background_player_pid:0:
background_player_prio:0:
bg_name::none
bg_pid:n:0
bg_prio::none
preactive__name::none
preactive__pid:n:0
preactive__prio::none
```

One active player:

```
@status
active_name::NPC:lnu4ank1hq8
active_pid:n:18440228
active_prio::low
background_player_name:0:
background_player_pid:0:
background_player_prio:0:
bg_name::none
```

```
bg_pid:n:0
bg_prio::none
preactive__name::none
preactive__pid:n:0
preactive__prio::none
```

Phone has interrupted music:

```
@status
active_name::phone
active_pid:n:6267049
active_prio::phone
background_player_name:1:NPC:lnu4ank1hq8
background_player_pid:1:18440228
background_player_prio:1:low
bg_name::NPC:lnu4ank1hq8
bg_pid:n:18440228
bg_prio::low
preactive__name::none
preactive__pid:n:0
preactive__prio::none
```

Phone has interrupted video recording:

```
@status
active_name::camera
active_pid:n:18624535
active_prio::low
background_player_name:0:
background_player_pid:0:
background_player_prio:0:
bg_name::none
bg_pid:n:0
bg_prio::none
preactive__name::phone
preactive__pid:n:6267049
preactive__prio::phone
```

Five high-priority players are stacked, acquired in this order: hiprioplayer, plr01, plr02, plr03, plr04:

```
@status
active_name::plr04
active_pid:n:18710747
active_prio::high
background_player_name:4:plr03,plr02,plr01,hiprioplayer
background_player_pid:4:18710732,18710728,18710564,18706455
background_player_prio:4:high,high,high,high
bg_name::plr03
bg_pid:n:18710732
bg_prio::high
preactive__name::none
preactive__pid:n:0
preactive__prio::none
```

/pps/services/multimedia/renderer/component/

Directory for `.all` object and for dynamically loaded plugins

Publishers

`mm-renderer`

Subscribers

Any app

The `.all` object

This `.all` object contains supported file and MIME types.

Attributes

Attribute	Description
audioencodeextensions	Comma-separated list of supported extensions for file outputs (e.g., `m4a,wav`).
mime	Comma-separated list of allowed combinations of playable MIME types (e.g., `3gpp,video`).

Other objects in the `component` directory

The `component` directory can also contain information about `mm-renderer`'s dynamically loaded plugins (`mm-renderer` has defined its own plugin interface for modularization and extensibility):

- `mmr-track-engine` (engine plugin for playing tracks)
- `mmr-playlist-engine` (engine plugin for playing playlists)
- `mmr-mmf-routing` (routing plugin for handling the actual play tasks)
- `mmr-mmfrip-routing` (routing plugin for ripping)

/pps/services/multimedia/renderer/context/<contextname>

Directory that the multimedia renderer uses for publishing context objects

Publishers

`mm-renderer`

Subscribers

Any app

Overview

Whenever a client calls *mmr_context_create()*, the `mm-renderer` manager creates a directory under `/pps/services/multimedia/renderer/context`, using the name given in the *mmr_context_create()* call. This *contextname* directory can contain several PPS objects:

- `/pps...<contextname>/param`
- `/pps...<contextname>/output#`
- `/pps...<contextname>/input`
- `/pps...<contextname>/metadata` (created when an input is attached to the context)
- `/pps...<contextname>/p#` (if input is a playlist, a `p#` object is created for each playlist entry)
- `/pps...<contextname>/play-queue` (created if input is a playlist)
- `/pps/services/multimedia/renderer/context/<contextname>/q#`
- `/pps...<contextname>/state`
- `/pps...<contextname>/status`

/pps/services/multimedia/renderer/context/<contextname>/input

Holds input parameters for the specified context

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Attributes

Attribute	Description
url	URL of the attached input.
type	Values: • `track` (a sinlge track) • `playlist` (a track sequence) • `autolist` (a single track formatted as a playlist)
repeat	Only for `playlist` and `autolist`, continuously replay the input. Values: • `"none"` (default) • `"track"` • `"all"`

/pps/services/multimedia/renderer/context/<contextname>/metadata

Metadata object for inputs attached to a multimedia renderer context

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Sample object

```
[n]@metadata
md_title_album::Ballads In White Forest   (2008)
md_title_artist::ALONE IN THE CHAOS
md_title_bitrate::188000
md_title_comment::http://www.jamendo.com/
md_title_duration::254066
md_title_mediatype::4
md_title_name::0000025
md_title_samplerate::44100
md_title_seekable::1
md_title_track::1
url::/accounts/1000/shared/music/set006/01 - 0000025.mp3
```

/pps/services/multimedia/renderer/context/<contextname>/output#

Holds output parameters for the specified context

The *contextname* is the name given in *mmr_context_create()*. The *#* is the output ID returned by *mmr_output_attach()*.

Publishers

`mm-renderer`

Subscribers

Any app

Attributes

Attribute	Description
type	Values: • `audio` (`volume` in the range of 0 to 100) and `audio_type` as specified in *audio_manager_get_name_from_type()* • `video` • `av` • `file` Output parameters may vary, depending on how your system is implemented. See *mmr_output_parameters()* for more information and examples.
url	URL of the attached output.

/pps/services/multimedia/renderer/context/<contextname>/p#

Object for input URL and parameters for tracks

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Overview

When the input is a playlist, the p# object is created to hold the URL and parameters for one track in the playlist. The # is the position of the track in the playlist (starting from 1).

/pps/services/multimedia/renderer/context/<contextname>/param

Contains the parameters set via *mmr_context_parameters()*

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Parameters

Parameter	Description
updateinterval	Allows an application to request a particular frequency in status updates (default is 1000 ms).
Parameters that map to `libcurl` library options: • `OPT_VERBOSE` • `OPT_CONNECTTIMEOUT_MS` • `OPT_LOW_SPEED_LIMIT` • `OPT_LOW_SPEED_TIME` • `OPT_USERAGENT` • `OPT_USERNAME` • `OPT_PASSWORD` • `OPT_PROXYUSERNAME` • `OPT_PROXYPASSWORD` • `OPT_COOKIE` • `OPT_COOKIEFILE` • `OPT_COOKIEJAR` • `OPT_COOKIESESSION` • `OPT_CAINFO` • `OPT_CAPATH` • `OPT_SSL_VERIFYPEER`	

Parameter	Description
• `OPT_SSL_VERIFYHOST` • `OPT_PROXY` • `OPT_NOPROXY` • `OPT_HTTPPROXYTUNNEL` • `OPT_PROXYPORT` • `OPT_PROXYTYPE` • `OPT_PROXYAUTH` • `OPT_HTTPAUTH` • `OPT_HTTPHEADER` • `OPT_DNSCACHETIMEOUT`	
Parameters that map to socket options: • `OPT_SO_RCVBUF` • `OPT_SO_SNDBUF`	See *getsockopt()* in the *Neutrino Library Reference*.

/pps/services/multimedia/renderer/context/<contextname>/play-queue

Object for the size of the playlist window

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Overview

When the input is a playlist, `mm-renderer` creates a *playlist window* for the currently playing item and the items in front of and behind it, using the following PPS objects in the *contextname* directory:

- *p#*—contains the parameters for one track in the playlist
- *play-queue*—represents the size of the playlist window
- *q#*—contains the metadata for one track in the playlist

Attribute	Description
end	Index of the last p# item in the window.
start	Index of the first p# item in the window.
total	Total number of items in the playlist. This is set when a track is first played.

/pps/services/multimedia/renderer/context/<contextname>/q#

Object for metadata for tracks

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Overview

When the input is a playlist, the `q#` object is created to hold the metadata for one track in the playlist. The *#* is the position of the track in the playlist (starting from 1).

/pps/services/multimedia/renderer/context/<contextname>/state

Holds the play state for the specified context

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

We recommend that you read this object in *delta mode* to ensure that you get all the errors and warnings that may occur.

For more on delta mode, see “Subscribing” in the *Persistent Publish/Subscribe Developer's Guide*.

Attributes

Attribute	Description
error	Most recent error code (deleted when playback is restarted).
error_pos	Play position when the error occurred.
input	Input URL (deleted when input is detached).
speed	Current set speed in units of 1/1000 of normal speed
state	Can be one of these values: • `idle` • `playing` • `stopped`
warning	Most recent warning (deleted when playback is stopped).
warning_pos	Play position when the warning occurred.

How *state*, errors, and warnings are set

Condition:	The *state* attribute is set to:	Other attributes set:
No input is attached	`idle`	none
An input is attached	`stopped` (from `idle`)	*input* is set to input's URL
Playback begins	`playing` (from `stopped`)	*error* and *error_pos* are deleted
End of media is reached	`stopped` (from `playing`)	*error* is set to `MMR_ERROR_NONE` (note that no error code is set if playback is stopped via a function call)
Warning occurs (note that warnings don't stop playback)	`playing`	*warning* and *warning_pos* are set
Error occurs (note that errors will stop playback)	`stopped`	*warning* and *warning_pos* are deleted; *error* and *error_pos* are set

For error codes, see `mm_error_code_t` in the *Multimedia Renderer Developer's Guide*.

/pps/services/multimedia/renderer/context/<contextname>/status

Status object for the multimedia renderer context

The *contextname* is the name given in *mmr_context_create()*.

Publishers

`mm-renderer`

Subscribers

Any app

Don't read this potentially high-bandwidth object in *delta mode*.

For more on delta mode, see "Subscribing" in the *Persistent Publish/Subscribe Developer's Guide*.

Attributes

Attribute	Description
bufferlevel	Two decimal numbers (in milliseconds): *level/capacity*
position	Play position compatible with *mmr_seek()* (value is *milliseconds* for single tracks; *tracknumber:milliseconds* for playlists)

/pps/services/multimedia/renderer/control

The `mm-renderer` service listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

`mm-renderer`

Commands

The commands correspond to functions defined in `renderer.h`. For example, the `contextOpen` command maps to *mmr_context_open()*. For more information, see "Multimedia Renderer Client API" in the *Multimedia Renderer Developer's Guide*.

Command	Parameters
`commandSend`	• *ctxt* (the context handle) • *cmd* (command string)
`contextClose`	*ctxt* (the context handle)
`contextCreate`	• *name* (the context name) • *flags* (must be `0`) • *mode* (file permissions for the context directory)
`contextDestroy`	*ctxt* (the context handle)
`contextOpen`	*name* (the context name)
`contextParameters`	• *ctxt* (the context handle) • *parms* (the dictionary object containing the parameters to set)
`inputAttach`	• *ctxt* (the context handle) • *url* (the URL of the new input) • *type* (`"track"`, `"playlist"`, or `"autolist"`)
`inputDetach`	*ctxt* (the context handle)

Command	Parameters
`inputParameters`	• *ctxt* (the context handle) • *parms* (the dictionary object containing the parameters to set)
`listChange`	• *ctxt* (the context handle) • *url* (the URL of the new playlist) • *delta* (difference between position of the current track on the old and new lists)
`outputAttach`	• *ctxt* (the context handle) • *url* (the URL of the new input) • *type* (`"audio"`, `"video"`, or `"file"`)
`outputDetach`	• *ctxt* (the context handle) • *output_id* (the output ID number)
`outputParameters`	• *ctxt* (the context handle) • *output_id* (the output ID number) • *parms* (the dictionary object containing the parameters to set)
`play`	*ctxt* (the context handle)
`seek`	• *ctxt* (the context handle) • *position* (the position to seek to)
`speedSet`	• *ctxt* (the context handle) • *speed* (the new play speed)
`stop`	*ctxt* (the context handle)
`trackParameters`	• *ctxt* (the context handle) • *index* (an index within the current playlist windoow; `0` for default) • *params* (the track parameters for the playlist; NULL to reset to default)

Examples

Attach an input to the specified context and play the track `/macqnx/RCPS_SuckerPunchTH_M2_TestFile.mpg`:

```
echo 'msg::inputAttach\ndat:json:{"ctxt":0, "url":"/mac
qnx/RCPS_SuckerPunchTH_M2_TestFile.mpg", "type":"track"}' >>
/pps/services/multimedia/renderer/control
```

/pps/services/networking/all/interfaces/<interface>

Status object for network interfaces

Publishers

Network Manager

Subscribers

Any app

Overview

For every connected network interface, the Network Manager creates an object under the `/pps/services/networking/all/interfaces/` directory for publishing status information. The name of each status object is the interface name as reported by the **`ifconfig`** utility (e.g., `en0`).

Attributes

Attribute	Data type	Description
connected	Boolean	Indicates whether (`true|false`) the given interface is connected.
fib	Number	FIB number.
httpproxy	String	IP address of the HTTP proxy server.
httpproxyloginrequired	Boolean	Indicates whether (`true|false`) the HTTP proxy requires login credentials.
ip4_ok	String	A `yes` here indicates whether IPv4 connectivity is available. Possible error strings: • `error_no_ip_addr` • `error_no_ip_gateway` • `error_no_nameserver` • `error_not_configured` • `error_not_connected` • `error_not_up`
ip6_ok	String	Same as for *ip4_ok*, but for IPv6 connectivity.
ip_addresses	JSON	Array of IP addresses assigned to this interface.
ip_bcastaddr	String	Interface's IP broadcast address (if it has one).
ip_dstaddr	String	Interface's IP destination address (if it has one).

Attribute	Data type	Description
ip_gateway	JSON	Array of IP gateways.
ip_ok	String	General status attribute for IP connectivity.
link_address	String	Link layer (MAC) address.
manual	String	Indicates whether (`yes|no`) manual or DHCP settings will be used. If *manual* is `yes`, these settings apply: • `ip_address=` • `gateway=` • `netmask=` • `nameservers=` • `searchdomains=` If *manual* is `no`, these settings apply: • `dhcp=on|off|auto` • `dhcp6=on|off|auto`
manual6	String	Indicates whether (`on|off`) IPv6 manual or DHCP settings will be used.
mtu	Number	MTU number for this interface.
nameservers	JSON	Array of nameserver addresses.
searchdomains	String	Array of strings to be used for DNS resolution.
type	String	Type of interface. Possible values: • `bb` (any BlackBerry Bridge BIS-B/BES-B or BBIO HTTP proxy connection) • `bluetooth_dun` (any Bluetooth tethering interface) • `cellular` (any cellular network interface) • `usb` (any direct USB cable to a PC or Mac) • `vpn` (any VPN tunnel) • `wifi` (any wireless network interface) • `wired` (any wired Ethernet interface)
up	Boolean	Indicates whether (`true|false`) the physical interface is up.

/pps/services/networking/all/proxy

Status object for proxy information

Publishers

Network Manager

Subscribers

Any app

All of this information (except for *httpproxylogin*) is also published to the `/pps/services/networking/all/status_public` object.

Attributes

Attribute	Data type	Description
ftpproxy	String	HTTP proxy of the connected network.
ftpproxy6	String	IPv6 HTTP proxy of the connected network.
httpproxy	String	HTTP proxy of the connected network.
httpproxy6	String	IPv6 HTTP proxy of the connected network.
httpproxylogin	String	User name and password (`username:password`).
httpproxyloginrequired	Boolean	Indicates whether (`true|false`) the HTTP proxy requires login credentials.
httpsproxy	String	HTTP proxy of the connected network.
httpsproxy6	String	IPv6 HTTP proxy of the connected network.

/pps/services/networking/all/status_public

Status object for the currently preferred network interface

This object contains status information for the *currently preferred* network interface (i.e., the currently active interface when running in station mode.) Further details about this particular interface are available it its object in the `/pps/services/networking/all/interfaces/<interface>` directory.

Publishers

Network Manager

Subscribers

Any app

Attributes

Attribute	Data type	Description
default_gateway	JSON	Gateway address.
default_interface	String	Network interface name from `ifconfig` (e.g., `en0`).
default_interface4	String	IPv4 netowrk interface.
default_interface6	String	IPv6 netowrk interface.
fib	Number	FIB number.
ftpproxy	String	HTTP proxy of the connected network.
ftpproxy6	String	IPv6 HTTP proxy of the connected network.
httpproxy	String	HTTP proxy of the connected network.
httpproxy6	String	IPv6 HTTP proxy of the connected network.
httpproxyloginrequired	Boolean	Indicates whether (`true|false`) the HTTP proxy requires login credentials.
httpsproxy	String	HTTP proxy of the connected network.
httpsproxy6	String	IPv6 HTTP proxy of the connected network.
ip4_ok	String	A `yes` here indicates whether IPv4 connectivity is available. Possible error strings: • `error_no_ip_addr`

Attribute	Data type	Description
		• `error_no_ip_gateway` • `error_no_nameserver` • `error_not_configured` • `error_not_connected` • `error_not_up`
ip6_ok	String	Same as for *ip4_ok*, but for IPv6 connectivity.
ip_ok	String	General status attribute for IP connectivity.
nameservers	JSON	Array of nameserver addresses.
priority	JSON	Name of the currently preferred network interface.
searchdomains	String	Array of strings to be used for DNS resolution.
cmd_output	String	Last line of output from `msg::cmd`.

/pps/services/networking/control

The Network Manager listens for commands on this control object

Publishers

Any app

Subscribers

Network Manager; any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/networking/control` object are of the form:

msg::*command_string*\nid::*ID_number*\ndat:json:{*JSON_data*}

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

res::*command_string*\nid::*ID_number*\ndat:json:{*JSON_data*}\nerr::*error_description*

Commands

The control object accepts the following commands:

`net_connected`

Informs the Network Manager of a network link becoming available. Contains the connected interface and specified networking parameters. The *interface* is that given by the `ifconfig` utility.

`net_disconnecting`

Informs the Network Manager of an imminent shutdown of the specified *interface*, allowing clients to clean up gracefully before the interface is torn down.

The Network Manager will publish notice of the impending shutdown to the

`/pps/services/networking/all/interfaces/<interface>` object.

`net_disconnected`

The *interface* that was disconnected.

`net_dyn`

Supplies the Network Manager with dynamic configuration data. The response will contain a simple `err::` attribute on error, empty on success.

The following table shows the command format:

msg::	id::	dat:json:
`net_connected`	Number	`["`*`interface`*`"{"`*`parameter`*`":"`*`value`*`", ...}]` (see below)
`net_disconnected`	Number	*interface*
`net_disconnecting`	Number	`["`*`interface`*`" | "`*`interface`*`",{"deadline":`*`milliseconds`*`}]`
`net_dyn`	Number	`["`*`interface`*`",{"gateway":"`*`addr`*`","nameservers":["`*`addr`*`", "`*`addr`*`"],"searchdomains":"`*`domain`*`"}]`

Networking parameters

Parameter	Description
ftpproxy	IPv4 FTP proxy.
ftpproxy6	IPv6 FTP proxy.
htpproxy	IPv4 HTTP proxy.
htpproxy6	IPv6 HTTP proxy.
httpsroxy	IPv4 HTTPS proxy.
httpsproxy6	IPv6 HTTPS proxy.
manual	Possible values: • `yes`—if set, these settings apply: • `ip_address=` • `gateway=` • `netmask=` • `nameservers=` • `searchdomains=`

Parameter	Description
	• no—if set, these settings apply: • dhcp=on|off|auto • dhcp6=on|off|auto
manual6	Possible values: • yes—if set, these settings apply: • ip6_address= • ip6_netmask=
type	The type of network interface. Possible values: • bb (any BlackBerry Bridge BIS-B/BES-B or BBIO HTTP proxy connection) • bluetooth_dun (any Bluetooth tethering interface) • cellular (any cellular network interface) • usb (any direct USB cable to a PC or Mac) • vpn (any VPN tunnel) • wifi (any wireless network interface) • wired (any wired Ethernet interface)

Requesting a ping or traceroute

You can send `ping` or `traceroute` networking commands in the `dat::` field. Reply will contain a simple `err::` attribute on error, empty on success.

For example, a client can write:

```
msg::cmd
id::5
dat::ping -n -c4 10.42.116.1
```

/pps/services/networking/proxy

Duplicate of `/pps/services/networking/all/proxy`

This is a duplicate of the `/pps/services/networking/all/proxy` object.

/pps/services/networking/status

Duplicate of `/pps/services/networking/all/status_public`

This is a duplicate of the `/pps/services/networking/all/status_public` object.

/pps/services/networking/status_public

Duplicate of `/pps/services/networking/all/status_public`

This is a duplicate of the `/pps/services/networking/all/status_public` object.

/pps/services/tethering/control

The Tether Manager listens for commands on this object

Publishers

Any app

Subscribers

Tether Manager; any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/tethering/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\ndat::`*`parameter_data`*

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

`res::`*`command_string`*`\nid::`*`ID_number`*`\nerr::`*`error_description`*

Messages sent by the client

The control object accepts the following commands:

msg::	dat::	Description
`disconnect_client`	`{"mac":`*`MAC_address`*`}`	Disconnect from the specified client.
`resume`	n/a	Bring `tetherman` out of the suspended state while waiting for a confirmation from the user.
`retrieve_profile`	`"type":"lan" | "wan"`	Retrieve configuration values stored from previous tethering sessions for the specified interface type (`lan` | `wan`). If an error is in the response, this means the tethering session will have to be restarted. If the data field is empty in the response, this means

msg::	dat::	Description
		that no value was stored from the previous session.
`start`	The following fields are used: • `"lantype":"`*`lan_interface_type`*`"` • `"lancfg":"{`*`lancfg`*`}"` • `"lanprofile":"{`*`lanprofile`*`}"` • `"wantype":"`*`wan_interface_type`*`"` where: ***lan_interface_type*** `wifiap` (Wi-Fi access point) ***lancfg*** • `gateway_address` (IP address of the device on the LAN side) • `gateway_subnet` (subnet mask of the device on the LAN side) • `dhcp_lease_time` (DHCP server lease timer) ***lanprofile*** • *ssid* (e.g., `car_Hotspot`) • *security_type* (e.g., `wpa_mixed`) • *passphrase* (e.g., `testtest`) • *max_num_sta* (e.g., `5`) ***wan_interface_type*** `bridge`	Start the tethering session using the specified interface parameters.
`stop`	n/a	Stop the tethering session.
`update_profile`	`"type":"lan" | "wan"`	Update the profile for the specified interface type (`lan` | `wan`).

Examples

If we want to observe responses from the `tetherman` service, we need to force the shell to keep the file descriptor open (because this is a server object). Here's an example `start` command:

```
# (exec 3<>/pps/services/tethering/control && cat >&3 && sleep 1
&& cat <&3)<<END
msg::start
id::123
dat::{"lantype":"wlanap","lancfg":{"gateway_address":"192.168.0.15","gateway_subnet":"255.255.255.0",
    "dhcp_lease_time":120},"wantype":"bridge","lanprofile":{"ssid":"car_Hotspot","security_type":"wpa_mixed",
    "passphrase":"openseseme","max_num_sta":5,"privacy":true,"block_conn":false}}
END
```

/pps/services/tethering/status

Status object for the Tether Manager

Publishers

Tether Manager; any app

Subscribers

Any app

Overview

The Tether Manager service (`tetherman`) publishes status information for tethering sessions. Here's a sample object:

```
[n]@status
activation:b:false
max_num_client2:b:false
reason::TETHER_INIT_EVENT
state::INACTIVE
```

Attributes

Attribute	Data type	Description
activation	Boolean	Indicates whether tethering has been activated.
max_num_client2	Boolean	Indicates whether the maximum number of clients has been reached.
reason	String	Reason for entering the current state. Values: • `LAN_FAILED_EVENT` • `LAN_IP_NOT_BLOCKED_EVENT` • `LAN_NOT_READY_EVENT` • `LAN_READY_EVENT` • `TETHER_INIT_EVENT` • `TETHER_START_EVENT` • `TETHER_STARTUP_CHECK_FAILED_EVENT` • `TETHER_STOP_EVENT` • `TETHER_USER_ACCEPT_EVENT` • `TIMER_ACTIVATION_EXPIRED` • `TIMER_SHUTDOWN_EXPIRED` • `TIMER_STARTUP_EXPIRED`

Attribute	Data type	Description
		• TIMER_SUSPEND_EXPIRED • WAN_FAILED_EVENT • WAN_HANDOVER_CLEAR_EVENT • WAN_HANDOVER_ALLOWED_EVENT • WAN_HANDOVER_NOT_ALLOWED_EVENT • WAN_IP_BLOCKED_EVENT • WAN_NOT_READY_EVENT • WAN_OUT_OF_COVERAGE_EVENT • WAN_READY_EVENT
state	String	Current state of the tetherman service. Values: • ACTIVE • INACTIVE • LAN_STARTING • SHUTDOWN • SHUTDOWN_WAIT_LAN • SHUTDOWN_WAIT_WAN • SUSPEND • WAN_STARTING

/pps/services/update/control

Software Update manager listens for messages on this control object

Publishers

Any app

Subscribers

Software Update — legacy HMI plugin (`swud-legacy-hmi.so`)

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message format

Commands sent to the `/pps/services/update/control` object are of the form:

`msg::`*`command_string`*`\nid::`*`ID_number`*`\ndat:json:{`*`JSON_data`*`}`

The `id::`*`ID_number`* field is optional.

Attributes

Attribute	Data type	Description
cmd	Number	Values 1,2,3 specify the Update Manager's behavior: • `1` = Reserved for future use. • `2` = Start an update (has no effect if update is unavailable). • `3` = Cancel this update.
data	String	Miscellaneous data; depends on the command used (for future use).

Event behavior

The Update Manager knows whether an update is available when the user inserts a USB stick containing a valid update file.

The update must contain a delta file (`.mld`) as well as a `.manifest` file. For details, see “Software Updates” in the *System Services Reference*.

When it determines whether an update is available (e.g., the user inserts a USB stick with a valid update), the Update Manager sets *updateAvailable* to `1` on the `/pps/services/update/status` object. The HMI prompts the user and then writes the appropriate command to this control object (`cmd:n:2` to install).

Example

Start an update:

```
# echo "cmd:n:2" > /pps/services/update/control
```

/pps/services/update/settings

Software Update manager reads this object for update settings

Publishers

Any app

Subscribers

Software Update — client configuration plugin (`swud-client-config.so`)

Attributes

Attribute	Data type	Description
`localUpdatesEnabled`	Boolean	Determines whether local (e.g., USB) software updates are enabled on the update client.
`updateGracePeriod`	Number (range: `0` to $2^{64}-1$)	The time (in seconds) after which an update will become mandatory and cannot be deferred.
`maxUpdateRetries`	Number (range: `0` to $2^{32}-1$)	Maximum number of retries allowed per software update.

Sample settings object

```
@settings
localUpdatesEnabled:b:true
maxUpdateRetries:n:5
updateGracePeriod:n:604800
```

/pps/services/update/status

Software Update manager communicates to apps via this status object

Publishers

Software Update — legacy HMI plugin (`swud-legacy-hmi.so`)

Subscribers

Any app

Attributes

Attribute	Data type	Description
`updateAvailable`	Number	Values `0` and `1` indicate update status: • `0` = No update available • `1` = Update available
`updateDetails`	JSON	Contains the following: • `sourceVersion`, a string indicating the version to update from • `targetVersion`, a string indicating the version to update to • `details`, a string giving a brief description of the update • `source`, a number indicating where the update came from: • `1` = USB • `n/a` (all other values are reserved)
`updateError`	String (up to 256 characters)	This attribute will be stored in an update object when an update becomes available. If the current version number of the target system fails to match the `sourceVersion` attribute populated when the update was discovered, the error description will indicate that the two versions don't match and that the update is invalid.

Sample status objects

1. When no update is available, the status object looks like this:

```
[n]@status
updateAvailable:n:0
```

2. When a valid update is available, the status object will look something like this:

```
[n]@status
updateAvailable:n:1
updateDetails:json:{"sourceVersion":"162","targetVersion":"9999","details":
"Geolocation Test","source":1}
```

3. The following status object indicates that an update has been discovered, but the update is considered unavailable because the versions don't match:

```
[n]@status
updateAvailable:n:0
updateDetails:json:{"sourceVersion":"87","targetVersion":"9999","details":"
Geolocation Test","source":1}
updateError::Current version (162) does not match source version
 (87),
invalid update
```

The update must contain a delta file (`.mld`) as well as a `.manifest` file. For details, see “Software Updates” in the *System Services Reference*.

When it determines whether an update is available (e.g., the user inserts a USB stick with a valid update), the Update Manager sets *updateAvailable* to `1` on this status object. The HMI prompts the user and then writes the appropriate command (`cmd:n:2` to start the update) to the `/pps/services/update/control` object.

/pps/services/update/target

Holds attributes that identify the target to receive the update

Publishers

Any app

Subscribers

Software Update — self-update plugin (`rb-self-update.so`)

Attributes

Attribute	Data type	Description
`hardwareID`	String (up to 100 characters)	Unique hardware identifier for the target. This must match a corresponding hardware ID in the manifest file for the software update.
`serialNum`	String (up to 100 characters)	Serial number of the source version (e.g., `16250279`).
`vendorID`	String (up to 100 characters)	Unique vendor identifier for the target. This must match a corresponding vendor ID in the manifest file for the software update.

Sample target object

```
@target
hardwareID::CAR2.1
serialNum::16250279
vendorID::QNX
```

/pps/services/vnc/discovery/

Directory for USB devices that could be MirrorLink devices

Publishers

`RealVNC`

Subscribers

`mlink-daemon`; any app

The `/pps/services/vnc/discovery/` directory contains a subdirectory (called `/usb/`) for objects that are created for USB devices found by the RealVNC discovery scripts. The RealVNC USB device provider, a dynamically loaded library that's used by `mlink-daemon`, reads this directory for information on the discovered USB devices.

The name of each object under the `/usb/` directory is the product ID in the latest builds. Here's a sample object:

```
[n]@0x685d
bus::0
dev::4
pid::0x685d
type::USB
vid::0x4e8
```

Attributes

Attribute	Data type	Description
`bus`	String	Bus number.
`dev`	String	Device number.
`pid`	String	Product ID.
`type`	String	Type of device, which is always `USB` on the QNX CAR platform.
`vid`	String	Vendor ID.

/pps/services/wifi/control

The WLAN service listens for commands on this object

Publishers

WLAN service; any app

Subscribers

Any app

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/services/wifi/control` object are of the form:

```
msg::command_string\nid::ID_number\ndat:json:{JSON_data}
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID_number\ndat:json:{JSON_data}\nerr::er
ror_description
```

Commands

The control object accepts the following commands:

Command	Values for dat field	Description
`Scan`	n/a	Request a connection scan. Note that scan results are not returned immediately. When the results are available, they're updated in the */pps/services/wifi/status* object (or can be retrieved with the `scan_results` command).

Command	Values for `dat` field	Description
`scan_results`	n/a	Request the last available scan results. This command doesn't initiate a new scan. See the /pps/services/wifi/status object for details.
`wifi_power`	One of the following: • `on` (station power on) • `off` (station power off) • `graceful_shutdown` (WLAN standby state) • `low_power` (regular WLAN operation mode) • `default` (WLAN default power mode)	Configure WLAN radio power modes.
`Wnet`	*netID*	Query for a saved profile with specified network ID. Note that password-type attributes are returned with null entries for security purposes.
`wnet_delete`	*netID*	Delete the specified profile. If the profile is the currently connected network, the network will be disconnected and the profile deleted.
`wnet_disable`	*netID*	Disable the specified saved profile. If the profile is the currently connected network, the network will be disconnected.
`wnet_disableall`	n/a	Disable all saved profiles.
`wnet_enable`	*netID*	Enable the specified saved profile. This may trigger a

Command	Values for dat field	Description
		new connection to the enabled profile.
wnet_enableall	n/a	Enable all saved profiles.
wnet_new	{*JSON_profile_attributes*} (See "*Profile attributes* (p. 232)" below.)	Add a new profile to the list of saved networks.
wnet_select	*netID*	Select the specified saved profile for connection, enabling the specified profile and disabling all other profiles. If a different profile is connected, it will be disconnected from that network.
wnet_update	*netID{JSON_profile_attributes}*	Update the *attributes* (p. 232) of an existing network profile. If the profile is the currently connected network, the network will be disconnected and the profile updated. Note that profile must be pushed again in the dat (can be first queried), even for those attributes that aren't modified. The timestamp of the provided profile must match the existing profile being updated to ensure it's the same network that the client wants updated (otherwise, EINVAL is returned). To update a *protected* attribute (i.e., password, psk, or wep_key0), precede the attribute name with new. For example, to

Command	Values for dat field	Description
		change psk, send the field newpsk. The wnet_update doesn't affect the _enable attribute. To change _enable, use wnet_enable or wnet_disable.

Profile attributes

Attribute	Values	Description
ap_handover	0 | 1	Specify whether to perform handovers between access points. If set to 1, the profile will use roaming.
auth_alg	OPEN	Authentication type for a WEP network.
band_select	0 | 1 | 2	Indicates which band the profile should connect on: • 0 = dual • 1 = 2.4G only • 2 = 5G only
ca_cert	*file_path*	Full path to the server certificate.
ca_path	*dir_path*	Full path to the certificate store.
client_cert	*file_path*	Full path to the client certificate.
eap	• AKA (Authentication and Key Agreement)	Type of EAP used by this network.

Attribute	Values	Description
	• FAST (Flexible Authentication via Secure Tunneling) • PEAP (Protected Extensible Authentication Protocol) • SIM (Subscriber Identity Module) • TLS (Transport Layer Security) • TTLS (Tunneled Transport Layer Security)	
_editability	• editable • credentials_only • uneditable	Controls whether the user can edit the profile.
_enable	0 | 1	Determines whether the profile is enabled (1) or disabled and won't be used (0).
enterprise	true | false	Indicates whether this is an enterprise profile (EMA/BDS).
group_id	*alphanumeric_id*	An ID value used by wpa_pps clients to identify the group this profile belongs to. This is not used by wpa_pps itself.
identity	*alphanumeric_id*	Username for the EAP session.
key_mgmt	• NONE • WPA-EAP • WPA-PSK	Security type used by the network.

Attribute	Values	Description
_name	*alphanumeric_name*	Arbitrary name of the profile.
owner	• BRIDGE • CARRIER_MNGR • EMA • UI • WPS	Indicates which client owns the profile. This field is used to determine the precedence for new profiles added and for profile priority.
pac_file	*file_path*	Path to the PAC file.
password	*alphanumeric_password*	The password for the EAP session. Note that this is a *protected* attribute.
phase1	FAST_PROVISIONING=1	Authenticates using PAC and sets up tunnel key.
phase2	• MSCHAPV2 (Microsoft Challenge Handshake Authentication Protocol version 2) • GTC (Generic Token Card)	Authentication method for the inner tunnel of an EAP connection.
pin	*number*	PIN for the EAP-SIM or EAP-AKA connection.
private_key	*file_path*	Full path to the client private key.
private_key_passwd	*alphanumeric_password*	Password for the client private key.
psk	*alphanumeric_or_hex*	The WPA passkey. Note that this is a *protected* attribute.
_saved	0 | 1	Controls how the UI recognizes the profile: • 0 = UI won't display the profile

Attribute	Values	Description
		• 1 = Profile is "saved" and recognized by the UI as valid
scan_ssid	0 | 1	Controls whether WLAN will do active scans to locate the network: • 0 = No active scans • 1 = Active scans (profile is marked "hidden")
ssid	*alphanumeric_name*	Name of the network to connect to.
_user_enable	0 | 1	Determines whether the profile is enabled by the user (1) or is disabled and won't be used (0).
_visibility	visible	Determines whether the profile is visible to the user.
wep_key0	*hex_characters*	WEP key used by this network. Note that this is a *protected* attribute.

Examples

1. If we want to observe responses from the WLAN service, we need to force the shell to keep the file descriptor open (because this is a server object). First we send a wnet_new command to set up a new profile:

   ```
   (exec 3<>/pps/services/wifi/control && cat >&3 && cat <&3)<<END
   msg::wnet_new
   id::123
   dat:json:{"ssid":"wifi_test","key_mgmt":"NONE"}
   END
   ```

   The control object might now look like this:

   ```
   @control
   res::wnet_new
   id::123
   dat::3
   ```

2. We can now use the *netID* that was returned (i.e., `3`) in further commands, such as `wnet_select`:

```
(exec 3<>/pps/services/wifi/control && cat >&3 && cat <&3)<<END
msg::wnet_select
id::124
dat::3
END
```

/pps/services/wifi/status

Status object for the WLAN service

Publishers

WLAN service; any app

Subscribers

Any app

Overview

The WLAN service publishes status information to this object. Here's a sample object:

```
[n]@status
ip_config:json:
    {"manual":"no","ip_addresses":["fe80:14::e2c7:9dff:fe4a:8e97/ffff:ffff:ffff:ffff::","192.168.99.135/255.255.255.0"],"gateway":
    ["10.222.96.1"],"nameservers":["10.222.146.10"],"searchdomains":["ott.qnx.com"]}
lan_tether_interface::tiw_sap0
lan_tether_power::off
lan_tether_state::not_ready
lan_tether_status:json:{"max_num_clients":8,"num_clients":0,"privacy":true,"block_conn":false,"mac":"e0:c7:9d:4a:8e:97"}
last_event::<3>CTRL-EVENT-BSS-REMOVED 327 e8:40:40:72:2a:5b
scan_results:json:[{"bssid":"20:aa:4b:85:5a:b8","frequency":"5785","signal_level":"-58","flags":["ESS"],"ssid":"car2-twonky"},
{"bssid":"00:0d:54:a0:18:29","frequency":"2412","signal_level":"-66","flags":["WPA-PSK-TKIP","ESS"],"ssid":"3Com-qa"},
{"bssid":"2c:36:f8:b8:cb:3c","frequency":"5200","signal_level":"-80","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"2c:36:f8:b8:cb:3f","frequency":"5200","signal_level":"-80","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"2c:36:f8:b8:cb:3d","frequency":"5200","signal_level":"-80","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"2c:36:f8:b8:cb:3e","frequency":"5200","signal_level":"-80","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"00:08:30:e5:e3:2e","frequency":"5220","signal_level":"-82","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"00:08:30:e5:e3:2d","frequency":"5220","signal_level":"-82","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"00:08:30:e5:e3:2c","frequency":"5220","signal_level":"-82","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"00:08:30:e5:e3:2f","frequency":"5220","signal_level":"-82","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:2a:5d","frequency":"5745","signal_level":"-88","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"e8:40:40:72:2a:5e","frequency":"5745","signal_level":"-87","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"e8:40:40:72:2a:5f","frequency":"5745","signal_level":"-87","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:2a:5c","frequency":"5745","signal_level":"-88","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"20:aa:4b:85:5a:b6","frequency":"2412","signal_level":"-51","flags":["ESS"],"ssid":"car2-twonky2"},
{"bssid":"70:ca:9b:9a:c7:f1","frequency":"2462","signal_level":"-55","flags":["WPS","ESS"],"ssid":"hb-cisco"},
{"bssid":"76:ca:9b:9a:c7:f1","frequency":"2462","signal_level":"-54","flags":["ESS"],"ssid":"hb-cisco-2"},
{"bssid":"2c:36:f8:b8:cb:3b","frequency":"5200","signal_level":"-80","flags":["ESS"],"ssid":"QNX-GUEST"},
{"bssid":"00:08:30:e5:e3:2b","frequency":"5220","signal_level":"-83","flags":["ESS"],"ssid":"QNX-GUEST"},
{"bssid":"e8:40:40:72:2a:5b","frequency":"5745","signal_level":"-89","flags":["ESS"],"ssid":"QNX-GUEST"},
{"bssid":"b8:be:bf:ef:71:41","frequency":"2462","signal_level":"-79","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"b8:be:bf:ef:71:43","frequency":"2462","signal_level":"-79","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"e8:40:40:72:22:2c","frequency":"5805","signal_level":"-91","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"b8:be:bf:ef:71:4d","frequency":"5180","signal_level":"-88","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"b8:be:bf:ef:71:4c","frequency":"5180","signal_level":"-87","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"b8:be:bf:ef:71:42","frequency":"2462","signal_level":"-79","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"b8:be:bf:ef:71:4e","frequency":"5180","signal_level":"-87","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"e8:40:40:72:22:2e","frequency":"5805","signal_level":"-90","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"02:1a:11:ff:c7:67","frequency":"2437","signal_level":"-76","flags":["ESS"],"ssid":""},
{"bssid":"e8:40:40:72:22:2d","frequency":"5805","signal_level":"-93","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"02:1a:11:f6:a5:23","frequency":"2437","signal_level":"-81","flags":["ESS"],"ssid":"FBI Van6"},
{"bssid":"e8:40:40:72:22:2b","frequency":"5805","signal_level":"-92","flags":["ESS"],"ssid":"QNX-GUEST"},
{"bssid":"b8:be:bf:ef:71:40","frequency":"2462","signal_level":"-79","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:22:23","frequency":"2437","signal_level":"-75","flags":["WPA2-EAP-TKIP+CCMP","ESS"],"ssid":"corpbb"},
{"bssid":"e8:40:40:72:22:21","frequency":"2437","signal_level":"-75","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"devel05"},
{"bssid":"b8:be:bf:ef:71:4f","frequency":"5180","signal_level":"-88","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:22:2f","frequency":"5805","signal_level":"-91","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:22:20","frequency":"2437","signal_level":"-75","flags":["WPA2-EAP-CCMP","ESS"],"ssid":"QNX-CORP"},
{"bssid":"e8:40:40:72:22:22","frequency":"2437","signal_level":"-75","flags":["WPA-EAP-TKIP","WPA2-EAP-CCMP","ESS"],"ssid":"corpwifi"},
{"bssid":"e8:40:40:72:22:24","frequency":"2437","signal_level":"-76","flags":["ESS"],"ssid":"QNX-GUEST"},
{"bssid":"02:1c:50:0c:bd:ee","frequency":"2437","signal_level":"-76","flags":["ESS"],"ssid":"DIRECT-yDDMP-MST60"},
{"bssid":"b8:be:bf:ef:71:4b","frequency":"5180","signal_level":"-88","flags":["ESS"],"ssid":"QNX-GUEST"}]
wifi_ap_status:json:{"bssid":"e0:c7:9d:4a:8e:97","channel":6,"band":"bg","max_rate":54,"ssid":"ssid-qnxcar","pairwise_cipher":"TKIP CCMP",
    "group_cipher":"TKIP CCMP","key_mgmt":"WPA-PSK"}
wifi_connected:b:true
wifi_interface::tiw_sta0
wifi_macaddress::e0:c7:9d:4a:8e:97
wifi_power::on
wifi_role:json:{"interface":"tiw_sta0","role":"WIFI_ROLE_STA"}
wifi_status:json:{"bssid":"20:aa:4b:85:5a:b8","ssid":"car2-twonky","id":"2","mode":"station","pairwise_cipher":"NONE","group_cipher":"NONE",
    "key_mgmt":"NONE","wpa_state":"COMPLETED","ip_address":"192.168.99.135","address":"e0:c7:9d:4a:8e:97"}
wnet_connected::2
```

Attributes

Attribute	Data type	Description
ip_config	JSON	IP configuration: • `manual` • `ip_addresses` • `gateway` • `nameservers` • `searchdomains` Note that this is cleared when *wifi_connected* changes.
lan_tether_interface	String	Interface used for tethering (e.g. `tiw_sap0`).
lan_tether_power	String	Indicates whether the tether interface is on or off.
lan_tether_state	String	Indicates whether the interface is ready for tethering (e.g., `not_ready`).
lan_tether_status	JSON	Interface status: • `max_num_clients` • `num_clients` • `privacy` • `block_conn` • `mac`
last_event	String	Last event message received by WLAN Manager from `wpa_supplicant`.
scan_results	JSON	Contains the latest scan results received by WLAN manager from the WLAN driver. Results are unfiltered, showing hidden networks as well as multiple BSSIDs on the same SSID network. Applies only when the interface is in station role.
wifi_ap_status	JSON	Access point status: • `band` • `bssid` • `channel` • `group_cipher` • `key_mgmt` • `max_rate` • `pairwise_cipher` • `ssid`

Attribute	Data type	Description
wifi_connected	Boolean	Indicates (`true` | `false`) whether the WLAN station is connected.
wifi_interface	String	WLAN interface name (e.g., `tiw_sta0`).
wifi_macaddress	String	WLAN interface's MAC address.
wifi_power	String	Indicates whether the WLAN interface is on or off. Applies only when the interface is in station role.
wifi_role	JSON	Indicates the active WLAN interface and its role (e.g., `wifi_role:json:{"interface":"tiw_sta0","role":"sta"}`). Possible roles are: • `ap`—access point • `sta`—station
wifi_status	JSON	General WLAN status: • `address` • `bssid` • `group_cipher` • `id` • `ip_address` • `key_mgmt` • `mode` • `pairwise_cipher` • `ssid` • `wpa_state`
wnet_connected	String	Network ID of the currently connected profile. Applies only when the interface is in station role and when *wifi_connected* is `true`.

/pps/system/keyboard/control

The Keyboard service listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

Keyboard service

This type of object is known as a *server object*, a special PPS object designed for point-to-point communication between a server and one or more clients. For details, see "Server objects" in the *Persistent Publish/Subscribe Developer's Guide*.

Message/response format

Commands sent to the `/pps/system/keyboard/control` object are of the form:

```
msg::command_string\nid::ID\ndat:json:{JSON_data}
```

Responses always reflect the *command_string* and *ID_number* that were sent in the message, along with any errors:

```
res::command_string\nid::ID\ndat:json:{JSON_data}\n error::error_de
scription
```

Commands

msg::	id::	dat:json:
Messages to send to the control object: • `show` • `hide`	The message's ID string (usually a number, but can be anything).	JSON data (payload) related to the message.

Examples

Show the keyboard:

```
echo "msg::show\nid::1\ndat:json:{}" > /pps/system/keyboard/control
```

Hide the keyboard:

```
echo "msg::hide\nid::2\ndat:json:{}" > /pps/system/keyboard/control
```

/pps/system/keyboard/status

The Keyboard service uses this object to reflect the keyboard's current state

Publishers

Keyboard service

Subscribers

Any app

Attributes

Attribute	Data type	Description
size	Number	Specifies height of the keyboard in pixels (range is 1 to screen height; default is 190).
visible	Boolean	Indicates whether keyboard is visible. The Keyboard service sets this attribute after receiving a `show` or `hide` command from the `/pps/system/keyboard/control` object.

/pps/system/navigator/appdata

Holds app-specific data to be retrieved by the app on launch

Publishers

Applications Navigator

Subscribers

Any app

The Applications Navigator is a service that controls how applications appear on the display. This is not to be confused with GPS turn-by-turn *navigation* (see `/pps/qnxcar/navigation/control`).

Overview

As each app in the Apps Section screen is launched, the app's name is written to the `/pps/system/navigator/appdata` object. For example, if the user taps the **Settings** button as well as the **BestParking** button, the object will look like this:

```
@appdata
[n]BestParking::
[n]Settings::
```

Each app can then read this object and retrieve any data published here.

/pps/system/navigator/applications/applications

The Applications Navigator publishes a list of installed apps to this object

Publishers

Applications Navigator

Subscribers

Any app

The Applications Navigator is a service that controls how applications appear on the display. This is not to be confused with GPS turn-by-turn *navigation* (see `/pps/qnxcar/navigation/control`).

Overview

Each app installed on the system appears in the `/pps/system/navigator/applications/applications` object:

```
@applications
AppSection.testDev_AppSection_2fc201a4::{86x86}native/default-icon.png,AppSection,,,auto,,
BestParking.testDev_BestParkinga4a73514::{48x48}native/appicon.png,BestParking,,,auto,,
Calendar.testDev_Calendar___f9395f5e::{86x86}native/icon.png,Calendar,bridge,,auto,,
Communication.testDev_mmunicationf1e9ffb6::{60x60}native/icon.png,Communication,,,auto,,
Contacts.testDev_Contacts___e207c473::{86x86}native/icon.png,Contacts,bridge,,auto,,
HelloWebWorks.testDev_lloWebWorks1fa80f60::native/icon.png,HelloWebWorks,,,,,
Home.testDev_Home_______2268ff__::{60x60}native/icon.png,Home,,,auto,,
Map.testDev_Map________12d3c___::{86x86}native/icon.png,Map,,,auto,,
MediaPlayer.testDev_MediaPlayer49ba23c5::{60x60}native/icon.png,MediaPlayer,,,auto,,
Memopad.testDev_Memopad____9bcd70f9::{86x86}native/icon.png,Memopad,bridge,,auto,,
Messages.testDev_Messages___e84f656c::{86x86}native/icon.png,Messages,bridge,,auto,,
Navigator.testDev_Navigator__a4514a37::{60x60}native/icon.png,Navigator,,,auto,,
Pandora.testDev_Pandora____33b5d5f7::{512x512}native/icon.png,Pandora,,,auto,,
PeaksAndValleys.testDev_sAndValleys6bb91d91::{86x86}native/icon.png,PeaksAndValleys,,,auto,,
Scout.testRel_Scout______4c04ede_::native/icon.png,Scout,sys,native/splash.png,landscape,,
Settings.testDev_Settings___595d2043::{86x86}native/icon.png,Settings,,,auto,,
Slacker.testDev_Slacker____e476201d::{90x90}native/icon.png,Slacker Radio,,,auto,,
Status.testDev_Status_____9432bc12::{60x60}native/icon.png,Status,,,auto,,
TuneIn.testDev_TuneIn_____95fb575d::{96x96}native/icon.png,TuneIn Radio,,,auto,,
Weather.testDev_Weather____ac24cfd4::{86x86}native/icon.png,Weather,,,auto,,
WeatherNetwork.testDev_therNetworke376bba_::{84x84}native/icon.png,WeatherNetwork,,,auto,,
carcontrol.testDev_carcontrol_21522f09::{86x86}native/default-icon.png,CarControl,,,auto,,
htmlgears.testDev_htmlgears__9202e3f9::{60x60}native/icon.png,HTMLGears,,,auto,,
navigation.testDev_navigation_6f060a14::{86x86}native/default-icon.png,Navigation,,,auto,,
sys.browser.new.testRel_browser_new77e89439::native/browserIcon.png,Browser,core.media,,auto,,
sys.keyboard.testRel_ys_keyboard5435fde8::icon.png,sys.keyboard,,,,,
tunneltilt.testDev_tunneltilt_84be6f25::{86x86}native/data/images/tunnelTiltIcon.png,TunnelTilt,,,auto,,
```

Each line in the object file can hold the following information:

- the directory where the app is installed (under `/app`)
- the size and location of the app's icon
- the app's name

- the app's category in the App Launcher screen (**ALL**, **VEHICLE**, etc.)
- the app's splash screen
- the app's orientation (e.g., `auto`, `landscape`)

If MirrorLink devices are detected on the system, the `mlink-daemon` service creates shortcuts for MirrorLink apps (in the `/apps/` directory) and then publishes the apps to this object. A line for a MirrorLink app looks like this:

```
[n]ml0app.shortcut100::native/icon.png,Nokia Drive,me
dia,,auto,,
```

/pps/system/navigator/command

Shows application tab actions

Publishers

Applications Navigator

Subscribers

Any app

The Applications Navigator is a service that controls how applications appear on the display. This is not to be confused with GPS turn-by-turn *navigation* (see `/pps/qnxcar/navigation/control`).

Overview

The `/pps/system/navigator/command` object shows the state of the application tabs in the HMI. Here's a sample object:

```
@command
AppSection:json:{"action":"pause"}
Browser:json:{"action":"reselect"}
Communication:json:{"action":"pause"}
HTMLGears:json:{"action":"pause"}
HelloWebWorks:json:{"action":"reselect"}
Map:json:{"action":"reselect"}
MediaPlayer:json:{"action":"pause"}
Pandora:json:{"action":"reselect"}
PeaksAndValleys:json:{"action":"reselect"}
Scout:json:{"action":"resume"}
Settings:json:{"action":"reselect"}
TunnelTilt:json:{"action":"reselect"}
Weather:json:{"action":"reselect"}
WeatherNetwork:json:{"action":"pause"}
carcontrol:json:{"action":"pause"}
home:json:{"action":"pause"}
navigation:json:{"action":"pause"}
voicecontrol:json:{"action":"pause"}
```

Each line shows the application name, followed by the `json` data type, followed by the `"action":"`*`value`*`"` pair. The values for `"action"` can be:

- `pause`—the app is being told it's in the background, so it should stop CPU-intensive display tasks (e.g., drawing navigation maps)
- `reselect`—the app is being told of a special request, so it should go to its home screen

- `resume`—the app is being told it's in the foreground, so it can resume what it was doing before it paused (e.g., start drawing navigation maps again)

/pps/system/navigator/status/mobile_hotspot

The WIFI HOTSPOT feature publishes status information to this object

Publishers

WIFI HOTSPOT (in Settings app); any app

Subscribers

Any app

Overview

When the WIFI HOTSPOT control in the Settings app is used to set up a mobile hotspot, this object reports the various states (`on` | `off` | `connected`) involved. Here's a sample object:

```
@mobile_hotspot
[n]dat:json:{"mobile_hotspot":{"state":"connected"}}
```

/pps/system/navigator/status/tethering

Status object for tethering sessions

Publishers

Tether Manager

Subscribers

Any app

Overview

When a tethering session has been started or stopped, this object simply reports the state (`on` or `off`). Here's a sample object:

```
@tethering
[n]dat:json:{"tethering":{"state":"off"}}
```

For more information on tethering, see the following objects:

- `/pps/services/tethering/control`
- `/pps/services/tethering/status`

/pps/system/navigator/windowgroup

Publishes window group identifiers for the main HMI apps

Publishers

Any app

Subscribers

Applications Navigator

The Applications Navigator is a service that controls how applications appear on the display. This is not to be confused with GPS turn-by-turn *navigation* (see `/pps/qnxcar/navigation/control`).

Overview

The `/pps/system/navigator/windowgroup` object shows the window groups for the main apps in the HMI. Here's a sample object:

```
@windowgroup
[n]AppSection::736053855-360457-13-bb-wk-win-group
[n]Communication::3155946061-360457-12-bb-wk-win-group
[n]MediaPlayer_mmplayer::666425069-360457-10-bb-wk-win-group
[n]carcontrol::3181133917-360457-11-bb-wk-win-group
[n]navigation::1707991092-360457-9-bb-wk-win-group
```

/pps/system/navigator/windowparams

Shows height, width, and other window parameters for apps

Publishers

Any app

Subscribers

Applications Navigator

The Applications Navigator is a service that controls how applications appear on the display. This is not to be confused with GPS turn-by-turn *navigation* (see `/pps/qnxcar/navigation/control`).

Overview

The `/pps/system/navigator/windowparams` object shows various window parameters for applications. Here's a sample object:

```
@windowparams
MediaPlayer:json:{"x":0, "y":0, "h":480, "w":800, "zorder":0}
```

Window parameter	Description
h	Height (in pixels).
w	Width (in pixels).
x	The *x* dimension from the top left (in pixels).
y	The *y* dimension from the top left (in pixels).
zorder	The *z* order of the window.

Chapter 4
List of Objects Used Internally

For this release of the QNX CAR platform, the objects listed below are used internally by various system processes. Third-party applications won't need to read from or write to these objects. Note that this list may change with future releases.

PPS directories and objects used internally

- `/pps/applications/appremote`
- `/pps/servicedata/schedule`
- `/pps/services/audio/stats`
- `/pps/services/authentication/`
- `/pps/services/apkruntime/`
- `/pps/services/certmgr/`
- `/pps/services/confstr/`
- `/pps/services/deviceproperties`
- `/pps/services/dlna/dmcclient/dmr/networkstate/<dmr_uuid>`
- `/pps/services/dlna/dmcclient/dmr/playstate/<dmr_uuid>`
- `/pps/services/dlna/dmcclient/dms/networkstate/<dms_uuid>`
- `/pps/services/dmc/`
- `/pps/services/dmr/control`
- `/pps/services/dmr/rendererCtrl`
- `/pps/services/dmr/rendererStatus`
- `/pps/services/dmr/status`
- `/pps/services/input/context/<contextname>`
- `/pps/services/input/control`
- `/pps/services/mediaserver/settings`
- `/pps/services/mm-player/`
- `/pps/services/multimedia/mediacontroller/notifications`
- `/pps/services/multimedia/sound/`
- `/pps/services/multimedia/sync/`
- `/pps/services/network-time/status`
- `/pps/services/notification/`

- `/pps/services/power/shutdown/control`
- `/pps/services/private/deviceproperties`
- `/pps/services/samba/control`
- `/pps/services/samba/smb`
- `/pps/services/slogger2/notify`
- `/pps/services/slogger2/verbose`
- `/pps/services/system_info/control`
- `/pps/services/tztrans/control`
- `/pps/services/vpn/`
- `/pps/system/authorization/control`
- `/pps/system/bookmarks/`
- `/pps/system/development/control`
- `/pps/system/development/devmode`
- `/pps/system/installer/coreos/`
- `/pps/system/installer/hmi/lastupdate`
- `/pps/system/installer/registeredapps/`
- `/pps/system/installer/removedapps/`
- `/pps/system/installer/stagedapps/`
- `/pps/system/installer/upd/current`
- `/pps/system/installer/upd/deferred`
- `/pps/system/launcher_priority`
- `/pps/system/navigator/status/app-timestamps`
- `/pps/system/nvram/deviceinfo`
- `/pps/system/power/dev/bus`
- `/pps/system/power/funcstatus/user_activity`
- `/pps/system/sapphire/`

Index