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: March 27, 2015

Contents

About This Reference

The *PPS Objects Reference* describes each PPS object supplied with the QNX SDK for Apps and Media 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* (p. 11)
Custom objects	*Setting Up Your Own Objects* (p. 19)
Details of each PPS object	*PPS Objects Reference Pages* (p. 21)
Objects used internally by system processes	*List of Objects Used Internally* (p. 103)

For more information about the PPS service itself, see:

- the PPS chapter in the QNX Neutrino *System Architecture* guide
- the 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**).

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

Application Launcher

- ***/pps/services/app-launcher*** (p. 34)
- ***/pps/services/launcher/control*** (p. 61)

Audio

- ***/pps/services/audio/audio_router_control*** (p. 36)
- ***/pps/services/audio/audio_router_status*** (p. 41)
- ***/pps/services/audio/control*** (p. 43)
- ***/pps/services/audio/devices/*** (p. 49)
- ***/pps/services/audio/status*** (p. 53)
- ***/pps/services/audio/types/*** (p. 54)
- ***/pps/services/audio/voice_status*** (p. 56)

Authorization

- ***/pps/accounts/*** (p. 22)

Geolocation

- ***/pps/services/geolocation/control*** (p. 58)
- ***/pps/services/geolocation/status*** (p. 60)

Keyboard

Multimedia

Navigator (Applications Window Manager)

Networking

System Information

Typographical conventions

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

The following table summarizes our conventions:

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

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

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

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

Notes point out something important or useful.

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

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

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 SDK for Apps and Media is built on the *Persistent Publish/Subscribe* (PPS) service, a simple filesystem-based facility that provides information persistence across reboots. Small and extensible, PPS allows interfacing from almost any higher-level language that supports 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 apps and HMI 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.

Apps and HMI services often use a *control* object for sending commands and a corresponding *status* object for publishing responses.

Client apps can read the special **.all** object to get notifications of changes to *all* the objects in a directory. They 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

PPS supports point-to-point communication between a server and one or more clients. An app can designate itself as the *server* when creating a PPS object. When a client writes to this object, only the server gets the message. PPS appends a unique identifier to the object name so that the server knows which client app is sending the message.

When the server replies, it must append the same identifier to the object name so that the response is sent only to the client indicated by that identifier. In this communication mode, both the server and the clients read from and write to the object. For more 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 the path to the 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 the default size of the delta backlog, in kilobytes (default is 256 kilobytes).

-l *argument*

Set the object 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.

You can set these options:

backlog=*size*

Set the total delta size to keep before flushing this client's buffer of deltas. The size is in kilobytes, so 4 means 4 KB. The default is 256 KB, unless you specify the `-d` option, which overrides the default delta size.

cred

Add client credentials to this object. This option is effective only when `server` is used, because it tells PPS to pass the client's PID, UID, and GID to the server by including these fields in the object name.

critical

Designate the publisher as critical to the object. For details, see the "Critical option" section in the *PPS Developer's Guide*.

delta

Open the object in delta mode, which means only the changes made to the object are returned by a read operation.

deltadir

Return the names of all objects listed 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. A value expression consists of an attribute name, followed by an operator, followed by a value.

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

flow=*backlog_size*

Deliver purge notifications for this object (similar to a *server object*). This flag takes an optional argument for the number of kilobytes of backlog (i.e., series of deltas) that the server is permitted. If you don't specify this argument, the `backlog` size is used; if this other option isn't defined, the default size of 256 KB is used.

When the server falls behind in reading the object and the backlog exceeds the size specified in `flow`, the object will be purged and the server will receive purge notifications of the form `|@objname`.

A purge will occur for a client if it doesn't read the reply data at a fast enough rate. In this case, the server will received purge notifications of the form `|@objname.clientid`.

The `flow` flag is effective only when `delta` is used, and is mutually exclusive with `backlog` and `server` (because it enables the server mode internally).

hiwater=*backlog_percentage*

Deliver overflow notifications for this object when the client backlog exceeds a certain limit. This flag takes a mandatory argument in the range of 1 to 99, to indicate the percentage of client backlog at which the server will begin receiving overflow notifications. We refer to this limit as the *high watermark* for overflow.

As long as the backlog remains above this limit, the server will receive a notification of the form `^@objname.clientid` for every write that it performs on the object.

The `hiwater` flag is valid only with the `flow` flag and must be explicitly set to enable overflow notifications. The default `hiwater` value of 100 means the service waits until the client backlog is full before purging the object and hence, no overflow notifications are sent.

`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 updates made on this object back to the process that wrote them. When this option is set, if a process writes data to the object and then reads the object using the same file descriptor, the process will read the data that it wrote. By default, this option isn't set and a *read()* operation won't return the data written with the same file descriptor, because this isn't considered a change.

`server`

Designate the publisher as a *server* for the object (see "*Server Objects* (p. 12)" for details).

`verbose`

Set the verbosity level for this object.

`wait`

Clear the `O_NONBLOCK` flag so that *read()* calls will wait for any object changes, including deltas.

Object format

Objects appear as files 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 afterwards begins 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

Messages written to server objects must have this format:

`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 attribute 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* `\n errstr::` *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/
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

Creating a PPS object is as easy as calling *open()* 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 uses the same mechanisms as performing those operations on other files in the filesystem. As shown in "*Overview of the PPS Service* (p. 11)" in this guide, as long as the data you write conforms to the format that 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*.

Location of PPS objects

When you develop your own apps, any objects that they use must be located at a path under the root PPS directory (**/pps**). Also, you must ensure that the filesystem grants write access to all PPS paths that your apps need to access. Filepath permissions are controlled by the Authorization Manager service, as described in the *System Services Reference*.

On Blackberry devices, some apps use objects accessed from a symbolic link at: **/accounts/1000/appdata/*application_ID*/pps**, where *application_ID* is the app's directory name, found under **/apps** on the target. This design is *not* supported by QNX systems—you must use objects under **/pps**.

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 to 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 new 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 opening 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 create your directory and assign 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. You can do this simply by calling *close()*.

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.

Mutexes

How you'll use mutexes and other synchronization tools is up to you and depends on the needs of your program. 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 update the display. In this case, mutexes ensure that one thread doesn't try to change attributes that the other thread is trying to use. Note that the synchronization needs of your programs may be different.

Chapter 3
PPS Objects Reference Pages

The following pages list the PPS objects found in the QNX Apps and Media image, in alphabetical order.

Each PPS object has its own reference page. The title of each page is the object's filename (e.g., **/pps/services/audio/status**).

/pps/accounts/

Directory that third-party applications use as their sandbox

This directory serves as a sandbox for third-party applications. When an app for a specific vendor is launched for the first time, PPS creates these subdirectories:

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

/pps/qnx/dbnotify/dbs

Object for media database notifications

Publishers

QDB

Subscribers

Any app

Overview

This object is used for database change notifications. For example, when a new song is selected, an artwork synchronization program may wake up and fetch the artwork of the selected song.

Here's a sample object:

```
@dbs
[n]db_mme::1
```

/pps/qnx/demo

The Cordova PPS Demo app uses this object to demonstrate how to use PPS objects in an HTML5 application

Publishers

Cordova PPS Demo

Subscribers

Any app

Overview

The **/pps/qnx/demo** Persistent Publish/Subscribe (PPS) object stores value pairs written by the Cordoba PPS Demo. This app demonstrates how to use the PPS service in an HTML5 application.

For information about how to use the Cordova PPS Demo, see "Cordova PSS Demo" in the QNX SDK for Apps and Media *User's Guide*. For information about how to install and start the Cordova PPS Demo, see "Building and deploying the Cordova PPS Demo" in the *HTML5 Developer's Guide*.

Here's a sample object:

```
@demo
abc::123
```

/pps/qnx/device/*device*

Device publishers write device connectivity details to this object

Publishers

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

For more information about 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 system, PPS objects appear under the **/pps/qnx/device/** directory to expose hardware connectivity details. For USB devices, object names are of the form:

usb-*bus_number*.*device_number*

If **usblauncher** is called with the `-s` option, the object name also includes the *stack_number* attribute before the *bus_number* (e.g., **usb-0.0.3**).

Here's a sample 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
```

For the full list of attributes that can be present in device objects published by **usblauncher**, see the Device Object reference for **usblauncher**.

/pps/qnx/device/*device*_ctrl

Control object for issuing commands to a device

Publishers

Any app

Subscribers

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

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

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

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, such as setting the power state of a USB hub or launching a USB stack version.

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

```
usblauncher -S 1
```

The USB service creates an object named **/pps/qnx/device/usb_ctrl1**.

For details on the commands that apps can send to the control object and on the responses that the object publishes, see the Device Control Object reference for **usblauncher**.

USB control examples

After starting the **usblauncher** process, enter this command from a terminal:

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

Then, from a second terminal, enter these commands:

```
sloginfo -w &
echo toggle_port_power::x,y,z >> /ramdisk/pps/qnx/device/usb_ctrl
```

The first terminal shows the command status and the power result of the command for the specified bus, device, and port:

```
# 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. If an error occurs, this attribute contains a nonzero error code. Details about the possible errors for USB control commands are found in the Device Control Object reference.

/pps/qnx/driver/*pid*

Device publishers write driver details to this object

Publishers

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

For more information about 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 system, PPS objects appear under the **/pps/qnx/driver/** directory to report details of the drivers for those connected devices. The objects are named for the IDs of the driver processes.

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
```

For the full list of attributes that can be present in driver objects published by **usblauncher**, see the Driver Object reference for **usblauncher**.

/pps/qnx/mount/*device*

Device publishers write filesystem and mountpoint information to this object

Publishers

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

For more information about 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 expose filesystem and mountpoint information for connected devices. Object names are of the form:

***rawdevice*[.*partition#*]**

For instance, for a USB stick that's assigned a device path of **/dev/umass0** and that has a DOS partition, these objects are published:

- **/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
```

For the full list of attributes that can be present in mount objects published by **usblauncher**, see the Mount Object reference for **usblauncher**.

/pps/qnx/qdb/config/*dbname*

QDB parses this object to set up a database

Publishers

Any app

Subscribers

QDB

Overview

The **/pps/qnx/qdb/config/** directory contains PPS objects that configure databases. When an object is copied into this directory, QDB parses the object and attempts to load the database with the same name. For example, when an app writes the **/pps/qnx/qdb/config/bluetoothdb** object, QDB attempts to load the **bluetoothdb** database.

Configuration parameters

Each configuration object specifies the database's storage and schema files and its policy settings such as backups and auto-attachment of other databases. The required syntax and the meanings of the supported parameters are given in the "Database configuration objects" section of the *QDB Developer's Guide*.

Parameter	Description
AutoAttach	Specifies other databases to attach to the current one (using the SQL `ATTACH DATABASE` statement).
BackupAttached	Lists attached databases that are to be backed up whenever the main database is.
BackupDir	Specifies the directories for storing database backups. These directories must exist when the database is loaded.
BackupVia	Specifies an interim directory to copy the database to before backing it up. QDB locks the database during the backup, so this setting lets you make an interim copy to reduce the amount of time that the database is locked.
ClientSchemaFile	Names the 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	Specifies the compression algorithm to apply to backups. Options: • `none` (default) • `lzo` • `bzip` • `diocopy` (direct I/O copy uisng DMA)

Parameter	Description
CompressionVia	Used with *BackupVia* and any *Compression* setting specified. The default is `false`. Set this to `true` if you want QDB to compress the database before backing it up.
DataSchemaFile	Names the file (with an absolute path) that contains the SQL commands to populate a database when it's created. This setting is processed only if *SchemaFile* is set.
Filename	Names the database storage (i.e., raw SQLite) file. This must be an absolute path but it can refer to any location. **This setting is required.**
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 for the database.
SizeAttached	Lists attached databases for which size information is to be retrieved whenever it's requested for the main database.
VacuumAttached	Lists attached databases that are to be vacuumed whenever the main database is.

/pps/qnx/qdb/status/*dbname*

QDB publishes the database status to this object

Publishers

QDB

Subscribers

Any app

Overview

For every loaded database, QDB publishes an object in the **/pps/qnx/qdb/status/** directory with the same name as the database. The status object indicates the database state after the loading attempt.

Status values

Each status object contains a *Status* attribute with one of these values:

AttachWait

QDB is waiting for an attached database (listed in the *AutoAttach* parameter) to become available.

Error

The configuration contained an error.

Initializing

QDB has read the configuration object and is now initializing the database.

Valid

The database has been configured and can be accessed.

/pps/services/app-launcher

Control object for launching applications based on name

Publishers

Launcher service; any app

Subscribers

Launcher service; any app

Overview

The launcher service (Application Launcher) provides this control object so clients can issue commands to start and stop applications based on their names. This object allows third-party applications such as speech-recognition programs to launch other applications simply by naming them, without having to read the **/apps** directory to obtain the app ID string. You can also read this object to learn the names of the installed applications.

Command format

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

req:json:{"id": *ID_number*,"cmd":"*command_string*","app":"*app_string*","dat":""}

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

The dat attribute is used for setting parameters that will be sent to the launcher service. Parameters can be either strings or JSON objects.

Starting and stopping applications

At boot time, Application Launcher publishes the names of all existing applications in the app_list attribute:

app_list:json:[*app_string*, *app_string*, ...]

You can launch any application given in app_list by issuing the `launch app` command. For example, to launch the application named "MediaPlayer", issue this command:

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

To stop the MediaPlayer application, send the `close app` command:

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

Responses

Application Launcher responds to each command by writing to the status attribute. This attribute contains the ID number used in the previously issued 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.

Application Launcher creates the ***/pps/system/navigator/command*** (p. 101) object for publishing application tab actions. Applications that subscribe to this object should open it using the `?wait` and `?delta` options, to receive all relevant changes. For information on these options, see "Subscribing" in the *Persistent Publish/Subscribe Developer's Guide*.

/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 object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

Message/response format

Commands sent to the **/pps/services/audio/audio_router_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* `}\nerror::` *error_description*

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 SCO stream is connected.
	volumecontrol	String	Indicates 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	Releases handle returned by `get_handle`.
`get_alias_handle`	*target*	Number	Retrieves a unique handle from the Audio Manager, based on the target handle. Audio on the new handle will be impacted by audio ducking whenever the target handle is impacted.

Commands	Parameters	Data type	Description
	audioman_handle	Number	New handle, an alias to the target handle, that the client should use for all other actions.
`get_handle`	*type*	String	Audio source type, one of: • `alert` • `default` • `inputfeedback` • `multimedia` • `pushtotalk` • `ringtone` • `soundeffect` • `texttospeech` • `videochat` • `voice` • `voicerecognition` • `voicerecording` • `voicetones`
	pid	Number	Process ID of the caller (returned by the *getpid()* call). Note that Audio Manager will get this automatically if it's not filled.
	suspended	Boolean	Indicates whether the audio handle is activated right away. Default is true.
	audioman_handle	Number	New handle, returned by Audio Manager, 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 is muting the given handle.
	muted_by_pid	String	Process ID of the audio source that is muting the given handle. There may be multiple sources muting this handle; if so, this field contains the ID of the first process doing the muting.
`get_type_concurrency_status`	*type*	String	Name of the audio type for which the audio concurrency policy is being returned.
	attenuated	Boolean	Indicates whether the audio type is being attenuated.
	muted	Boolean	Indicates whether the audio type is being muted.
	muted_by	String	Name of the audio source that is muting the given audio type.

Commands	Parameters	Data type	Description
	muted_by_pid	String	Process ID of the audio source that is muting the given audio type. There may be multiple sources muting this type; if so, this field contains the ID of the first process doing the muting.
`get_voice_enhanced_audio_option`	*source*	String	Name of the voice source, one of: • `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, one of: • `normal` • `boost_bass` • `boost_treble`
`get_voice_mode`	*source*	String	Name of the voice source, one of: • `cellular` (default) • `voip`
	mode	String	Voice mode, one of: • `ringer` • `on` • `off`
`numchans`	*number*	Number	Number of channels (1 or 2).
`pcm_input_closed`	*audioman_handle*	Number	Handle returned by `get_handle`. This command notifies apps that a PCM channel for input has been closed/suspended by the **libasound** library.
`pcm_input_opened`	*audioman_handle*	Number	Handle returned by `get_handle`. This command notifies apps that a PCM channel for input has been opened through the **libasound** library. The Audio Manager uses a default type of `generic` for the source.
`pcm_output_closed`	*audioman_handle*	Number	Handle returned by `get_handle`. This command notifies apps that a PCM channel for output has been closed/suspended by the **libasound** library.
`pcm_output_opened`	*audioman_handle*	Number	Handle returned by `get_handle`. This command notifies apps that a PCM channel for output has been opened by the **libasound** library.

Commands	Parameters	Data type	Description
`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	Audio source type, one of: • `alert` • `generic` • `multimedia` • `soundeffect` • `ringtone` • `texttospeech` • `videochat` • `voice` • `voicerecognition` • `voicerecording`
	input	String	Name of input device, overridden by the audio source (see ***/pps/services/audio/devices/*** (p. 49) for the supported devices). The **default** device clears the input setting.
	output	String	Name of output device, overridden by the audio source. A client should set the *input* and *output* parameters only to override the default routing path. For example, if the user has a headset during a phone call, the default routing that the Audio Manager picks is through the headset. However, the user could force the device to send output through the loudspeaker and get input from the handset, in which case the phone app would have to tell **phone-pps** to override the output to `speaker` and the input to `handset`.
`set_voice_enhanced_audio_option`	*source*	String	Name of the voice source, one of: • `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, one of: • `normal` • `boost_bass` • `boost_treble`

Commands	Parameters	Data type	Description
`set_voice_mode`	*source*	String	Name of the voice source, one of: • `cellular` (default) • `voip`
	mode	String	Voice mode, one of: • `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 content 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/*** (p. 49) entry.

/pps/services/audio/control

The Audio Manager listens for commands on this control object

Publishers

Audio Manager; any app

Subscribers

Audio Manager; any app

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

Audio Manager library

Besides reading from and writing to the PPS audio objects directly, you may also use the Audio Manager library, which allows your apps to process events from audio devices. This library provides a C/C++ interface to audio devices accessible through the underlying PPS framework. Using this library, you can get and set audio device properties and 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* `}\nerror::` *error_description*

Commands

Command	Parameters	Data type	Description
`adjust_input_level`	*name*	String	Device name, as listed under ***/pps/services/audio/devices/*** (p. 49).
	level	Double	Percentage representing the adjustment of the input volume level.
`adjust_output_level`	*name*	String	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 when a voice call is activated.

Command	Parameters	Data type	Description
	level	Double	Percentage representing the adjustment of the voice output volume level.
`decrease_output_level`	*name*	String	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	Device name.
	property	String	Name of the device's attribute.
	value	String	Value of the given attribute.
`get_hdmi_info`	*numchans*	Number	Number of audio channels that the connected HDMI device supports.
	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.

Command	Parameters	Data type	Description
	volumecontrol	String	Type of volume control that the attached HDMI device supports: • `unavailable` (no volume control) • `simple` (supports only increase and decrease) • `percentage` (supports full control, including mute, specific steps, etc...)
`get_headphone_enable`	*name*	String	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` • `idle` • `open` • `streaming`
	suspended	Boolean	Indicates whether HFP is suspended as requested by the device. Default is `false`.
	remoteVAD	Boolean	Indicates whether HFP also supports VAD and voice recording.
`get_input_level`	*name*	String	Device name. If an invalid name is given, the input 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	Device name.
`get_output_level`	*name*	String	Device name. The device's *volumecontrol* must be `percentage`.
`get_output_mute`	*name*	String	Device name. The device's *volumecontrol* must be `percentage`.

Command	Parameters	Data type	Description
`get_voice_output_level`	*name*	String	Name of the device to get the output volume level when a voice call is activated.
	level	Double	Percentage representing the current output volume level.
`get_voice_output_mute`	*name*	String	Device name.
	muted	Boolean	Indicates whether the device output is muted.
`increase_output_level`	*name*	String	Device name. The device's *volumecontrol* can be `percentage` or `simple`.
`keep_output_alive`	*name*	String	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` (deprecated)	*audiomode*	String	Tunings to apply to the hardware codec: • `audio` • `video` • `voice` • `record`
`set_hac_enabled`	*enabled*	Boolean	Indicates whether Hearing Aid Compatibility (HAC) 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 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...)

Command	Parameters	Data type	Description
			The hdmi audio driver sends HDMI information to the Audio Manager before the HDMI handler calls `set_hdmi_enable`, to prevent this function from publishing inaccurate information.
`set_hdmi_mode`	*mirror*	Boolean	Indicates whether mirror mode is enabled.
`set_headphone_enable` (may be deprecated)	*name*	String	Name of the device to be activated as the default when this event occurs.
	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, a volume level greater than 92% is allowed.
`set_input_level`	*name*	String	Device name. If an invalid name is given, the input 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	Device name.
	muted	Boolean	Indicates whether the given device input is muted.
`set_output_level`	*name*	String	Device name. The device's *volumecontrol* must be `percentage`.
	level	Double	Percentage representing the desired output volume level.
`set_output_mute`	*name*	String	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 if so, TTY is enabled during a voice call.
`set_voice_output_level`	*name*	String	Name of the device to set the voice output volume level when a voice call is activated.
	level	Double	Percentage representing the desired output volume level.
`set_voice_output_mute`	*name*	String	Device to mute/unmute.
	muted	Boolean	Indicates whether the device output is muted.
`toggle_input_mute`	*name*	String	Device name. If an invalid name is given, the input level of the currently selected input device is updated. If the given device is output

Command	Parameters	Data type	Description
			only, the corresponding input device when the output device is selected is updated.
`toggle_output_mute`	*name*	String	Device name. The device's *volumecontrol* must be `percentage`.
`toggle_voice_output_mute`	*name*	String	Name of the device to toggle the mute status when a voice call is activated.

Examples

To set the volume level of the headset to 75%:

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

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

To mute the speaker:

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

/pps/services/audio/devices/

Directory for listing supported audio devices

Publishers

Audio Manager

Subscribers

Any app

Each audio device is represented by one PPS object, which is named after the device.

Supported devices

Device name	Description
a2dp	Bluetooth A2DP connection. When selected, the default input (onboard mic) is selected.
audioshare	Virtual audio device accessible through video share.
btsco	Bluetooth SCO/HFP connection.
hac	HAC telecoil used for hearing aids. When selected, the default input (onboard mic) is selected.
handset	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	Output device connected through the headset jack. When selected, the default input (onboard mic) is selected.
miracast	Audio device available from a Wi-Fi display connection. Miracast is the protocol used for this connection.
mirrorlink	Audio device available from a MirrorLink connection. MirrorLink is a protocol for integrating mobile devices with embedded systems.
speaker	Main speaker on the device (handsfree on mobile phones).
tones	Virtual audio device for the system tones.
toslink	Optical audio device (“Toshiba Link”) used by some receivers.

Device name	Description
tty	Telecommunications device for the deaf (connected through the headphone jack).
usb	USB audio device.
voice	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	Current default audio device for capture (e.g., `handset`).
input.path	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 do noise cancellation).
connected	Boolean	Indicates whether a given device is connected.
controlpps	String	Path of the device's PPS control object. If this object exists, then the device is controlled by a PPS interface instead of an audio driver.
dependency	Boolean	Indicates whether this device depends on another device (if so, 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). Note that if *inchans* is 0, then no input is supported for this device.
input.device	String	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: • `BT A2DP Out` • `BT SCO Out` • `HDMI Out` • `Master` • `PCM Mixer` • `Tones Out` • `TosLink Out`

Attribute	Data type	Description
		• `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	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/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	Indicates whether the headphone output volume is limited by regulations (e.g., the setting may be made 100% by an app, but the regulation limits the volume to 90%). This field is `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	Unsafe volume level for headphones.
hpunsafezone	Boolean	Unsafe volume range for headphones.
hpunsafezone.supported	Boolean	Indicates whether the unsafe volume range is supported.
input.*device*.gain	Number	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	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	Output gain (in percent) for this device.

/pps/services/audio/types/

Directory for listing supported audio types

Publishers

Audio Manager

Subscribers

Any app

Each audio type is represented by one PPS object, which is named after the type.

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 can take certain actions when an event occurs. For example, a multimedia application may decide to pause when it's being muted by a higher-priority audio source.

Supported audio types

Audio type	Description
alarm	Used for an alarm, such as a clock alarm.
alert	Notifiers, such as calendar events, email, and SMS.
cmas	Used for Commercial Mobile Alert System (CMAS) emergency broadcast systems.
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.
reserved_0	Placeholder value; unused.
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 Voice-Activated Dialing (VAD).
voicerecording	Used for voice-recording services.
voicetones	Dual-tone multi-frequency signaling (DTMF) and call-progress tones, but can also be used to play back non-tone-based audio during phone calls.

Audio type attributes

Each audio type object has the following attributes:

Attribute	Data type	Description
active_pid (deprecated)	String	Process ID of the currently playing application. Note that this is used only with the multimedia type.
attenuated	Boolean	Indicates whether the audio type is being attenuated.
muted	Boolean	Indicates whether the audio type is being muted.
muted_by_pid	String	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 content 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.audioshare.muted:b:false
voice.output.audioshare.volume:n:100.000000
voice.output.btsco.muted:b:false
voice.output.btsco.volume:n:100.000000
voice.output.hac.muted:b:false
voice.output.hac.volume:n:0.000000
voice.output.handset.muted:b:false
voice.output.handset.volume:n:0.000000
voice.output.hdmi.muted:b:false
voice.output.hdmi.volume:n:0.000000
voice.output.headphone.muted:b:false
voice.output.headphone.volume:n:0.000000
voice.output.headset.muted:b:false
voice.output.headset.volume:n:0.000000
voice.output.lineout.muted:b:false
voice.output.lineout.volume:n:0.000000
voice.output.miracast.muted:b:false
voice.output.miracast.volume:n:0.000000
voice.output.mirrorlink.muted:b:false
voice.output.mirrorlink.volume:n:0.000000
voice.output.speaker.muted:b:false
voice.output.speaker.volume:n:0.000000
voice.output.tones.muted:b:false
voice.output.tones.volume:n:100.000000
voice.output.toslink.muted:b:false
voice.output.toslink.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:0.000000
```

For more information on the audio devices, see the ***/pps/services/audio/devices/*** (p. 49) entry.

/pps/services/geolocation/control

The Geolocation service listens for commands on this object

Publishers

Geolocation service; any app

Subscribers

Geolocation service; any app

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

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 these commands:

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

Because the Geolocation service is multithreaded, 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 for the browser.
location_manager_location_gnss_on	Indicates whether the Global Navigation Satellite System is on.

/pps/services/launcher/control

Control object for launching applications based on ID

Publishers

Launcher service; Homescreen

Subscribers

Launcher service; Homescreen

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

Overview

The launcher service (Application Launcher) provides this control object so clients can issue commands to start and stop applications based on their IDs. In the QNX Apps and Media image, the Homescreen uses this object to launch applications. If you write your own window manager, it can use this object.

Message/response format

Commands sent to the **/pps/services/launcher/control** object have the following form:

`msg::` *command_string* `\ndat::` *application_ID* `[,` *display_parameters* `]\nid::` *ID_number*

Here, *command_string* contains the name of the command and *application_ID* contains the name of the app when it's loaded. The app names are found in the **/apps** directory. The *display_parameters* string is used only with the `start` command for launching applications; details on this command and other supported commands are given in "*Commands* (p. 62)". The *ID_number* can be a number or any other identifier meaningful to the client.

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

`res::` *command_string* `\ndat::` *process_ID* `,` *process_name* `\nid::` *ID_number*

The launcher service stores the process ID (PID) and name in the dat attribute. The PID must be saved by the client so it can later refer to the same application. For instance, when the user selects to stop the app in the HMI, the client must pass the PID in the `stop` command.

Because this is a server object, to observe its status, you must force the shell to keep the file descriptor open. In the following example, we use the `exec` command to do this, then direct the response messages to **stderr**. To start apps through PPS in the QNX Apps and Media image, you must slay the HMI and issue commands to the launcher control object. In this case, we start an application called Peaks And Valleys. The service writes the PID of the new process into the dat attribute in its response. You can use this PID to pause or stop the app.

```
# slay -13 homescreen
# exec 3<> /pps/services/launcher/control
```

```
# echo "msg::start\ndat::PeaksAndValleys.testDev_sAndValleys6bb91d91,
        ORIENTATION=0,WIDTH=900,HEIGHT=480\nid::1234" >&3
# cat <&3
@control
res::start
id::1234
dat::1589296,dev_mode
# echo "msg::stop\ndat::1589296\nid::1234" >&3; cat <&3
@control
res::stop
id::1234
```

Commands

The control object accepts these commands:

start

Launches an application based on the application ID.

stop

Stops an application from running.

freeze

Pauses an application. The application appears frozen on the screen.

thaw

Unpauses an application.

The format for these commands is as follows:

msg::	dat::	id::
`start`	ID of the application to launch, followed by a list of display parameters. The ID string is based on the app's directory name found in **/apps**. For the display parameters, you can list the orientation, height, and width of the app display area. Individual parameters are separated by commas and each one is specified by listing its name, followed by an equal sign (=) and the value. You can set these parameters: • ORIENTATION — 0 for landscape mode, 1 for portrait mode • WIDTH — Width of the app display area, in pixels • HEIGHT — Height of the app display area, in pixels We strongly recommend setting the WIDTH and HEIGHT parameters. Although the launcher service still launches the app (using the full screen to display it) if these parameters are not provided, the full-screen display might not be optimal for viewing the app.	Number (or any other identifier meaningful to the clients involved)

msg::	dat::	id::
	In the reference Apps and Media images, the Homescreen application sets these parameters in the `start` command, which it sends when the user launches an app in the HMI.	
`stop`	PID of the application to close. This must be the PID returned in the dat attribute of the `start` response.	Number (or any other identifier meaningful to the clients involved)
`freeze`	PID of the application to pause.	Number (or any other identifier meaningful to the clients involved)
`thaw`	PID of the paused application to resume running.	Number (or any other identifier meaningful to the clients involved)

Examples

To launch the Settings application, write the following to the **/pps/services/launcher/control** object:

```
echo "msg::start\ndat::Settings.testRel_Settings___595d2043,
        ORIENTATION=0,WIDTH=900,HEIGHT=480\nid::1"
        > /pps/services/launcher/control
```

The service then writes this type of response (containing the PID) to the object:

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

You must pass the PID in the `stop` command to close this application:

```
echo "msg::stop\ndat::2015282\nid::1" > /pps/services/launcher/control
```

If you don't specify the PID, Application Launcher stops all active applications when you issue the following `stop` command:

```
echo "msg::stop\ndat::\nid::" > /pps/services/launcher/control
```

In the default QNX Apps and Media image, only one application can be active at a time. If you modify the HMI to allow for multiple applications to be active at the same time but want to affect only one application, you must specify the PID when sending the `stop`, `freeze`, or `thaw` command.

/pps/services/multimedia/mtp/

Directory that MTP driver processes use for publishing objects containing device information

Publishers

MTP driver processes

Subscribers

Any app

Overview

The **/pps/services/multimedia/mtp/** directory stores objects containing information about attached MTP devices. Objects summarizing different devices are stored in different subdirectories. Each subdirectory has a name in this format:

PID– Busno Devno

The name consists of the PID of the MTP driver started when the device was attached, followed by the bus number and device number. With each device attachment, the system starts another MTP driver process; if 5 MTP devices are attached, 5 drivers are launched. Within any of these driver subdirectories, you can find the ***devinfo*** (p. 65) object (which contains device-identifying and system software details) and a *subdirectory with storage objects* (p. 67) (which store the last modification time and other details for the storages on the device).

/pps/services/multimedia/mtp/*driverdir*/devinfo

Object for storing vendor and software information about an MTP device

Publishers

MTP driver process

Subscribers

Any app

Overview

The **/pps/services/multimedia/mtp/*driverdir*/devinfo** object stores fields that identify the manufacturer, vendor, and model name of an attached MTP device and that describe its system software. The *driverdir* directory is created by the MTP driver launched when the device is attached. Here's a sample object:

```
[n]@devinfo
Device::PlaysForSure
FunctionMode::0
Manufacturer::samsung
Name::Galaxy Nexus
SerialNumber::0149BDCB1800E01F
SoftwareVersion::1.0
StandardVersion::100
VendorExtDesc::microsoft.com: 1.0; android.com: 1.0;
VendorExtId::0x6
VendorExtVersion::100
```

Attributes

Attribute	Data type	Description
Device	String	Device type (always `PlaysForSure`)
FunctionMode	Hexadecimal string	Function mode supported by the device. Modes express different states and capabilities. A value of 0 means the device supports only Standard Mode. For details on the meanings of different field values, see section 5.1.1.5 of revision 1.1 of the *Media Transfer Protocol* specification.
Manufacturer	String	Device manufacturer
Name	String	Device model
SerialNumber	String	Serial number of the MTP function. This number is unique among MTP functions with the same Name and SoftwareVersion values.
SoftwareVersion	String	Software or firmware version of the device, in a vendor-specific format

Attribute	Data type	Description
StandardVersion	Number	Protocol version supported by the device (always 100, which represents version 1.00)
VendorExtDesc	String	List of vendor extensions supported by the device. Extensions let vendors define additional capabilities for MTP devices.
VendorExtId	Hexadecimal string	ID of vendor of extensions supported by the device. This ID is assigned by I3A.
VendorExtVersion	Number	Version of vendor-extension set supported by the device

/pps/services/multimedia/mtp/*driverdir*/storages/

Directory used by an MTP driver process to publish storage-related objects

Publishers

MTP driver process

Subscribers

Any app

Overview

The **/pps/services/multimedia/mtp/*driverdir*/storages/** directory contains objects that represent the presistent storage mediums of the device. For example, suppose the user attaches a smartphone that has an SD card inserted. The MTP driver that manages communication with that smartphone creates the *driverdir***/storages** directory and publishes two objects in **storages**—one for the phone's internal storage and the other for the SD card. The name of each object is the ID of the storage that it describes. The content of a storage object looks like this:

```
[n]@00010001
FileSystemType::0x2
LastKnownStorageModificationTime::1136011133
MaxCapacity::12724617216
StorageDesc::Phone
StorageID::00010001
StorageLabel::SECZ9519043CHOHB
StorageType::0x3
```

Attributes

Attribute	Data type	Description
FileSystemType	Hexadecimal string	Filesystem used by the storage. For details on the meanings of different field values, see section 5.2.2.2 of revision 1.1 of the *Media Transfer Protocol* specification.
LastKnownStorage ModificationTime	Number	Timestamp indicating when the storage was last modified
MaxCapacity	Number	Maximum capacity (in bytes) of the storage
StorageDesc	String	Description of the storage (e.g., `256Mb SD Card`)
StorageID	String	Storage ID, as a string. The first four digits identify the storage's physical location while the last four identify its logical partition.
StorageLabel	String	Volume label. This field is present only if the filesystem on the storage is mounted.
StorageType	Hexadecimal string	Type (i.e., physical nature) of the storage. For details on the meanings of different field values, see section 5.2.2.1 of the *Media Transfer Protocol* specification.

/pps/services/multimedia/renderer/component/

Directory that the multimedia renderer uses for publishing component (plugin) objects

Publishers

mm-renderer

Subscribers

Any app

The `.all` object

The **.all** object lists all dynamically loaded plugins, the library files implementing them, and the file types supported for playback or recording. Here is a sample object:

```
[n]@mmr-playlist-engine
dll::mmr-playlist-engine.so
plugin::Playlist engine plugin
[n]@mmr-track-engine
dll::mmr-track-engine.so
plugin::Single-track engine plugin
[n]@mmr-mmf-routing
dll::mmr-mmf-routing.so
mime::video/mp4,video/m4v,audio/m4a,audio/aac,audio/wav,audio/mp1,
audio/mp2,audio/mp3,audio/x-mp3,audio/mpeg,audio/x-mpeg,audio/mpg,
audio/x-mpg,audio/mpeg3,audio/x-mpeg3,audio/x-mpegaudio,video/avi,
audio/avi,audio/wma,video/x-mp4,video/x-m4v,audio/x-m4a,audio/3gpp
,video/3gpp,video/3gpp2,audio/3gpp2
plugin::QNX MMF routing plugin
[n]@mmr-audiomgmt-plugin
dll::mmr-audiomgmt-plugin.so
plugin::QNX Audio Management plugin
```

Each section begins with a `[n]@` *plugin-name* line and describes one loaded plugin. The plugin library file and a human-readable description are always provided (in the dll and plugin attributes). Other attributes may list the MIME types playable by the plugin or the filename extensions supported for audio recording (depending on the plugin). Also, any default settings for a plugin, as defined in the **mm-renderer** configuration file, appear in its section of the **.all** object.

Attributes

The **.all** object contains at least these attributes:

Attribute	Description
audioencodeextensions	Comma-separated list of supported extensions for file outputs (e.g., `m4a,wav`).
dll	Library file implementing the plugin.

Attribute	Description
mime	Comma-separated list of allowed combinations of playable MIME types (e.g., `3gpp,video`).
plugin	Description of the plugin.

Plugin objects in the `component` directory

The **component** directory contains objects that describe the dynamically loaded plugins of **mm-renderer** (**mm-renderer** has defined its own plugin interface for modularization and extensibility). Depending on your system, you will see some or all of the following objects:

mmr-track-engine

Engine plugin for handling track inputs

mmr-playlist-engine

Engine plugin for handling playlist inputs

mmr-mmf-routing

Routing plugin for playing individual tracks

mmr-mmfrip-routing

Routing plugin for ripping tracks

mmr-audiomgmt-plugin

Plugin for communicating with the **audioman** service, which controls audio routing

Each of these objects contains at least the dll and plugin attributes and may contain others, as explained previously.

/pps/services/multimedia/renderer/context/*contextname/*

Directory that the multimedia renderer uses for publishing objects for a context

Publishers

mm-renderer

Subscribers

Any app

Overview

Whenever a client calls *mmr_context_create()*, the **mm-renderer** service 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:

- ***$mmr_ppsdir/ contextname/*param**
- ***$mmr_ppsdir/ contextname/*output#**
- ***$mmr_ppsdir/ contextname/*input**
- ***$mmr_ppsdir/ contextname/*metadata** (created when an input is attached to the context)
- ***$mmr_ppsdir/ contextname/*p#** (if the input is a playlist, a **p#** object is created for each playlist entry)
- ***$mmr_ppsdir/ contextname/*play-queue** (created if the input is a playlist)
- ***$mmr_ppsdir/ contextname/*q#** (if the input is a playlist, a **q#** object is created for each playlist entry)
- ***$mmr_ppsdir/ contextname/*state**
- ***$mmr_ppsdir/ 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

The contents of this object depend on what the client specified when calling *mmr_input_parameters()* for this context. The attributes that the **input** object may contain are:

Attribute	Description
url	URL of the attached input.
type	Values: • `track` (a single track) • `playlist` (a track sequence) • `autolist` (a single track formatted as a playlist)
repeat	How to replay the input (for playlist and autolist input types only). Possible values: • `"none"` (default) • `"track"` • `"all"`

/pps/services/multimedia/renderer/context/*contextname*/metadata

Contains metadata for the input attached to the specified 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
```

Attributes

The attributes published to this object depend on the media file type of the input.

/pps/services/multimedia/renderer/context/*contextname*/output#

Holds parameters for an output attached to 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#

Holds the input URL and parameters for an individual track

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

Publishers

mm-renderer

Subscribers

Any app

Overview

When the input is a playlist, a **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).

Attributes

The contents of this object depend on what the client specified when calling *mmr_track_parameters()* for this track. The attributes that this object may contain are the same as those that can be found in the **input** object.

/pps/services/multimedia/renderer/context/*contextname*/param

Contains the parameters defined for the specified context

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

Publishers

mm-renderer

Subscribers

Any app

Attributes

The contents of this object depend on what the client specified when calling *mmr_context_parameters()* for this context. The attributes that the **param** object may contain are:

Attribute	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` • `OPT_SSL_VERIFYHOST` • `OPT_PROXY` • `OPT_NOPROXY`	

Attribute	Description
• 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 C Library Reference*.

/pps/services/multimedia/renderer/context/*contextname*/play-queue

Holds information about 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

Attributes

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#

Contains metadata for a track within a playlist

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

Publishers

mm-renderer

Subscribers

Any app

Overview

When the input is a playlist, a **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).

Attributes

The contents of this object depend on the media file type of the track.

/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 details 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 play 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 changed:
No input is attached	`idle`	none
An input is attached	`stopped` (from `idle`)	input is set to input's URL

Condition:	The state attribute is set to:	Other attributes changed:
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 by a function call)
Warning occurs (note that warnings don't stop playback)	`playing`	warning and warning_pos are set
Error occurs (note that errors do 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

CAUTION:

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

For details 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 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)
`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)

Command	Parameters
	• *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)

Example

Suppose you want to attach an input to a context and play the track **/macqnx/RCPS_SuckerPunchTH_M2_TestFile.mpg**. You would write this command to the **control** object:

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

/pps/services/networking/all/interfaces/

Directory for storing status objects for network interfaces

Publishers

Network Manager (**net_pps**)

Subscribers

Any app

Overview

The **/pps/services/networking/all/interfaces/** directory contains status objects for all network interfaces either listed in the **net_pps** command line or set up through the configuration files in **/var/etc/system/config/network/**. Each object has the same name as the interface as reported by the `ifconfig` utility (e.g., **en0**, **fec0**).

The objects that can be found in this directory vary with the target hardware. However, the **lo0** object, which represents the loopback interface, may be present on any target.

Attributes

Attribute	Data type	Description
connected	Boolean	Indicates whether the interface is currently connected.
fib	Number	FIB number.
httpproxy	String	IP address of the HTTP proxy server.
httpproxyloginrequired	Boolean	Indicates whether the HTTP proxy requires login credentials.
ip4_ok	String	A `yes` here indicates IPv4 connectivity is available. Otherwise, one of these error strings appears: • `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).
ip_gateway	JSON	Array of IP gateways.

Attribute	Data type	Description
ip_ok	String	General status attribute for IP connectivity.
link_address	String	Link layer (MAC) address.
manual	String	Indicates whether manual IPv4 settings 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	When `on`, indicates that IPv6 manual settings will be used. When `off`, it means that 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: • `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 the physical interface is up.

/pps/services/networking/all/proxy

Status object for proxy information

Publishers

Network Manager

Subscribers

Any app

Overview

Network Manager publishes the locations of proxy servers to this object.

All of this information (except for httpproxylogin) is also published to the */pps/services/networking/all/status_public* (p. 87) object.

Attributes

Attribute	Data type	Description
ftpproxy	String	FTP proxy of the connected network.
ftpproxy6	String	IPv6 FTP 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 the HTTP proxy requires login credentials.
httpsproxy	String	HTTPS proxy of the connected network.
httpsproxy6	String	IPv6 HTTPS proxy of the connected network.

/pps/services/networking/all/status_public

Status object for the currently preferred network interface

Publishers

Network Manager

Subscribers

Any app

Overview

This object contains status information for the *currently preferred* network interface (i.e., the currently active interface when running in station mode). You can obtain more information about the interface by reading its object in the */pps/services/networking/all/interfaces/* (p. 84) directory.

Attributes

Attribute	Data type	Description
cmd_output	String	Output from the last executed command (which was written in the msg:: attribute in the networking control object).
default_gateway	JSON	Gateway address.
default_interface	String	Network interface name from `ifconfig` (e.g., **en0**, **fec0**).
default_interface4	String	Name of the network interface currently used to route IPv4 traffic. The active (or current) interface is the connected interface listed earliest in the arguments passed to **net_pps**. For instance, this command: `net_pps en0 tiw_sta0` means that when **en0** is connected, it will be the default interface because it has priority over **tiw_sta0** based on its earlier listing in the command line.
default_interface6	String	Name of the network interface currently used to route IPv6 traffic. The active (or current) interface is the connected interface listed earliest in the arguments passed to **net_pps**, as for default_interface4.
fib	Number	FIB number.
ftpproxy	String	FTP proxy of the connected network.
ftpproxy6	String	IPv6 FTP 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 the HTTP proxy requires login credentials.
httpsproxy	String	HTTPS proxy of the connected network.

Attribute	Data type	Description
httpsproxy6	String	IPv6 HTTPS proxy of the connected network.
ip4_ok	String	A `yes` here indicates whether IPv4 connectivity is available. Otherwise, one of these error strings appears: • `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_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.

/pps/services/networking/control

The Network Manager service listens for commands on this control object

Publishers

Network Manager; any app

Subscribers

Network Manager; any app

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

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 these commands:

`net_connected`

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

`net_disconnected`

Informs Network Manager that the specified interface was disconnected.

`net_disconnecting`

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

Network Manager publishes a notice of the impending shutdown to the appropriate ***/pps/services/networking/all/interfaces/interface*** (p. 84) object.

`net_dyn`

Supplies Network Manager with dynamic configuration data. The response will contain the err attribute on error and will be empty on success.

The following table shows the command format:

msg::	id::	dat:json:
net_connected	Number	["*interface*"{"*parameter*":"*value*", ...}]
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=
	• 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)

Parameter	Description
	• `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 attribute. The reply will contain the err attribute on error and will be empty on success.

For example, a client can write:

```
msg::cmd
id::5
dat::ping -n -c4 10.42.116.1
```

/pps/services/networking/interfaces/

Directory for storing network interfaces

Publishers

Network Manager (`net_pps`)

Subscribers

Any app

Overview

The **/pps/services/networking/interfaces/** directory contains objects that represent all the available network interfaces on your target hardware. The objects that are found in this directory vary with the target hardware, but you should see the **lo0** object, which represents the loopback interface that should be present on every target. You should see the same list of interfaces reported when you run the `ifconfig` utility (e.g., **en0**, **fec0**).

The objects in the **/pps/services/networking/interfaces/** directory differ from the interfaces that are managed by `net_pps`, which are tracked in the **/pps/services/networking/all/interfaces/** directory. This directory may have objects for interfaces that aren't managed by `net_pps`. Changes to the interface are mirrored in interface objects in both the **/pps/services/networking/all/interfaces/** and **/pps/services/networking/interfaces/** directories.

Attributes

This object has the same attributes as the **/pps/services/networking/all/interfaces/** object. For more information about the attributes for this object, see "Attributes" in */pps/services/networking/all/interfaces/* (p. 84).

/pps/services/networking/proxy

Duplicate of **/pps/services/networking/all/proxy**

This is a duplicate of the ***/pps/services/networking/all/proxy*** (p. 86) 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* (p. 87) object.

/pps/system/info

The Settings app reads system software information from this object

Publishers

Image generation utilities

Subscribers

Settings app

Overview

When the system image is built, the image generation utilities output the **/base/etc/os.version** file, which stores build and version information. The **mksysimage.py** script writes this information to an object in the PPS persistence directory (**/var/pps/system/info**), so that the information is available when the system boots for the first time.

When PPS is started, it creates the **/pps/system/info** object and copies the build and version information from persistent storage into the latter object. The Settings app then reads this object so it can display the system software details to the user. Here's a sample object:

```
@info
Branch::anm1.1
Build_Number::101
Build_URL::http://pandapal-lab/job/appsmedia_image_anm11_linux_
omap5/101/
Generation_Date::Thu Oct  2 12:59:29 2014
Image_Script::Unknown
Platform::omap5uevm.ext
Product::AnM
SDP_Revision::5456
Variant::omap5uevmdemo
```

Attributes

Attribute	Description
Branch	Branch of the build (based on software release).
Build_Number	Build number.
Build_URL	Location of the build package.
Generation_Date	Timestamp of when the software package was built.
Image_Script	Script used to build the system image.
Platform	Target platform for the system software.
Product	Short name of the product.

Attribute	Description
SDP_Revision	Revision number of the underlying QNX SDP installation.
Variant	Hardware board variant supported by the system software.

/pps/system/keyboard/control

The Keyboard service listens for commands from the HMI on this control object

Publishers

Any app

Subscribers

Keyboard service

This object is a server object, designed to process requests from individual clients and deliver the results to the clients that issued the requests. For more information, see the "*Server objects* (p. 12)" subsection.

Message/response format

Commands sent to the **/pps/system/keyboard/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* `}\nerror::` *error_description*

Commands

msg::	id::	dat:json:
Message to send to the control object, either `show` or `hide`.	The message's ID string (usually a number, but can be anything).	JSON data (payload) related to the message. The current commands don't require extra data, so this field should be left empty, as shown in the sample commands that follow.

Examples

To show the keyboard:

```
echo "msg::show\nid::1\ndat:json:{}" > /pps/system/keyboard/control
```

To 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 status

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 the keyboard is visible. The Keyboard service sets this attribute after receiving a `show` or `hide` command from the ***/pps/system/keyboard/control*** (p. 97) object.

/pps/system/navigator/applications/applications

The Applications Window Manager publishes information about installed apps to this object

Publishers

Applications Window Manager

Subscribers

Any app

Overview

Each app installed on the system appears in the **/pps/system/navigator/applications/applications** object:

```
@applications
AudioDemo.testDev_AudioDemo__82a252b9::native/icon.png,AudioDemo,media,,auto,,
BrowserLite.testDev_BrowserLite353323d6::native/icon.png,Browser Lite,media,,auto,,
CordovaPPSdemo.testDev_dovaPPSdemod339185a::native/default-icon.png,Cordova PPS Demo,,,auto,,
PeaksAndValleys.testDev_sAndValleys6bb91d91::native/icon.png,PeaksAndValleys,games,,auto,,
Settings.testRel_Settings___595d2043::native/icon.png,Settings,vehicle,,auto,,
Shutdown.testDev_Shutdown___f9c26a76::native/icon.png,Shutdown,vehicle,,auto,,
VideoDemo.testDev_VideoDemo__c6ddf0de::native/icon.png,VideoDemo,media,,auto,,
com.example.ipcamera.testRel_le_ipcamerad35b63dd::native/icon.png,IP Camera,,,auto,,
com.example.mediaplayer.testRel_mediaplayer178ac554::native/icon.png,MediaPlayer,,,auto,,
com.example.photoviewer.testRel_photoviewer775eb853::native/icon.png,PhotoViewer,,,auto,,
com.qnx.com.testDev_com_qnx_com67e92ba1::native/res/icon/blackberry10/icon-80.png,com.qnx.com,,
{768x1280}native/res/screen/blackberry10/splash-1280x768.png:
{720x720}native/res/screen/blackberry10/splash-720x720.png:
{1280x768}native/res/screen/blackberry10/splash-768x1280.png,auto,,
helloworld.testDev_helloworld_a520b600::native/default-icon.png,HelloWorld,,,auto,,
rearview_camera.testDev_view_camerad91629db::native/resources/icon.png,Camera,vehicle,,auto,,
sys.browser.gYABgJYFHAzbeFMPCCpYWBtHAm0::native/app_icon_browser.png,Browser,internet,
{768x1280}native/splash.png:{1280x768}native/splash_portrait.png:
{720x720}native/splash_720x720.png,auto,,
```

In this example, the information for some apps is shown over several lines but in the actual PPS object, each app has all its information on one line.

The app information includes:

- the installation directory under **/apps**
- the location of the app's icon, relative to the installation directory
- the app's name
- the app's category (`media`, `games`, etc...)
- the relative paths of the splash screens; multiple entries must be separated by colons (`:`) with each screen entry prefaced with its resolution (e.g., `{1280x768}native/splash_portrait.png:`

`{720x720}native/splash_720x720.png`)

- the app's orientation (e.g., `auto`, `landscape`)

/pps/system/navigator/command

Shows application display actions

Publishers

Applications Window Manager

Subscribers

Any app

Overview

The **/pps/system/navigator/command** object shows the current display actions of HMI apps. Here's a sample object:

```
@command
rearview_camera:json:{"action":"pause"}
```

In the shipped images, the Camera app is the only app for which the display action is published to this object.

Each line shows an application name, followed by the `json` data type, followed by the `"action":"`*value*`"` string pair. The values for the `"action"` field can be:

- `pause`—the app is being told it's in the background, so it should stop CPU-intensive display tasks (e.g., drawing album cover flows)
- `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 was paused (e.g., start drawing cover flows again)

/pps/system/navigator/windowgroup

Stores identifiers of window groups used by HMI apps

Publishers

HMI apps

Subscribers

Applications Window Manager

Overview

The **/pps/system/navigator/windowgroup** object shows the window groups for HMI apps. Here's a sample object:

```
@windowgroup
[n]rearview_camera::{4c1e7a50-e3de-4048-850f-b06bc6bbe965}
```

In the shipped images, only the Camera app publishes its window group ID to this object.

Chapter 4
List of Objects Used Internally

For this release of the QNX SDK for Apps and Media, 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/applications/weathernetwork/***
- **/pps/servicedata/schedule**
- **/pps/services/apkruntime/**
- **/pps/services/audio/private/**
- **/pps/services/audio/stats**
- **/pps/services/authentication/**
- **/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/geolocation/settings**
- **/pps/services/input/context/** ***contextname***
- **/pps/services/input/control**
- **/pps/services/mediaserver/settings**
- **/pps/services/mesa**
- **/pps/services/mm-player/**
- **/pps/services/multimedia/mediacontroller/notifications**
- **/pps/services/multimedia/mediaplayer/***
- **/pps/services/multimedia/sound/**
- **/pps/services/multimedia/sync/**
- **/pps/services/network-time/status**
- **/pps/services/networking/all/status**
- **/pps/services/networking/status**
- **/pps/services/notification/**
- **/pps/services/notify/***

- **/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/language**
- **/pps/system/launcher_priority**
- **/pps/system/navigator/proc/***
- **/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

A

C

D

Q

S

T

U