HAL 0.5.14 Specification

David Zeuthen

<david@fubar.dk>

Version 0.5.14

Table of Contents

Chapter 1. Introduction

Table of Contents

About

This document concerns the specification of HAL which is a piece of software that provides a view of the various hardware attached to a system. In addition to this, HAL keeps detailed metadata for each piece of hardware and provide hooks such that system- and desktop-level software can react to changes in the hardware configuration in order to maintain system policy.

HAL represents a piece of hardware as a *device object*. A device object is identified by a unique identifer and carries a set of key/value paris referred to as *device properties*. Some properties are derived from the actual hardware, some are merged from *device information files* and some are related to the actual device configuration. This document specifies the set of device properties and gives them well-defined meaning. This enable system and desktop level components to distinguish between the different device objects and discover and configure devices based on these properties.

HAL provides an easy-to-use API through D-Bus which is an IPC framework that, among other things, provides a system-wide message-bus that allows applications to talk to one another. Specifically, D-Bus provides asynchronous notification such that HAL can notify other peers on the message-bus when devices are added and removed as well as when properties on a device are changing.

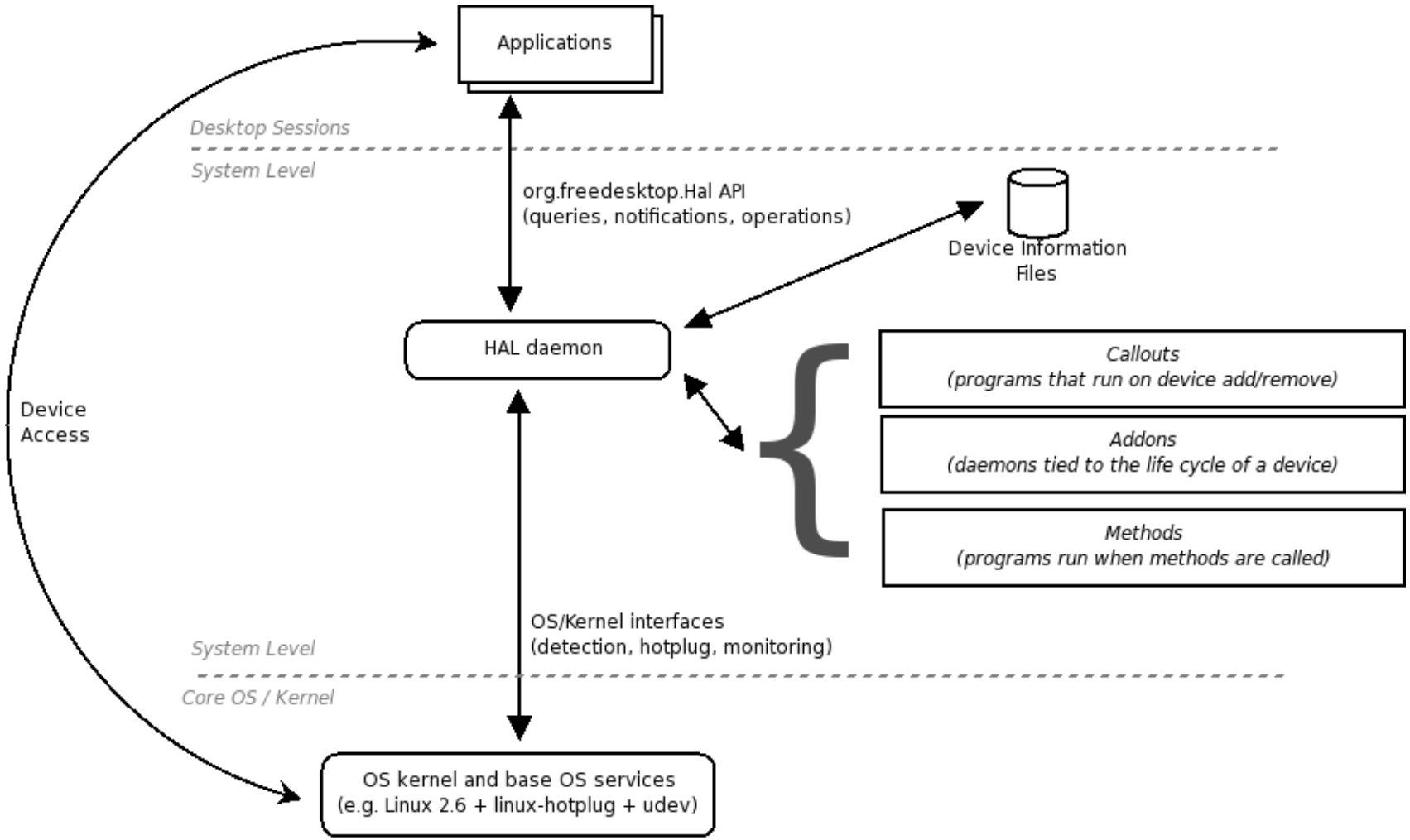
The most important goal of HAL is to provide plug-and-play facilities for UNIX-like desktops with focus on providing a rich and extensible description of device characteristics and features. HAL has no other major dependencies apart from D-Bus which, given sufficient infrastructure, allows it to be implemented on many UNIX-like systems. The major focus, initially, is systems running the Linux 2.6 series kernels.

Acknowledgements

Havoc Pennington's article ''Making Hardware Just Work'' motivated this work. The specification and software would not exist without all the useful ideas, suggestions, comments and patches from the Free Desktop and HAL mailing lists.

Architecture of HAL

The HAL consists of a number of components as outlined in the diagram below. Note that this diagram is high-level and doesn't capture all implementation details.



Details on each component

- *HAL daemon*

  A system-wide service that maintains a database of device objects. The daemon is responsible for merging information from device information files and managing the life cycle of device objects. The service is implemented as a daemon and uses helpers to query devices for specific information.

- *Applications*

These are applications consuming services from HAL; this includes desktop-wide session daemons for maintaining policy such as power and disk/volume management.

- *Callouts*

  Callouts are programs that run when device objects are added and removed in the HAL daemon. This is useful for 3rd party software to merge additional information onto the device object before it is announced on D-Bus. Callouts are specified on a per-device basis with the `info.callouts.add` and `info.callouts.remove`. See the section called “ info namespace ” for details.

- *Methods*

  It is possible to specify that a given HAL device object implements a specific D-Bus interface, e.g. `org.freedesktop.Hal.Device.Frob` with a set of methods `Foo`, `Bar` and `Baz` and have programs run when applications call into this interface. This is defined in the `info.interfaces` property, consult the section called “ info namespace ” for details.

- *Addons*

  An *addon* can be characterized as a daemon whose life cycle is tied to a device object in HAL. And addon can also *claim* a specific interface on the device object to provide services to applications for configuring / using the device without having to spawn a new program for every method call. HAL provides a facility to launch/destroy one or more addons per device object using the `info.addons` property. See the section called “ info namespace ” for details.

- *Device Information Files*

  A set of files that matches properties on device objects and merges additional information. These files are used, for among other things, to specify what callouts, methods and addons to associate with a device object. For example, for drives using removable media, HAL includes an add-on daemon which sole purpose is to continously poll the drive to detect media change.

The D-Bus system message bus is used to provide a ''network API'' to applications. As D-Bus is designed to be language independent, potentially many languages / runtime systems will be able to easily access the services offered by HAL.

Device Objects

It is important to precisely define the term HAL device object. It's actually a bit blurry to define in general, it includes what most UNIX-like systems consider first class objects when it comes to hardware. In particular, a device object should represent the smallest unit of addressable hardware. This means there can be a one-to-many relationship between a physical device and the device objects exported by HAL. Specifically, a multi-function printer, which appear to users as a single device may show up as several device objects; e.g. one HAL device object for each of the printing, scanning, fax and storage interfaces. Conversely, some devices may be implemented such that the HAL device object represent several functional interfaces. HAL is not concerned with this duality of either one-to-many or many-to-one relationships between device objects and the actual iron constituting what users normally understand as a single piece of hardware; a device object represents the smallest addressable unit.

Device objects in HAL are organised on a by-connection basis, e.g. for a given device object X it is possible to find the device object Y where X is attached to Y. This gives structure to the device database of HAL; it is possible to map the devices out in a tree. Further, software emulation devices exported by the operating system kernel, such as SCSI emulation for USB Storage Devices, are also considered device objects in HAL. This implies that operating system kernel specific bits leak into the device object database. However applications using HAL will not notice this, such device objects are not referenced anywhere in the device objects that users are interested in; they are merely used as glue to build the device tree.

In addition to provide information about what kind of hardware a device object represents (such as a PCI or USB device) and how to address it, HAL merges information about the functional interfaces the operating system kernel provides in order to use the device; in most cases this is represented on the device object as a string property with the name of the special device file in `/dev`. In addition to the special device file, a number of other useful properties are merged. This means that both hardware and functional properties are on the same device object, which may prove to be useful for an application programmer. For example, an application might query HAL for the device object that exports the special device file `/dev/input/mouse2` and learn that this is provide by an USB mouse from a certain manufacturer by checking the properties that export the USB vendor and product identifiers. See the section called “Device Capabilities” and Chapter 5, *Device Properties* for details.

Finally, HAL provides one or more *D-Bus interfaces* for applications to configure and/or use the device. These interfaces are discussed in Chapter 6, *D-Bus interfaces*.

Summarizing, a device object is comprised by

- *UDI*

  This is the the *Unique Device Identifer*, that is unique for a device object - that is, no other device object can have the same UDI at the same time. The UDI is computed using bus-specific information and is meant to be unique across device insertions and independent of the physical port or slot the device may be plugged into.

- *Properties*

  Each device object got a set of properties which are key/value pairs. The key is an ASCII string while the value can be one of several types, see below. Properties are arranged into name spaces using ''.'' as a separator.

  - `string` - UTF8 string
  - `strlist` - ordered list with UTF8 strings
  - `int` - 32-bit signed integer
  - `uint64` - 64-bit unsigned integer
  - `bool` - truth value
  - `double` - IEEE754 double precision floating point number

- *Interfaces*

  Applications can configure and/or use a device using D-Bus interfaces. Typically, there's a one-to-one relationship between capabilities/namespaces and interfaces.

Properties of a device object carry all the important information about a device object. For organisational reasons properties are also namespaced using ''.'' as a separator.

It can be useful to classify properties into four groups

- Metadata - Information about how the devices are connected with respect to each other (parent/child relationships), what kind of device it is, what functionality it provides etc.
- Facts - vendor ID, product ID, disk serial numbers, number of buttons on a mouse, formats accepted by a mp3 player and so on.
- Usage specific information - Network link status, special device file name, filesystem mount location etc.
- Policy - How the device is to be used be users; usually defined by the system administrator.

The first category is determined by HAL, the second category includes information merged from either querying the hardware itself or from device information files. The third category is intercepted by monitoring the hardware and finally the last is merged from files under control of the system administrator. This document is concerned with precisely defining several properties; see Chapter 5, *Device Properties* and onwards for more information. As a complement to device properties, HAL also provides *conditions* on HAL device objects. Conditions are used to relay events that are happening on devices which are not easily expressed in properties. This includes events such as ''processor is overheating'' or ''block device unmounted''.

There is a special hal device object referred to as the ''root computer device object''. This device object represent the entire system as a whole and all other devices are either directly or indirectly childs of this device object. It has the UDI `/org/freedesktop/Hal/devices/computer`.

The fundamental idea about HAL is that all ''interesting'' information about hardware that a desktop application needs, can be obtained by querying HAL.

Device Capabilities

Mainstream hardware isn't very good at reporting what it really is, it only reports, at best, how to interact with it. This is a problem; many devices, such as MP3 players or digital still cameras, appear to the operating system as plain USB Mass Storage devices when the device in fact is a lot more than just that. The core of the problem is that without external metadata, the operating system and desktop environment will present it to the user as just e.g. a mass storage device.

As HAL is concerned with merging of external metadata, through e.g. device information files, there needs to be some scheme on how to record what the device actually is. This is achieved by two textual properties, `info.category` and `info.capabilities`. The former

describes *what the device is* (as a single alphanumeric keyword) and the latter describes *what the device does* (as a number of alphanumeric keywords separated by whitespace). The keywords available for use is defined in this document; we'll refer to them in following simply as *capabilities*.

HAL itself, assigns capabilities on device detection time by inspecting the device class (if available, it depends on the bus type) and looking at information from the operating system and the hardware itself.

User mode drivers such as `libgphoto2` and `sane` provides device information to merge information about devices they can drive. As such, device objects represent an USB interface gain additional properties such as ''scanner'' or ''camera''.

Having a capability also means that part of the property namespace, prefixed with the capability name, will be populated with more specific information about the capability. Indeed, some properties may even be required such that applications and device libraries have something to expect. For instance, the capability for being a MP3 player may require properties defining what audio formats the device support (e.g. Ogg and MP3), whether it support recording of audio, and how to interact with the device. For example, the latter may specify ''USB Storage Device'' or ''proprietary protocol, use libfooplayer''.

Finally, capabilities have an inheritance scheme, e.g. if a device has a capability `foo.bar`, it must also have the capability `foo`.

Chapter 2. Device Information Files

Table of Contents

Device information files (`.fdi` files is a shorthand) are used to merge arbitrary properties onto device objects. The way device information files works is that once all device properties are merged onto a device object it is tried against the set of installed device information files. Device information files are used for both merging facts and policy settings about devices.

Matching

Each device information file got a number of `<match key="some_property" [string|int|bool|..]="required_value" >` directives that is tested against the properties of the device object. If all the match directives passes then the device information can include `<[merge|append|prepend|addset] key="some_property" type="[string|int|bool|..]">` directives to respectively merge new properties or append to existing properties on the device object. It's important to emphasize that any previously property stemming from device detection can be overridden by a device information file.

The `<match>`, `<merge>`, `<append>`, `<prepend>` and `<addset>` directives always requires the `key` attribute which must be either a property name on the device object in question or a path to a property on another device object. The latter case is expressed either through direct specification of the UDI, such as `/org/freedesktop/Hal/devices/computer:foo.bar` or indirect references such as `@info.parent:baz`

where the latter means that the device object specified by the UDI in the string property `info.parent` should be used to query the property `baz`. It is also possible to use multiple indirections, e.g. for a volume on a USB memory stick the indirection `@block.storage_device:@storage.originating_device:usb.vendor_id` will reference the `usb.vendor_id` property on the device object representing the USB interface.

When the property to match have been determined a number of attributes can be used within the `<match>` tag:

- `string` - match a string property; for example `<match key="foo.bar" string="baz">` will match only if 'foo.bar' is a string property assuming the value 'baz'.
- `string_outof` - work like `string` but the match is done against a list of strings separated by ';'. For example: `<match key="system.hardware.product" string_outof="Satellite A30;Portable PC">` In this example the line matches if `system.hardware.product` is exactly 'Satellite A30' or 'Portable PC'.
- `int` - match an integer property
- `int_outof` - work like `int` but the match is done against a list of integer separated by ';'. For example: `<match key="usb.product_id" int_outof="0x1007;0x1008;0x1009">` In this example the line matches if `usb.product_id` is 0x1007, 0x1008 or 0x1009.
- `uint64` - match property with the 64-bit unsigned type
- `bool` - match a boolean property
- `double` - match a property of type double
- `exists` - used as `<match key="foo.bar" exists="true">`. Can be used with 'true' and 'false' respectively to match when a property exists and it doesn't.
- `empty` - can only be used on string or strlist properties with 'true' and 'false'. The semantics for 'true' is to match only when the string is non-empty.
- `is_ascii` - matches only when a string property contain only ASCII characters. Can be used with 'true' or 'false'.
- `is_absolute_path` - matches only when a string property represents an absolute path (the path doesn't have to exist). Can be used with 'true' or 'false'.
- `sibling_contains` - can only be used with string and strlist (string list). For a string key this matches when a sibling item contains the (sub-)string in the same property. For a string list, this is if a string matches an item in the list.
- `contains` - can only be used with string and strlist (string list). For a string key this matches when the property contains the given (sub-)string. For a string list this match if the given string match a item of the list.

- `contains_ncase` - like `contains` but the property and the given key are converted to lowercase before check.
- `contains_not` - can only be used with strlist (string list) and string properties. For a string list this match if the given string not match any of the item of the list (or the property is not set for the device). For a string this match of the property not contains the (sub-)string. You can use this attribute to construct if/else blocks together with e.g. `contains`.
- `contains_outof` - like `contains` but can be used only with strings and the match is done against a list of (sub-)strings separated by ';'. For example: `<match key="system.hardware.product" contains_outof="D600;D610;C540">` In this example the line matches if `system.hardware.product` contains D600, D610 or C540.
- `prefix` - can only be used with string properties. Matches if property begins with the key.
- `prefix_ncase` - like `prefix` but the property and the given key are converted to lowercase before the check.
- `prefix_outof` - like `prefix` but the match is done against a list of prefixes separated by ';'. For example: `<match key="system.hardware.product" prefix_outof="1860;2366;2371">` In this example the line matches if `system.hardware.product` starts with 1860, 2366 or 2371.
- `suffix` - can only be used with string properties. Matches if property ends with the key.
- `suffix_ncase` - like `suffix` but the property and the given key are converted to lowercase before the check.
- `compare_lt` - can be used on int, uint64, double and string properties to compare with a constant. Matches when the given property is less than the given constant using the default ordering.
- `compare_le` - like `compare_lt` but matches when less than or equal.
- `compare_gt` - like `compare_lt` but matches when greater than.
- `compare_ge` - like `compare_lt` but matches when greater than or equal.
- `compare_ne` - like `compare_lt` but matches when not equal.

Merging

The `<merge>`, `<append>`, `<prepend>` and `<addset>` directives all require the `type` attribute which specifies what to merge. The following values are supported

- `string` - The value is copied to the property. For example `<merge key="foo.bar" type="string">baz</merge>` will merge the value 'baz' into the property 'foo.bar'.
- `strlist` - For `<merge>` the value is copied to the property and the current property will be overwritten. For `<append>` and `<prepend>`

the value is append or prepend to the list as new item. For `<addset>` the strlist is treated as a set and the value is appended if, and only if, the value doesn't exist already. Usage of `<copy_property>` overwrite the complete list with the value of the given property to copy from.

- `bool` - Can merge the values 'true' or 'false'
- `int` - Merges an integer
- `uint64` - Merges an unsigned 64-bit integer
- `double` - Merges a double precision floating point number
- `copy_property` - Copies the value of a given property - supports paths with direct and indirect UDI's. For example `<merge key="foo.bar" type="copy_property">@info.parent:baz.bat</merge>` will merge the value of the property `baz.bat` on the device object with the UDI from the property `info.parent` into the property `foo.bar` on the device object being processed.

The `<remove>`, directive require only a key and can be used with all keys. For `strlist` there is additionally a special syntax to remove a item from the string list. For example to remove item 'bla' from property 'foo.bar': `<remove key="foo.bar" type="strlist">bla</remove>`

Search Paths

Device Information files are read from two directories

- `/usr/share/hal/fdi` - for files provided by packages
- `/etc/hal/fdi` - for files provided by the system administrator / user

in exactly that order. This means that the files provided by the system administrator will be processed last such that they can overwrite / change properties caused by the device information files provided by packages. The following directory structure is used in `/usr/share/hal/fdi`

- `information` - device information files used to merge device information
  - `10freedesktop` - included with the hal package
  - `20thirdparty` - from a 3rd party, not included in hal package
- `policy` - device information files to merge policy properties such as addons or callouts.
  - `10osvendor` - included with the hal package

    - `20thirdparty` - from a 3rd party, not included in hal package
- `preprobe` - device information files read before probing devices
    - `10osvendor` - included with the hal package
    - `20thirdparty` - from a 3rd party, not included in hal package

As evident, third party packages should drop device information files in

- `/usr/share/hal/fdi/information/20thirdparty`
- `/usr/share/hal/fdi/policy/20thirdparty`
- `/usr/share/hal/fdi/preprobe/20thirdparty`

The `/etc/hal/fdi` tree uses this layout

- `information` - device information files used to merge device information
- `policy` - device information files to merge policy properties such as addons or callouts.
- `preprobe` - device information files to read before probing devices

All device information files are matched for every hal device object in the following order.

1. When a device is discovered, the `preprobe` device information files (e.g. all files from `/usr/share/hal/fdi/preprobe` and `/etc/hal/fdi/preprobe`) are processed.

   Typically, this class of device information files is used to tell HAL to leave the device alone by setting the bool property `info.ignore` to TRUE. It can also be used to run programs, preprobe callouts, prior to normal device investigation.

2. HAL now runs the preprobe callouts.

3. HAL now probes/investigates the device.

4. All the `information` device information files (e.g. all files from `/usr/share/hal/fdi/information` and `/etc/hal/fdi/information`) are processed.

   These device information files are typically used to associate extra information with a device object.

5. All the `policy` policy information files (e.g. all files from `/usr/share/hal/fdi/policy` and `/etc/hal/fdi/policy`) are processed.

These device information files are typically used to associate callouts and addons with a device object.

6. HAL now runs the callouts, starts addons, and then finally announces the device on the system message bus.

Chapter 3. Access Control

Table of Contents

Access to hardware by unprivileged users is traditionally granted in two ways either by granting access to the *special device file* or allowing access through another process, using IPC acting on behalf of the user. HAL follows the latter model and uses the system-wide message bus (D-Bus) as the IPC mechanism. In addition, HAL has support for modifying the ACL's (access control lists) on a device file to grant/revoke access to users based on several criteria.

Device Files

If HAL is built with `--enable-acl-management` (requires both `--enable-console-kit` and `--enable-policy-kit`) then ACL's on device objects with the capability `access_control` are automatically managed according to the properties defined in the section called "access_control namespace". In addition, for this configuration, HAL ships with a device information file (normally installed in `/usr/share/hal/fdi/policy/10osvendor/20-acl-management.fdi`) that merges this capability on device objects that are normally accessed by unprivileged users through the device file. This includes e.g. sound cards, webcams and other devices but excludes drives and volumes as the latter two are normally accessed by a user through mounting them into the file system.

HAL uses PolicyKit to decide what users should have access according to PolicyKit configuration; see the PolicyKit privilege definition file `/usr/share/PolicyKit/policy/org.freedesktop.hal.device-access.policy` on a system with HAL installed for the default access suggested by the HAL package and/or OS vendor.

In addition, 3rd party packages can supply device information files to specify (via the `access_control.grant_user` and `access_control.grant_group` properties) that a given user or group should always have access to a device file. This is useful for system-wide software (such as AV streaming management) that runs as an unprivileged system user. This interface is supposed to be stable so 3rd party packages can depend on it.

Device Files policies

This is a list of the device file policies/rules delivered with the HAL package to manage ACL's as defined via `access_control.type` and the current default Policykit policies for inactive and active users.

Type	Description	allow_inactive	allow_active
audio-player	Directly access audio players.	no	yes
camera	Directly access digital cameras.	no	yes
cdrom	Directly access optical drives.	yes	yes
dvb	Directly access DVB devices.	no	yes
fingerprint-reader	Directly access to fingerprint reader devices.	no	yes
floppy	Directly access Floppy devices.	yes	yes
ieee1394-avc	Directly access Firewire AVC devices.	no	yes
ieee1394-iidc	Directly access Firewire IIDC devices.	no	yes
smart-card-reader	Directly access Smart Card Reader security devices.	no	yes
joystick	Directly access Joystick devices.	yes	yes
modem	Directly access serial modem devices.	auth_admin_keep_always	auth_admin_keep_always
mouse	Directly access Mouse (input) devices	yes	yes
obex	Directly access OBEX devices.	no	yes
pda	Directly access PDA devices.	no	yes
ppdev	Directly access parallel port devices.	auth_admin_keep_always	auth_admin_keep_always
printer	Directly access printer devices.	no	yes
removable-block	Directly access removable block devices.	no	no
scanner	Directly access scanners.	no	yes
sound	Directly access sound devices.	no	yes
video	Directly access Video devices.	yes	yes
video4linux	Directly access video capture devices.	no	yes

D-Bus Interfaces

If HAL is built without ConsoleKit support (e.g. without `--enable-console-kit`) access to the various D-Bus interfaces that provides mechanisms is only protected by the D-Bus security configuration files (e.g. using `at_console` to restrict to console user on Red Hat systems) and, in certain cases, restricted to the super user.

If ConsoleKit support is enabled, access to D-Bus interfaces is currently hardcoded to only allow active users at the system console. If PolicyKit support is enabled, the PolicyKit library will be in charge of determining access; see the PolicyKit privilege definition files in `/etc/PolicyKit/privileges` on a system with HAL installed for the default access suggested by the HAL package and/or OS vendor.

Chapter 4. Locking

Table of Contents

As HAL is a mechanism that enables programs in a desktop session to enforce the policy of the users choice, unexpected things can happen. For example, if the user is in the middle of partitioning a disk drive, it is desirable to keep the desktop from mounting partitions that have not yet been prepared with a suitable file system. In fact, in such a situation data loss may be the result if a volume have an old file system signature indicating it's mountable and, simultenously, another tool is writing to the raw block device. The mechanism that automounters use, HAL, provides locking primitives to avoid this.

Further, for multi-user systems, several desktop sessions may run on a system each on their own display. Suppose that one session becomes idle and the power management daemon in that session decides to suspend the system according to user preferences in the idle session. The result is that users at other seats will see the system suspend and this is not desirable. The power management daemons in all sessions need to cooperate to ensure that the system only suspends when e.g. all sessions are idle or not at all. The mechanism that each power management daemon uses, HAL, provides locking primitives that can be used to achieve this.

Overview

HAL provides a mechanism to lock a specific D-Bus interface either for a specific device or for all the devices the caller have access to.

The former is achieved by using the `AcquireInterfaceLock()` and `ReleaseInterfaceLock()` methods on the `org.freedesktop.Hal.Device` interface that every device object implements (see the section called “org.freedesktop.Hal.Device interface”). By using this API, a caller can prevent any other caller from invoking methods on the given interface for the given device object - other callers will simply see the `org.freedesktop.Hal.Device.InterfaceLocked` exception if they attempt to invoke a method on the given interface on the given device. The locker can specify whether the lock is *exclusive* meaning if multiple clients clients can hold the lock or if only one client can hold the lock at one time. If a client don't have access to the interface of the device, attempts to lock will fail with a `org.freedesktop.Hal.PermissionDenied` exception. If a client loses access to a device (say, if his session is switched away from using fast user switching) while holding a lock, he will lose the lock; this can be tracked by listening to the `InterfaceLockReleased` signal.

All local clients, whether they are active or not, can always lock interfaces on the root computer device object (this doesn't mean that they are privileged to use the interfaces though) - the rationale is that this device object represents shared infrastructure, e.g. power management, and even inactive sessions needs to participate in managing this.

If another client already holds a lock exclusively, attempts from other clients to acquire the lock will fail with the `org.freedesktop.Hal.Device.InterfaceAlreadyLocked` exception even if they have access to the device.

In addition, a client may opt to lock all devices that he got access to by using the `AcquireGlobalInterfaceLock()` and `ReleaseGlobalInterfaceLock()` methods on the `org.freedesktop.Hal.Manager` interface on the `/org/freedesktop/Hal/Manager` object (see the section called "org.freedesktop.Hal.Manager interface"). Global interface locks can also be obtained exclusively if the caller so desires. Unlike per-device interface locking, it is not checked at locking time whether the locker have access to a given device; instead checking is done when callers attempt to access the interface.

The algorithm used for determining if a caller is locked out is shown below. A caller A is locked out of an interface IFACE on a device object DEVICE if, and only if,

1. Another caller B is holding a lock on the interface IFACE on DEVICE and A don't have either a global lock on IFACE or a lock on IFACE on DEVICE; or

2. Another caller B is holding the global lock on the interface IFACE and B has access to DEVICE and and A don't have either a global lock on IFACE or a lock on IFACE on DEVICE.

In other words, a caller A can grab a global lock, but that doesn't mean A can lock other clients out of devices that A doesn't have access to. Specifically a caller is never locked out if he has locked an interface either globally or on the device in question. However, if two clients have a lock on a device, then both can access it. To ensure that everyone is locked out, a caller needs to use an exclusive lock.

Note that certain interfaces will also check whether other locks are being held on other device objects. This is specified on a per-interface basis in Chapter 6, *D-Bus interfaces*.

If a process holding locks disconnects from the system bus, the locks being held by that process will be released.

Guidelines

Locking is only useful if applications requiring exclusive access actually use the locking primitives to cooperate with other applications. Here is a list of guidelines.

- *Disk Management / Partitioning*

  In order to prevent HAL-based automounters from mounting partitions that are being prepared, applications that access block devices directly (and pokes the kernel to reload the partitioning table) should lock out automounters by either a) obtaining the `org.freedesktop.Hal.Device.Storage` lock on each drive being processed; or b) obtaintaing the global `org.freedesktop.Hal.Device.Storage` lock. This includes programs like fdisk, gparted, parted and operating system installers. See also the section called "org.freedesktop.Hal.Device.Volume interface" and the `hal-lock`(1) program and manual page.

- *Power Management*

  Typically, a desktop session includes a session-wide power management daemon that enforces the policy of the users choice, e.g. whether the system should suspend to ram on lid close, whether to hibernate the system after the user being idle for 30 minutes

and so on. In a multi-user setup (both fast user switching and multi-seat), this can break in various interesting ways unless the power management daemons cooperate. Also, there may be software running at the system level who will want to inhibit a desktop session power management daemon from suspending / shutting down.

- System-level software that do not wish to be interrupted by the effect of someone calling into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface MUST hold the `org.freedesktop.Hal.Device.SystemPowerManagement` lock non-exclusively on the root computer device object. For example, the YUM software updater should hold the lock when doing an RPM transaction.

In addition, any power management session daemon instance

- ... MUST hold the `org.freedesktop.Hal.Device.SystemPowerManagement` lock non-exclusively on the root computer device object unless it is prepared to call into this interface itself. This typically means that the PM daemon instance simply acquires the lock on start up and releases it just before it calls into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface. In other words, the PM daemon instance needs to hold the lock exactly when it doesn't want other PM daemon instances to call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface. This means that if the user have configured the PM daemon instance to go to sleep after 30 minutes of inactivity, the lock should be released then.

- ... MUST not hold the lock when the session is inactive (fast user switching) UNLESS an application in the session have explicitly called Inhibit() on the `org.freedesktop.PowerManagement` D-Bus session bus interface of the PM daemon.

- ... MUST check that no other process is holding the lock (using the `IsLockedByOthers` method on the standard `org.freedesktop.Hal.Device` interface) before calling into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface. If another process is holding the lock, it means that either 1) another session is not prepared to call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface; OR 2) some system-level software is holding the lock. The PM daemon instance MUST respect this by not calling into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface itself.

However, any Power management daemon instance

- ... MAY prompt the user, if applicable, to ask if she still wants to perform the requested action (e.g. call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface) despite the fact that another process (possibly from another user) is indicating that it does not want the system to e.g. suspend. Only if the user agrees, the power management instance should call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface. Typically, it's only useful to prompt the user with such questions if the request to call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface originates from user input, e.g. either a hotkey, the user clicking a suspend button in the UI or an application invoking the `Suspend()` method on the `org.freedesktop.PowerManagement` D-Bus session interface of the PM daemon.

- ... MAY ignore that other processes are holding the lock and call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface anyway, but ONLY if if the request to call into the `org.freedesktop.Hal.Device.SystemPowerManagement` interface originated from e.g. lid close, critically low battery or other similar conditions.

- ... MAY still call `SetPowerSave()` on the `org.freedesktop.Hal.Device.SystemPowerManagement` interface even if other processes are

holding the lock.

Chapter 5. Device Properties

Table of Contents

Properties are arranged in a namespaces using ''.'' as a separator and are key/value pairs. The value may assume different types; currently int32, double, bool, UTF8 strings and UTF8 string lists are supported. The key of a property is always an ASCII string without any whitespace. When a property changes, HAL will emit a D-Bus signal that applications can catch.

org.freedesktop.Hal namespace

The `org.freedesktop.Hal` namespace contain properties that can be considered metadata about HAL itself and not about device objects. They are only available at the root object (`/org/freedesktop/Hal/devices/computer`) and allow to query information about HAL from fdi files.

Key (type)	Values	Mandatory	Description

Key (type)	Values	Mandatory	Description
org.freedesktop.Hal.version (string)	example: 0.5.13	Yes	The version number of the running HAL daemon.
org.freedesktop.Hal.version.major (int)	example: 0	Yes	The major version number of the running HAL daemon.
org.freedesktop.Hal.version.minor (int)	example: 5	Yes	The minor version number of the running HAL daemon.
org.freedesktop.Hal.version.micro (int)	example: 13	Yes	The micro version number of the running HAL daemon.

General Properties

The section represents properties that aren't tied to either physical or functional characteristics of what the device object represents.

info namespace

The info namespace contain properties that can be considered metadata about device objects. These properties are always available.

Key (type)	Values	Mandatory	Description
info.subsystem (string)	example: pci, usb, ide_host, ide, block, usbif, scsi_host, scsi	Yes	Describes what subsystem the device is connected to
info.udi (string)	example: /org/freedesktop/Hal/devices/pci_10ec_8139	Yes	The HAL unique device id
info.capabilities (strlist)	example: 'block, storage, storage.cdrom'	No	A string list of capabilities describing what the devices does
info.category (string)	example: storage.cdrom	No	The prominent capability describing what the device is
info.product (string)	examples: ''SleekKeyboard'', ''MouseMan 2003'', ''Volume'', ''LS-120 SLIM3 00 UHD Floppy''	No	The name of the device; should not be used in any UI; use subsystem / capability specific properties instead.
info.vendor (string)	examples: Logitch, Mustek	No	The name of the vendor of the device; should not be used in any UI; use subsystem / capability specific properties instead.
info.parent (string)	example: /org/freedesktop/Hal/devices/computer	Yes, for all non-root device objects	The UDI of the device object that this device object is connected to.
info.locked (bool)		No	If this property is available and set to TRUE it means that a process is using the device that the hal device object in question represents and no other process should attempt to use or configure the device. The

Key (type)	Values	Mandatory	Description
			lock is only advisory.
info.locked.reason (string)	example: ''The optical drive is currently being used to record a CD-RW disc.''	Only available if info.locked is set to TRUE.	A localized text suitable for UI display
info.locked.dbus_service (string)	example: :1.278	Only available if info.locked is set to TRUE.	The base D-BUS service of the process holding the lock.
info.is_recalled (bool)		No	This is set if the hardware may be recalled and should be checked for any potential problem.
info.recall.vendor (string)	Dell, Sony, HP, Panasonic, etc.	Yes, if info.is_recalled is TRUE	The vendor responsible for the hardware recall.
info.recall.website_url (string)		Yes, if info.is_recalled is TRUE	Users should check this website for more details and if their hardware may affected by any possible fault.

linux namespace

The linux namespace contain properties that can be considered metadata about device objects in Linux systems. These properties are available only on Linux systems.

Key (type)	Values	Mandatory	Description
linux.subsystem (string)	pci, usb, ide_host, ide, block, usb, usbif, scsi_host, scsi	Yes	Describes what Linux subsystem the device is connected to. This can differ from info.subsystem
linux.sysfs_path (string)	for example: /sys/class/sound/seq	No	Path to the devuce in the sysfs. Could also be *.linux.sysfs_path depending on the subsystem in some cases.
linux.device_file (string)	for example: /dev/snd/pcmC0D1c or /dev/input/event6	No	Path to the corresponding device file in /dev/ on linux system.
info.linux.driver (string)	for example: pcspkr, vesafb, serial8250	No	Name of the driver/module corresponding to the device and/or subsystem.
linux.hotplug_type (int)		Yes	The type of hotplug event in a linux system.
linux.acpi_type (int)		No (except for ACPI devices)	The type of ACPI device. Normaly only for HAL internal use.

Key (type)	Values	Mandatory	Description
linux.apm_type (int)		No (except for APM devices)	The type of APM device. Normaly only for HAL internal use.
linux.pmu_type (int)		No (except for PMU devices)	The type of PMU device. Normaly only for HAL internal use.

Callouts

Callouts are programs invoked when the device object are added and removed. As such, callouts can be used to maintain system-wide policy (that may be specific to the particular OS) such as changing permissions on device nodes, updating the systemwide /etc/fstab file or configuring the networking subsystem.

There are three different classes of callouts. A callout involves sequentially invoking all executable programs in the string list in listed order.

All callouts are searched for and execute in a minimal environment. In addition, the UDI of the device object is exported in the environment variable UDI. All properties of the device object are exported in the environment prefixed with HAL_PROP_. If a device is added or removed is exported in the environment variable HALD_ACTION . The search path for the callout includes the following paths:

1. $libexecdir (typically /usr/libexec (e.g. Red Hat) or /usr/lib/hal (e.g. Debian))
2. $libdir/hal/scripts (typically /usr/lib/hal/scripts or /usr/lib64/hal/scripts)
3. $bindir/ (typically /usr/bin)

including $PATH the HAL daemon was started with during system initialization. Depending on the distribution, this typically includes /sbin, /usr/sbin, /bin, /usr/sbin. If the program to run is not found in any of these paths, the it *will not* run even if the given path is absolute. To be portable across operating systems, third party packages providing callouts must therefore only use $libdir/hal/scripts.

If ConsoleKit support is enabled, the variables CK_NUM_SEATS (number of seats), CK_NUM_SESSIONS (number of sessions), CK_SEATS (tab sep. list of seat-id's), CK_SEAT_seat-id (tab sep. list of session-id's for a seat), CK_SEAT_NUM_SESSIONS_seat-id (number of sessions on a seat), CK_SESSION_SEAT_session-id (the seat that a session belongs to) and CK_SESSION_IS_ACTIVE_session-id (whether a given session is active) and CK_SESSION_UID_session-id (the user of the session) and CK_SESSION_IS_LOCAL_session-id (whether a session is local), CK_SESSION_HOSTNAME_session-id (host name of session's display if it's not local), will be exported as well. Example:

```
CK_NUM_SEATS=1
CK_NUM_SESSIONS=2
CK_SEATS=Seat1
CK_SEAT_Seat1=Session1  Session3
CK_SEAT_NUM_SESSIONS_Seat1=2
CK_SESSION_IS_ACTIVE_Session1=true
```

```
CK_SESSION_IS_ACTIVE_Session3=false
CK_SESSION_IS_LOCAL_Session1=true
CK_SESSION_IS_LOCAL_Session3=true
CK_SESSION_SEAT_Session1=Seat1
CK_SESSION_SEAT_Session3=Seat1
CK_SESSION_UID_Session1=500
CK_SESSION_UID_Session3=501
```

Note that all ConsoleKit object paths given are just base names; the real D-Bus object path can be reconstructed by appending `/org/freedesktop/ConsoleKit/` prepended to the given identifer.

The HAL daemon is not suspended while callouts are executing. Thus, callouts can communicate with the HAL daemon using the D-BUS network API. Hence, one application of callouts is to merge or modify properties on a device object.

To reduce round trips and increase privacy, callouts can (and should) communicate with the HAL daemon using a peer to peer D-Bus connection specified by the `HALD_DIRECT_ADDR` environment variable. There is convience API in libhal to do this.

Key (type)	Values	Mandatory	Description
`info.callouts.add` (string list)		No	A string list with the programs which should be executed (with `HALD_ACTION=add`) when the device is added to the GDL (global device list) but just before it is announced through the D-BUS network API.
`info.callouts.remove` (string list)		No	A string list with the programs that should be executed (with `HALD_ACTION=remove`) when the device is removed from the GDL (global device list). The device isn't removed before the last callout has finished.
`info.callouts.preprobe` (string list)		No	A string list with the programs that should be executed (with `HALD_ACTION=preprobe`) before the device is probed (e.g. investigated) and can be used to avoid causing unnecessary I/O.
`info.callouts.session_add` (string list)		No	A string list with all programs that should be executed (with `HALD_ACTION=session_add`) when a session is added. Can only be set on the root computer device object. The environment also contains the variables `HALD_SESSION_ADD_SESSION_ID`, `HALD_SESSION_ADD_SESSION_UID` and `HALD_SESSION_ADD_SESSION_IS_ACTIVE` to identify the session. This is only used when HAL is built with ConsoleKit support.
`info.callouts.session_remove` (string list)		No	A string list with all programs which should be executed (with `HALD_ACTION=session_remove`) when a session is removed. Can only be set on the root computer device object. The environment also contains the variables `HALD_SESSION_REMOVE_SESSION_ID`, `HALD_SESSION_REMOVE_SESSION_UID` and `HALD_SESSION_REMOVE_SESSION_IS_ACTIVE` to identify the session. This is only used when HAL is built with ConsoleKit support.

Addons

Addons are programs that run for the life time of the device object. They are searched for and execute in the same environment as callouts (e.g. with `HAL_PROP_*` set in the environment to represent the device properties) and are launched just before the device is announced on D-Bus (but just after the last add callouts have finished). When the device object goes away, HAL will send a `SIGTERM` to the process.

Key (type)	Values	Mandatory	Description
`info.addons` (strlist)		No	List of programs to run when device is added. Each program will need to call the `AddonIsReady()` method in order for the device to show up on D-Bus.

Singleton Addons

Singleton Addons are programs that are started by HAL to handle a set of devices. They are identified by the command line used to start them. They MUST implement the org.freedesktop.Hal.SingletonAddon interface. on the path `/org/freedesktop/Hal/Singleton` path on the direct connection to the HAL daemon.

When a device is added with an `info.addons.singleton` string list key, the elements of that key are used as the command line to start the singleton if the singleton is not already running. Once the singleton has called `SingletonAddonIsReady` on org.freedesktop.Hal.Manager interface, it will receive `DeviceAdded` calls on its org.freedesktop.Hal.SingletonAddon interface for all devices that have its commandline in `info.addons.singletona`.

If a device is added and the singleton specified in `info.addons.singleton` is already running, the singleton will recieve `DeviceAdded` on its org.freedesktop.Hal.SingletonAddon interface for that new device.

When a device is removed that is being handled by a singleton, the singleton will recieve `DeviceRemoved` on org.freedesktop.Hal.SingletonAddon . When it is no longer handling any more devices it should exit cleanly.

Key (type)	Values	Mandatory	Description
`info.addons.singleton` (strlist)		No	A list of commandlines for singleton addons which should service this device.

Method calls

Method calls on a specific interface on a device object can be implemented by the HAL daemon running a program. Note that this is not the only way to implement support for method calls; if you expect a lot of method calls it is preferable to implement an addon and use the `ClaimInterface()` API since it reduces the overhead of spawning a process and it can handle both complex incoming and return types as well. See the section called “org.freedesktop.Hal.Device interface” for details on claiming interfaces via an addon..

Note that method calls implemented via running a program are limited to the return type being an a signed 32-bit integer (this will change in a future release). The incoming parameters are limited to only basic types and arrays of strings. The parameters are passed via stdin using a textual representation. As such, there is a lot of overhead with handling method calls by spawning programs

and as such it should only be used for situtations where the nature of the method call is that it will not be frequently used.

As with addons, method calls are searched for and execute in the same minimal environment as callouts (e.g. with HAL_PROP_* set in the environment to represent the device properties) and in addition the environment variables HAL_METHOD_INVOKED_BY_UID (the uid of the caller) and HAL_METHOD_INVOKED_BY_SYSTEMBUS_CONNECTION_NAME (the unique system bus connection name of the caller) are set. Additionally, if HAL is built with ConsoleKit support, HAL_METHOD_INVOKED_BY_PID and HAL_METHOD_INVOKED_BY_SELINUX_CONTEXT (but only if the running system have SELinux enabled) will be set. If HAL itself, or a HAL addon, is invoking a method, then these variables will not be present. Here's an example

```
HAL_METHOD_INVOKED_BY_UID=500
HAL_METHOD_INVOKED_BY_PID=22553
HAL_METHOD_INVOKED_BY_SELINUX_CONTEXT=user_u:system_r:unconfined_t
HAL_METHOD_INVOKED_BY_SYSTEMBUS_CONNECTION_NAME=:1.138
```

In addition, with ConsoleKit support, HAL_METHOD_INVOKED_BY_SESSION will be set to (the basename) of the ConsoleKit session object path but only if the caller is in a session. The method handler can then use the previously mentioned CK_SESSION_* to learn everything about the context of the caller.

Key (type)	Values	Mandatory	Description
info.interfaces (strlist)		No	A list of D-Bus interfaces that the device object supports apart from the standard org.freedesktop.Hal.Device interface.
<iface>.method_names (strlist)	example: 'Foo', 'Bar', 'Baz'	No	If a D-Bus interface is implemented by executing a program for every method, this property contains an ordered list of the method names.
<iface>.method_argnames (strlist)	example: 'foo_arg1 foo_arg2', '', 'baz_arg1'	No	This property contains the names of the arguments for each method. Each entry is a white-space separated list for that particular method.
<iface>.method_signatures (strlist)	example: 'si', '', 'as'	No	This property contains the D-Bus signature for each method. The signature should only cover incoming arguments; each method is defined as returning an integer.
<iface>.method_execpaths (strlist)	example: 'foo-binary', 'bar-binary', 'baz-binary'	No	This property contains the name of the program to execute when this method is called. The return code of the program will be passed as the integer result to the D-Bus caller. If a program wants to return an error, it just needs to write two lines to stderr; the first line is the exception name to throw and the second line is the exception detail.

Items in the <iface>.* clearly must correspond with each other. The whole mechanism is best explained by an example:

```
info.interfaces = {'org.freedesktop.Hal.Device.Volume'}
org.freedesktop.Hal.Device.Volume.method_argnames = {'mount_point fstype extra_options', 'extra_options', 'extra_options'}
org.freedesktop.Hal.Device.Volume.method_execpaths = {'hal-storage-mount', 'hal-storage-unmount', 'hal-storage-eject'}
org.freedesktop.Hal.Device.Volume.method_names = {'Mount', 'Unmount', 'Eject'}
```

```
org.freedesktop.Hal.Device.Volume.method_signatures = {'ssas', 'as', 'as'}
```

which, for example, shows that the `Mount()` method on the interface `org.freedesktop.Hal.Device.Volume` takes three arguments: `mount_point` (a string), `fstype` (a string) and `extra_options` (an array of strings).

Subsystem-Specific Properties

In this section properties for device objects that represent addressable hardware is described. Availability of these depends on the value of the `info.subsystem` property. These properties are not of particular interest to application developers, instead they are useful for libraries and userspace drivers that needs to interact with the device given a UDI. Knowledge of various subsystem-specific technologies is assumed for this section to be useful.

bluetooth_acl namespace

Device objects representing Asynchronous Connection-oriented Links.

Key (type)	Values	Mandatory	Description
`bluetooth_acl.address` (uint64)		Yes	Address of the device at the other end of the connection.
`bluetooth_acl.originating_device` (string)		Yes	The UDI of the Bluetooth HCI (of capability `bluetooth_hci`) that the connection is made through.

bluetooth_hci namespace

Device objects representing a Bluetooth Host Controller Interface.

Key (type)	Values	Mandatory	Description
`bluetooth_hci.address` (uint64)		Yes	Address of the host controller interface.
`bluetooth_hci.originating_device` (string)		Yes	The UDI of the physical device (e.g. an USB interface) that provides the HCI hardware.

bluetooth_sco namespace

Device objects representing Synchronous Connection-Oriented links.

Key (type)	Values	Mandatory	Description
`bluetooth_sco.address` (uint64)		Yes	Address of the device at the other end of the connection.

Key (type)	Values	Mandatory	Description
bluetooth_sco.originating_device (string)		Yes	The UDI of the Bluetooth HCI (of capability bluetooth_hci) that the connection is made through.

block namespace

Device objects representing addressable block devices, such as drives and partitions, will have info.subsystem set to block and will export a number of properties in the block namespace.

Key (type)	Values	Mandatory	Description
block.device (string)	example: /dev/sda	Yes	Special device file to interact with the block device
block.major (int)	example: 8	Yes	Major number of special file to interact with the device
block.minor (int)	example: 1	Yes	Minor number of special file to interact with the device
block.is_volume (bool)		Yes	True only when the block device is a volume that can be mounted into the file system. In this case the volume capability will be set and thus, properties, in the volume namespace are available.
block.no_partitions (bool)		Yes	For toplevel block devices, this is TRUE only when no known partition tables have been found on the media (In this case, if the storage device contain a file system it will be accessible using the same special device file as the one for this device object and the device object representing the filesystem will appear as a separate device object as a child). For the child, that is when block.is_volume is true, this property is TRUE exactly when it was created for a storage device with the storage.no_partitions_hint set to TRUE.
block.have_scanned (bool)		Yes	An internal property used by HAL to specify whether a top level block device have already been scanned for filesystems.

ccw namespace

Device objects that represent s390 ccw devices (when info.subsystem is set to ccw) are represented by the properties below.

Key (type)	Values	Mandatory	Description
ccw.devtype (string)	example: 1732/01	Yes	Device type/model or n/a
ccw.cutype (string)	example: 1731/01	Yes	Control unit type/model
ccw.cmb_enable (int)	example: 1	Yes	If channel measurements are enabled

Key (type)	Values	Mandatory	Description
ccw.availability (string)	example: good	Yes	Can be one of 'good', 'boxed', 'no path', or 'no device'
ccw.online (int)	example: 1	Yes	Online status
ccw.bus_id (string)	example: 0.0.f588	Yes	The device's bus id in sysfs
ccw.subchannel.pim (int)	example: 0x80	No	path installed mask
ccw.subchannel.pam (int)	example: 0x80	No	path available mask
ccw.subchannel.pom (int)	example: 0xff	No	path operational mask
ccw.subchannel.chpid0..7 (int)	example: 0x40	No	channel path ids

The following properties describe ccw devices where linux.driver is either dasd-eckd or dasd-fba.

Key (type)	Values	Mandatory	Description
ccw.dasd.use_diag (int)	example: 0	Yes	If the device driver shall use diagnose calls to access the device
ccw.dasd.readonly (int)	example: 0	Yes	If the device can only be accessed readonly
ccw.dasd.discipline (string)	example: ECKD	No	The dasd discipline used to access the device

The following properties describe ccw devices where linux.driver is zfcp. They are only present when ccw.online = 1.

Key (type)	Values	Mandatory	Description
ccw.zfcp.in_recovery (int)	example: 0	Yes	Shows whether the adapter is currently in recovery
ccw.zfcp.failed (int)	example: 0	Yes	Shows whether the adapter is in failed state

The following properties describe ccw devices where linux.driver is of the form tape_3xxx .

Key (type)	Values	Mandatory	Description
ccw.tape.state (string)	example: IN_USE	Yes	The current status of the tape
ccw.tape.operation (string)	example: REW	Yes	A three-letter mnemonic of the current tape operation
ccw.tape.medium_state (string)	example: no medium	No	If ccw.online = 1, shows whether a tape is loaded
ccw.tape.blocksize (int)	example: 512	No	If ccw.online = 1, shows the blocksize used for reads and writes to the tape

The following properties describe ccw devices where linux.driver is 3270.

Key (type)	Values	Mandatory	Description

Key (type)	Values	Mandatory	Description
ccw.3270.model (int)	example: 3	Yes	The model of the device, determining rows and columns
ccw.3270.rows (int)	example: 32	Yes	The number of rows
ccw.3270.columns (int)	example: 80	Yes	The number of columns

ccwgroup namespace

Device objects that represent groups of ccw devices (when info.subsystem is set to ccwgroup have the properties specified below.

Key (type)	Values	Mandatory	Description
ccwgroup.online (int)	example: 1	Yes	Online status
ccwgroup.bus_id (string)	example: 0.0.f588	Yes	The device's bus id in sysfs

The following properties describe ccwgroup devices where linux.driver is qeth.

Key (type)	Values	Mandatory	Description
ccwgroup.qeth.large_send (string)	example: TSO	No	Whether large send is provided. Can be "no", "EDDP" (software) or "TSO" (hardware).
ccwgroup.qeth.card_type (string)	example: OSD_1000	Yes	Type of the card
ccwgroup.qeth.checksumming (string)	example: sw checksumming	No	The method used to checksum incoming packets
ccwgroup.qeth.canonical_macaddr (int)	example: 0	No	Specifies the token ring macaddress format. Not valid in layer2 mode and for ethernet devices.
ccwgroup.qeth.broadcast_mode (string)	example: broadcast_allrings	No	The scope of token ring broadcasts. Not valid in layer2 mode and for ethernet devices.
ccwgroup.qeth.fake_broadcast (int)	example: 0	No	Whether to fake broadcast capability. Not valid in layer2 mode.
ccwgroup.qeth.fake_ll (int)	example: 0	No	Whether to add a faked link level header to packets. Not valid in layer2 mode.
ccwgroup.qeth.layer2 (int)	example: 0	No	Whether the card operates in layer 2 mode
ccwgroup.qeth.portname (string)	example: OSAPORT	No	The port name which has been specified for the card
ccwgroup.qeth.portno (int)	example: 0	No	The relative port number on the card
ccwgroup.qeth.buffer_count (int)	example: 16	Yes	Number of inbound buffers used
ccwgroup.qeth.add_hhlen (int)	example: 0	No	How much additional space is provided in the hardware header in skbs in front of packets

Key (type)	Values	Mandatory	Description
ccwgroup.qeth.priority_queueing (string)	example: always queue 2	No	Which priority queueing algorithm is to be used
ccwgroup.qeth.route4 (string)	example: no	No	Whether the card has a routing functionality for ipv4. Not valid in layer2 mode.
ccwgroup.qeth.route6 (string)	example: no	No	Whether the card has a routing functionality for ipv6. Not valid in layer2 mode.
ccwgroup.qeth.state (string)	example: UP (LAN ONLINE)	Yes	The device's current state

The following properties describe ccwgroup devices where linux.driver is ctc.

Key (type)	Values	Mandatory	Description
ccwgroup.ctc.protocol (int)	example: 0	Yes	The protocol/method used by the connection
ccwgroup.ctc.type (string)	example: CTC/A	Yes	The device/connection type
ccwgroup.ctc.buffer (int)	example: 32768	No	The maximum buffer size of the connection

The following properties describe ccwgroup devices where linux.driver is lcs.

Key (type)	Values	Mandatory	Description
ccwgroup.lcs.portnumber (int)	example: 0	Yes	The port on the card that is used
ccwgroup.lcs.type (string)	example: OSA LCS card	Yes	The type of the card
ccwgroup.lcs.lancmd_timeout (int)	example: 5	Yes	The timeout value for LAN commands in seconds

The following properties describe ccwgroup devices where linux.driver is claw.

Key (type)	Values	Mandatory	Description
ccwgroup.claw.api_type (string)		Yes	Determines the packing algorithm for outgoing pakets (matching the remote peer)
	IP		Using the IP protocol
	PACKED		Using an enhanced packing algorithm
	TCPIP		Using the TCP/IP protocol
ccwgroup.claw.adapter_name (string)	example: RS1	Yes	The host name of the remote communication peer.

Key (type)	Values	Mandatory	Description
ccwgroup.claw.host_name (string)	example: LNX1	Yes	The host name of the local adapter.
ccwgroup.claw.read_buffer (int)	example: 4	Yes	The number of read buffers allocated
ccwgroup.claw.write_buffer (int)	example: 5	Yes	The number of write buffers allocated

drm namespace

The drm namespace is present for Direct Rendering Manager device objects. They represent a Direct Rendering Interface.

Key (type)	Values	Mandatory	Description
drm.dri_library (string)		Yes	Name of the dri (Direct Rendering Interface) library (e.g. i915).
drm.version (string)		Yes	The drm version (of the kernel module/diver).

ibmebus namespace

Devices on the BM Ebus are represented by device objects where info.subsystem equals ibmebus. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ibmebus.devspec (string)	example: /lhea@23c00100/ethernet@23e0010	Yes	The IBM Ebus device spec.
ibmebus.type (string)	example:	Yes	The type of IBM Ebus device

ide namespace

ATA and IDE drives are represented by device objects where info.subsystem equals ide. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ide.host (int)		Yes	Corresponds to ide_host.host_number of the ide_host device that is the parent of this device object
ide.channel (int)		Yes	Identifies the IDE channel of the host interface

ide_host namespace

The ide_host namespace is present for device objects where info.subsystem is set to ide_host. Such device objects represent IDE and ATA host adaptors for harddisks and optical drives as found in the majority of computer systems.

Key (type)	Values	Mandatory	Description
ide_host.number (int)		Yes	A unique number identifying the IDE host adaptor
ide_host.linux.sysfs_path (string)	example: /sys/devices/pci0000:00/0000:00:07.1/ide0	Yes (only on Linux)	Equals linux.sysfs_path

ieee1394 namespace

Device objects with info.subsystem set to ieee1394 represent IEEE 1394 devices. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ieee1394.specifier_id (int)		Yes	TODO

ieee1394_host namespace

Device objects with info.subsystem set to ieee1394_host represent IEEE 1394 host adaptors. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ieee1394_host.is_busmgr (bool)		Yes	TODO
ieee1394_host.is_irn (bool)		Yes	TODO
ieee1394_host.is_root (bool)		Yes	TODO
ieee1394_host.node_count (int)		Yes	TODO
ieee1394_host.nodes_active (int)		Yes	TODO

ieee1394_node namespace

Device objects with info.subsystem set to ieee1394_node represent IEEE 1394 nodes on a IEEE 1394 bus. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ieee1394_node.capabilities (int)		Yes	TODO
ieee1394_node.guid (int)		Yes	TODO
ieee1394_node.nodeid (int)		Yes	TODO

Key (type)	Values	Mandatory	Description
ieee1394_node.vendor (int)		Yes	TODO
ieee1394_node.vendor_id (int)		Yes	TODO

iucv namespace

Device objects with info.subsystem set to iucv are using the "Intra-User Comminication Vehicle" and are described by the following properties.

Key (type)	Values	Mandatory	Description
iucv.bus_id (string)	example: netiucv0	Yes	The device's bus id in sysfs

The following properties describe iucv devices where linux.driver is netiucv.

Key (type)	Values	Mandatory	Description
iucv.netiucv.user (string)	example: linux12	Yes	The guest name of the connection's target
iucv.netiucv.buffer (int)	example: 32768	Yes	The maximum buffer size of the connection

leds namespace

Device objects with info.subsystem set to leds are LED (light-emitting diode) devices.

Key (type)	Values	Mandatory	Description
leds.device_name (string)	example: iw1-phy0, tpacpi	Yes	The name of the related led device
leds.colour (string)	example: green, orange	No	The colour of the LED
leds.function (string)	example: radio, power, standby, batt	Yes	The function of the LED.
leds.num_levels (int)		Yes	The brightness levels supported by the LED device.

modem namespace

Serial device objectes that are known to be modems should also gain the modem capability in their info.capabilities list. Modem device objects must also be serial device objects.

Key (type)	Values	Mandatory	Description

Key (type)	Values	Mandatory	Description
modem.command_sets (string)	example: GSM-07.07, GSM-07.05	Yes	This property defines the command sets the modem device supports.
	IS-707-A	Yes	This modem supports the IS-707-A standard AT commands (commonly used by CDMA cellular devices). Implies V.250 support.
	GSM-07.07	Yes	This modem supports the GSM-07.07 standard AT commands (commonly used by GSM-based cellular devices). Implies V.250 support.
	GSM-07.05	Yes	This modem supports the GSM-07.05 standard AT commands (commonly used by GSM-based cellular devices). Implies V.250 support.
	ITU-V.250	Yes	This modem supports the ITU V.250 standard AT commands (also known as Hayes-compatible).

memstick namespace

Device objects with the capability **memstick** represent a Sony MemoryStick device. No namespace specific properties.

memstick_host namespace

Device objects with **info.subsystem** set to **memstick_host** represent Sony MemoryStick host adaptors. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
memstick_host.host (int)		Yes	A unique number identifying the Sony MemoryStick host adaptor

mmc namespace

Device objects with **info.subsystem** set to **mmc** represent MultiMediaCard or Secure Digital cards. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
mmc.cid (string)	example: 01504153303038424 13a1a8083003a9d	Yes	Card Identification Data register (unique for every card in existence)
mmc.csd (string)	example: 005d013213598067b6d9cfff1640002d	Yes	Card Specific Data register
mmc.scr (string)	example: 00a5000000410000	Only for SD cards	SD Card Register

Key (type)	Values	Mandatory	Description
mmc.rca (int)	example: 8083	Yes	Card bus address
mmc.oem (string)		Yes	Card OEM distributor
mmc.date (string)	example: 10/2003	Yes	Manufacturing date
mmc.serial (int)	example: 0x3a1a8083	Yes	Card serial number
mmc.hwrev (int)	example: 4	Yes	Hardware revision
mmc.fwrev (int)	example: 1	Yes	Firmware revision
mmc.type (string)	example: SDIO	No	Card type

mmc_host namespace

Device objects with info.subsystem set to mmc_host represent MultiMediaCard or Secure Digital host adaptors. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
mmc_host.host (int)		Yes	A unique number identifying the MMC/SD host adaptor
mmc_host.slot_name (string)	example: internal, external	No	A unique string identifying the slot name

pci namespace

This namespace contains properties for device objects representing functions on devices on a PCI bus. These properties are available exactly when info.subsystem equals pci.

Key (type)	Values	Mandatory	Description
pci.device_class (int)	example: 3	Yes	Device Class
pci.device_subclass (int)	example: 0	Yes	PCI Device Sub Class
pci.device_protocol (int)	example: 0	Yes	Device Protocol
pci.product_id (int)	example: 0x4c4d	Yes	Product ID
pci.vendor_id (int)	example: 0x1002	Yes	Vendor ID
pci.subsys_product_id (int)	example: 0x009e	Yes	Subsystem product id
pci.subsys_vendor_id (int)	example: 0x1028	Yes	Subsystem vendor id

Key (type)	Values	Mandatory	Description
pci.linux.sysfs_path (string)	example: /sys/devices/pci0000:00/0000:00:01/0000:01:00.0	Yes (only on Linux)	Equals linux.sysfs_path
pci.product (string)	Rage Mobility P/M AGP 2x	No	Name of the product per the PCI database
pci.vendor (string)	ATI Technologies Inc	No	Name of the vendor per the PCI database
pci.subsys_product (string)	Inspiron 7500	No	Name of the subsystem product per the PCI database
pci.subsys_vendor (string)	Dell Computer Corporation	No	Name of the subsystem vendor per the PCI database

(FIXME: Some key PCI information (bus, slot, port, function etc.) is missing here)

platform namespace

Devices that are built into the platform or present on busses that cannot be properly enumerated (e.g. ISA) are represented by device objects where info.subsystem equals platform. These kind of devices are commonly, somewhat incorrectly, called legacy devices.

Key (type)	Values	Mandatory	Description
platform.id (string)	example: serial	Yes	Device identification

pnp namespace

Device objects that represent Plug and Play (PnP) devices (e.g. System Board or PS/2 Port for PS/2-style Mice). For these devices info.subsystem is set to 'pnp'.

Key (type)	Values	Mandatory	Description
pnp.id (string)	example: PNP0a03 or SMCf010	Yes	This property contains the PnP ID of the device.
pnp.description (string)	example: 'ECP printer port'	No	Description from the pnp.ids file. Only available if: HAL was compiled with support for pnp.ids, if the file is available and if the ID was part of the file.
pnp.serial.irq (int)	example: 5	No	Only available if the PnP device is a serial device (as e.g. serial Wacom Tablet devices). Contains the prefered irq of the device.
pnp.serial.port (string)	example: 0x200	No	Only available if the PnP device is a serial device (as e.g. serial Wacom Tablet devices). contains info about the prefered serial port of the device.
pnp.serial.baud_base (int)	example: 38400	No	Only available if the PnP device is a serial device (as e.g. serial Wacom Tablet devices). Contains info about the needed baud_base to setup the device correctly.

ppdev namespace

The namespace for parallel port devices. No namespace specific properties.

ps3_system_bus namespace

Devices on the PlayStation 3 system bus are represented by device objects where **info.subsystem** equals **ps3_system_bus**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ps3_system_bus.id (string)	example: serial	Yes	Device identification

scsi namespace

SCSI devices are represented by device objects where **info.subsystem** equals **scsi**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
scsi.host (int)		Yes	Corresponds to **scsi_host.host** of the **scsi_host** device that is the parent of this device object
scsi.bus (int)		Yes	SCSI channel number
scsi.target (int)		Yes	SCSI identifier number
scsi.lun (int)		Yes	SCSI Logical Unit Number
scsi.type (string)	Example: disk	Yes	SCSI device type
	cdrom		This is a SCSI cdrom device.
	comm		This is a SCSI communication device.
	disk		This is a SCSI disk device.
	medium_changer		This is a SCSI media changer (e.g. for CD/Tape).
	printer		This is a SCSI printer.
	processor		This is a SCSI processor device.
	raid		This is a SCSI raid device.
	scanner		This is a SCSI scanner.
	tape		This is a SCSI tape device.
	unknown		The type of this SCSI device is unknwon.

scsi_host namespace

The **scsi_host** namespace is present for device objects where **info.subsystem** is set to **scsi_host**. Such device objects represent SCSI host adaptors for SCSI devices as found in some computer systems.

Key (type)	Values	Mandatory	Description
scsi_host.host (int)		Yes	A unique number identifying the SCSI host adaptor

sdio namespace

Device objects with **info.subsystem** set to **sdio** represent MultiMediaCard or Secure Digital cards with SDIO interface. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
sdio.rca (int)	example: 8083	Yes	Card bus address
sdio.card_id (int)	example: 1	Yes	SDIO Class for the interface
sdio.class_id (int)	example: 0x03	Yes	SDIO Class for the interface
sdio.product_id (int)	example: 0x046a	Yes	Product ID
sdio.vendor_id (int)	example: 0x039d	Yes	Vendor ID
sdio.product (string)	SQN1130 WiMAX Network Card	No	Name of the product per the SDIO database
sdio.vendor (string)	Sequans Communications	No	Name of the vendor per the SDIO database

serial namespace

Device objects that represent serial devices (e.g. /dev/ttyS* or /dev/ttyUSB*).

Key (type)	Values	Mandatory	Description
serial.originating_device (string)	example: **/org/freedesktop/Hal/devices/pnp_PNP0501**	Yes	UDI of the device the serial device is bound to.
serial.device (string)	example: /dev/ttyS0	Yes	The device node to access the OSS device.
serial.port (int)	example: 0	Yes	The port number of the device. For USB serial devices it's the port number of the serial device itself. For other serial devices, where is no device port info available, it's based on the number in **serial.device**.
serial.type (string)	example: platform, usb, unknown	Yes	This property defines the type of the serial device.

ssb namespace

Devices on the Sonics Silicon Backplane (SSB) bus are represented by device objects where **info.subsystem** equals **virtio**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
ssb.bus_id (string)	example:	Yes	The SSB Bus ID of the device

usb_device namespace

For device objects representing USB devices the property **info.subsystem** will be **usb_device**, and the following properties will be available. Note that the corresponding USB interfaces are represented by separate device objects as children.

Key (type)	Values	Mandatory	Description
usb_device.bus_number (int)	example: 1	Yes	The USB bus the device is attached to
usb_device.configuration_value (int)	example: 1	Yes	The current configuration the USB device is in; starting from 1
usb_device.configuration (int)	example: Bulk transfer configuration	No	Human-readable description of the current configuration the USB device is in
usb_device.num_configurations (int)	example: 1	Yes	Number of configurations this USB device can assume
usb_device.device_class (int)	example: 0	Yes	USB Device Class
usb_device.device_subclass (int)	example: 0	Yes	USB Device Sub Class
usb_device.device_protocol (int)	example: 0	Yes	USB Device Protocol
usb_device.is_self_powered (bool)	example: false	Yes	The device, in the current configuration, is self powered
usb_device.can_wake_up (bool)	example: true	Yes	The device, in the current configuration, can wake up
usb_device.max_power (int)	example: 98	Yes	Max power drain of device, in mA
usb_device.num_interfaces (int)	example: 1	Yes	Number of USB Interfaces in the current configuration
usb_device.num_ports (int)	example: 0	Yes	Number of ports on a hub. Zero for non-hubs
usb_device.port_number (int)	example: 1	Yes	The port number on the parent hub that the device is attached to, starting from 1
usb_device.speed (double)	examples: 1.5, 12.0, 480.0	Yes	Speed of device, in Mbit/s
usb_device.version (double)	examples: 1.0, 1.1, 2.0	Yes	USB version of device
usb_device.level_number (int)	example: 2	Yes	Depth in USB tree, where the virtual root hub is at depth 0

Key (type)	Values	Mandatory	Description
usb_device.linux.device_number (string)	example: 19	Yes (only on Linux)	USB Device Number as assigned by the Linux kernel
usb_device.linux.parent_number (string)	example: 19	Yes (only on Linux)	Device number of parent device as assigned by the Linux kernel
usb_device.linux.sysfs_path (string)	example: /sys/devices /pci0000:00/0000:00:07.2 /usb1/1-1/1-1.1	Yes (only on Linux)	Equals linux.sysfs_path
usb_device.product_id (int)	example: 0x3005	Yes	USB Product ID
usb_device.vendor_id (int)	example: 0x04b3	Yes	USB Vendor ID
usb_device.device_revision_bcd (int)	example: 0x0100	Yes	Device Revision Number encoded in BCD with two decimals
usb_device.serial (string)		No	A string uniquely identifying the instance of the device; ie. it will be different for two devices of the same type. Note that the serial number is broken on some USB devices.
usb_device.product (string)	example: IBM USB HUB KEYBOARD	No	Name of the product per the USB ID Database
usb_device.vendor (string)	example: IBM Corp.	No	Name of the vendor per the USB ID Database

usb namespace

Device objects that represent USB interfaces, ie. when info.subsystem assumes usb, are represented by the properties below. In addition all the usb_device.* properties from the parent USB device is available in this namespace but only with the usb prefix instead of usb_device.

Key (type)	Values	Mandatory	Description
usb.interface.class (int)	example: 0x03	Yes	USB Class for the interface
usb.interface.subclass (int)	example: 0x01	Yes	USB Sub Class for this interface
usb.interface.protocol (int)	example: 0x01	Yes	USB Protocol for the interface
usb.interface.description (int)	example: SyncML interface	No	Human-readable description for the interface provided by the device
usb.interface.number (int)	example: 1	Yes	Number of this interface, starting from zero
usb.linux.sysfs_path (string)	example: /sys/devices/pci0000:00 /0000:00:07.2/usb1/1-1/1-1.1/1-1.1:1.0	Yes (only on Linux)	Equals linux.sysfs_path

vio namespace

Devices on the IBM pSeries/iSeries 'vio' bus are represented by device objects where **info.subsystem** equals **vio**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
vio.id (string)	example: 1,10,3d	Yes	Device identification
vio.type (string)	example: viodasd, v-scsi, 1-lan	Yes	Device type.

virtio namespace

Devices on the VirtIO bus are represented by device objects where **info.subsystem** equals **virtio**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
virtio.id (string)	example: serial	Yes	Device identification

vmbus namespace

Virtual devices of the VMBus, which is part of the Hyper-V technologies (which is a hypervisor based virtualization solution) included in the Windows Server 2008, are represented by device objects where **info.subsystem** equals **vmbus**. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
vmbus.bus_id (string)	example: vmbus_0_0, vmbus_0_1	Yes	ID of the bus.
vmbus.bus_number (int)		Yes	The VMBus the device is attached to.
vmbus.device_id (string)	example: {5558a04b-62a7-48f5-ae07f1d51ae62faa}	Yes	ID of the device.
vmbus.device_number (int)		Yes	The number of the device on a bus.
vmbus.class_id (string)	example: {f8615163-df3e-46c5-913ff2d2f965ed0e}	Yes	Class identifier of the device.

xen namespace

Device objects representing virtual devices under the Xen Virtual Machine Monitor, such as frontend network or block devices, will have **info.subsystem** set to **block** and will export a number of properties in then **xen** namespace.

Key (type)	Values	Mandatory	Description

Key (type)	Values	Mandatory	Description
xen.bus_id (string)	example: vif-0	Yes	The XenBus ID of the device
xen.path (string)	example: device/vif/0	Yes	The XenBus path of the device
xen.type (string)	example: vif	Yes	The type of Xen device

Functional Properties

The section describe functional properties of device objects, that is, properties that are merged onto device objects representing addressable hardware. In most circumstances such properties stem from a kernel level driver attached to the device represented by the device object, however, as HAL can merge properties from anywhere, they may have been merged from device information files or callouts.

ac_adapter namespace

Device objects with the capability ac_adapter represent all the devices capable of powering the system from AC power

Key (type)	Values	Mandatory	Description
ac_adapter.present (bool)		Yes	The state of the adapter, i.e. whether it is providing power to the unit from mains power.

alsa namespace

Device objects with the capability alsa represent all the streams available through ALSA on a soundcard.

Key (type)	Values	Mandatory	Description
alsa.card (int)		Yes	Card number in system as registered by ALSA.
alsa.card_id (string)	Examples: I82801DBICH4, MP3	No	Textual description of the card.
alsa.device (int)		Yes	Device number assigned by ALSA for a current card.
alsa.device_file (string)		Yes	The device node to access the ALSA device.
alsa.device_id (string)	Examples: Intel 82801DB-ICH4 MIC2 ADC	No	Textual description of the specific device for a card
alsa.pcm_class (string)		No	The PCM class of the device.
	generic		A standard PCM sound device (SND_PCM_CLASS_GENERIC).
	multi		A multichannel device PCM sound device (SND_PCM_CLASS_MULTI) which e.g. contains a generic and a modem device.

Key (type)	Values	Mandatory	Description
	digitizer		A PCM digitizer device (SND_PCM_CLASS_DIGITIZER).
	modem		A PCM modem device (SND_PCM_CLASS_MODEM).
	unknown		The value is 'unknown' if the kernel provide no information about the PCM device class of the device (e.g. the file pcm_class is missing).
	none		The value is 'none' if this there is no PCM class for this device.
alsa.originating_device (string)		Yes	UDI of the device the ALSA device is bound to.
alsa.type (string)		Yes	The type of the stream.
	control		Stream is control device.
	capture		Stream is capture device.
	midi		Stream is MIDI device.
	playback		Stream is playback device.
	unknown		The type of the device is unknown.
	hw_specific		This is a hardware specific device (as e.g. from snd_fm801 for Fortemedia FM801 PCI Audio). The driver can use it freely for purposes that are not covered by standard ALSA API.
	timer		Stream is the global ALSA timer device. This means, the device is for all ALSA devices/cards.
	sequencer		Stream is the global ALSA sequencer device. This means, the device is for all ALSA devices/cards.
	unknown		Stream is unknown device.

battery namespace

Device objects with the capability `battery` represent all the devices having some battery (in many cases - rechargeable) inside.

Key (type)	Values	Mandatory	Description
battery.present (bool)		Yes	This is present as some smart batteries can have acpi/pmu entries, and be physically missing.
battery.type (string)		Yes	This property defines the type of the device holding the battery. This property is defined for the development simplicity - battery indicators can use it to find the proper iconic representation.

Key (type)	Values	Mandatory	Description
	pda		The device containing the battery is a personal digital assistant, e.g. a device that looks like a handheld computer to do specific tasks such as keeping notes or containing a personal database
	ups		A battery powered power supply that is guaranteed to provide power to a computer in the event of interruptions in the incoming electrical power. Most of the time this is an external device.
	primary		The battery is a primary power source for the system - an example are laptop batteries.
	mouse		The device containing the battery is a mouse.
	keyboard		The device containing the battery is a keyboard.
	keyboard_mouse		The device containing the battery is a combined mouse and keyboard.
	camera		The device containing the battery is a camera.
	usb		The device containing the battery is a generic usb device.
	unknown		The device containing the battery is not covered by other types.
battery.charge_level.unit (string)	Examples: mWh, percent	No	The physical unit used by the charge level properties (maximum and current). In many cases, this property is omitted - which indicates that the charge properties are measured in some unknown units. The units should never be mAh as this is not a measurement of charge.
battery.charge_level.design (int)		Yes	The maximum level of charge the device was designed for. Measured in "battery.charge_level.unit" units.
battery.charge_level.last_full (int)		Yes	The maximum level of charge the device could hold the last time it was full. Measured in "battery.charge_level.unit" units.
battery.charge_level.current (int)		Yes	The current level of charge which the device can is holding. Measured in "battery.charge_level.unit" units.
battery.charge_level.rate (int)		No	The discharge/charge rate measured in "battery.charge_level.unit" units per second.

Key (type)	Values	Mandatory	Description
battery.charge_level.warning (int)		No	Once the charge level of the battery drops below this value its state changes to 'warning'. Measured in "battery.charge_level.unit" units.
battery.charge_level.low (int)		No	Once the charge level of the battery drops below this value its state changes to 'low'. Measured in "battery.charge_level.unit" units.
battery.charge_level.granularity_1 (int)		No	Granularity value one of the battery measured in "battery.charge_level.unit" units .
battery.charge_level.granularity_2 (int)		No	Granularity value two of the battery measured in "battery.charge_level.unit" units.
battery.reporting.unit (string)	Examples: mWh, mAh, percent	No	The physical unit used by the charge level properties (maximum and current) as reported by the hardware. In many cases, this property is omitted - which indicates that the charge properties are measured in some unknown units.
battery.reporting.design (int)		Yes	The maximum level of charge the device was designed for, as reported by the hardware. Measured in "battery.reporting.unit" units.
battery.reporting.last_full (int)		No	The maximum level of charge the device could hold the last time it was full, as reported by the hardware. Measured in "battery.reporting.unit" units.
battery.reporting.current (int)		No	The current level of charge which the device is holding, as reported by the hardware. Measured in "battery.reporting.unit" units.
battery.reporting.rate (int)		No	The discharge/charge rate as reported by the hardware measured in "battery.reporting.unit" units per second.
battery.reporting.warning (int)		No	Once the hardware charge level of the battery drops below this value its state changes to 'warning'. Measured in "battery.reporting.unit" units.
battery.reporting.low (int)		No	Once the hardware charge level of the battery drops below this value its state changes to 'low'. Measured in "battery.reporting.unit" units.
battery.reporting.granularity_1 (int)		No	Hardware granularity value one of the battery measured in "battery.reporting.unit" units .

Key (type)	Values	Mandatory	Description
battery.reporting.granularity_2 (int)		No	Hardware granularity value two of the battery measured in "battery.reporting.unit" units.
battery.charge_level.capacity_state (string)	Examples: ok, critical	No	The capacity state of the battery.
battery.voltage.unit (string)	Examples: mV	No	The physical measurement unit used by the voltage properties (design and current).
battery.voltage.design (int)		Yes	The voltage level for which the battery is designed for. Measured in "battery.voltage.unit" units.
battery.voltage.current (int)		Yes	The voltage level currently emitted by the battery. Measured in "battery.voltage.unit" units.
battery.alarm.unit (string)	Examples: mWh, mAh	No	The physical measurement unit used by the alarm property.
battery.alarm.design (int)		No	Once the charge level of the battery drops below this value its state changes to 'alarm'. Measured in "battery.alarm.unit" units.
battery.remaining_time (int)		No	Remaining time, in seconds, that the battery can provide power (if discharging) or the time until charged (if charging). This is an estimate and may be imprecise. This key is not present for invalid data.
battery.remaining_time.calculate_per_time (bool)		No	If this property is true the battery.remaining_time becomes guessed from battery.charge_level.current and time.
battery.charge_level.percentage (int)		No	Charge, normalised to percent. This is useful if an application does not want to process the raw values and do all the extra checks on the result. This key is not present for invalid data.
battery.is_rechargeable (bool)		No	True if the battery unit is rechargeable, false if its is one-time (disposable after one usage).
battery.rechargeable.is_charging (bool)		Only if battery.is_rechargeable is TRUE	TRUE if, and only if, the battery is charging.
battery.rechargeable.is_discharging (bool)		Only if battery.is_rechargeable is TRUE	TRUE if, and only if, the battery is discharging.

Key (type)	Values	Mandatory	Description
battery.command_interface (string)		No	The abstract name allowing daemons and/or user-level apps to distinguish some groups of devices having similar programming interface. Introduced mostly for the daemons' coding simplicity.
battery.vendor (string)		No	Vendor of the battery.
battery.model (string)		No	Make of the battery.
battery.reporting.technology (string)	example: LION	No	The technology of the battery as reported by the hardware.
battery.technology (string)	lead-acid, lithium-ion, lithium-polymer, nickel-metal-hydride, unknown	No	The technology of the battery processed to a few standard types. This key is needed as the hardware often does not specify the description text for a battery, and so we have to calculate it from the output of battery.reporting.technology.
battery.serial (string)		No	A string uniquely identifying the instance of the battery; it will be different for two (otherwise) identical batteries.
battery.quirk.do_not_poll (bool)		No	True if HAL should not poll the battery, False or not available at all if HAL should show the default behavior.

biometric namespace

Device objects with the capability biometric represent a biometric device (e.g. fingerprint reader) . No namespace specific properties.

biometric.fingerprint_reader namespace

Device objects with the capabilities biometric.fingerprint_reader and biometric represent a biometric fingerprint reader.

Key (type)	Values	Mandatory	Description
biometric.fingerprint_reader.access_method (strlist)	example: libfprint	No	Indicates installed device driver libraries that can handle this device. These drivers can export information in biometric.fingerprint_reader.[access_method] sub-namespaces. Can also be used by libraries or programs providing extra device information to indicate the presence of this information in the appropriate sub-namespace.

button namespace

Device objects with the capability `button` represent the devices capable of providing a state to the system.

Key (type)	Values	Mandatory	Description
`button.type` (string)		No	The type of button
	lid		The switch on a laptop that senses whether the lid is open or closed
	power		The main power button on the computer.
	sleep		The sleep button on a computer capable of putting the computer into a suspend state
`button.has_state` (bool)		no	True if the button maintains state, e.g. can toggled on/off
`button.state.value` (bool)		Only when `button.has_state` is TRUE	State of the button, TRUE if it is enabled

Device objects with this capability may emit the following events.

Condition Name	Parameters	Example	Description
`ButtonPressed`	`button.type (string)`	sleep	Emitted when a button is pressed

camera namespace

Device objects with the capability `camera` represent digital still cameras that can be attached to a computer to exchange files. This does not include card readers for memory cards used for cameras. This capability can't, in general, be reliably probed from the hardware so the information needs to be merged from either device information files or callouts. Therefore this capability should be merged on the appropriate device object that represents the addressable piece of hardware that is the digital still camera; for USB devices this would be the device object representing the appropriate USB interface. The following properties are available:

Key (type)	Values	Mandatory	Description
`camera.access_method` (string)		Yes	This property defines how the device is accessed
	storage		The device is accessed as a Mass Storage device through a kernel driver. Application Developers should descent down the device object tree to find the device object of capability `storage` in order to access the device.
	user		The device is accessed from userspace through a userspace driver.

Key (type)	Values	Mandatory	Description
camera.libgphoto2.support (bool)		No	If true, the device is supported by a userspace driver from the libgphoto2 project.

input namespace

This namespace is concerned with human input devices such as keyboards, mice, pointing devices and game controllers. If a device object has the capability `input` then the following properties are available

Key (type)	Values	Mandatory	Description
input.device (string)		Yes	Special device file for recieving input events
input.x11_driver (string)	e.g. "evdev"	No	X11 input driver to use

input.joystick namespace

The input device is a joystick. No namespace specific properties.

input.keyboard namespace

The input device is a normal keyboard. No namespace specific properties.

input.keymap namespace

Device objects with the capability `input.keymap` provide facilities to remap keyboard buttons.

Key (type)	Values	Mandatory	Description
input.keymap.data (strlist)	e.g. "e017:brightnessup"	No	The scancode is represented in hex and the keycode name as as string. The keycode name is not case sensitive. On Linux, the keycode name should be the same constant as present in /usr/include/linux/input.h with the 'KEY_' prefix removed, e.g. 'KEY_SLEEP' -> 'sleep'. You can append as many `input.keymap.data` values as there are keys to remap.

input.keypad namespace

The input device have keypad keys. No namespace specific properties.

input.keys namespace

The input device have keys that can be pressed. No namespace specific properties.

input.mouse namespace

The input device is a mouse. No namespace specific properties.

input.switch namespace

The input device is a switch, e.g. it has buttons with state. No namespace specific properties.

input.tablet namespace

The input device is a tablet. No namespace specific properties.

input.xkb namespace

Device objects with the capability **input.keys** can provide information about their physical layout.

Key (type)	Values	Mandatory	Description
input.xkb.rules (string)	e.g. "base"	Yes	XKB rules file to use; 'base' is standard, but 'xorg' or 'xfree86' may be needed for backwards compatibility with very old versions of XKB data.
input.xkb.model (string)	e.g. "logicdp"	Yes	Physical keyboard model (e.g. Logitech Cordless Freedom Pro), as given to XKB.
input.xkb.layout (string)	e.g. "us"	Yes	Keyboard layout (as engraved on the keys).
input.xkb.variant (string)	e.g. "nodeadkeys"	No	Variant of the XKB layout (if any) to use.
input.xkb.options (strlist)	e.g. "ctrl:nocaps"	No	Options to be provided to XKB.

keyboard_backlight namespace

Device objects with the capability **keyboard_backlight** represent all the devices capable handling the keyboard backlight. (Note: normally those devices have also the **leds** capability.)

Key (type)	Values	Mandatory	Description
keyboard_backlight.num_levels (bool)		Yes	The brightness levels supported by the LED device.

killswitch namespace

Device objects with the capability `killswitch` represent switches to turn a radio on and off. See also the section called "org.freedesktop.Hal.Device.KillSwitch interface".

Key (type)	Values	Mandatory	Description
`killswitch.type` (string)		Yes	Type of the kill switch
	wlan		Kill switch is for turning Wireless LAN (WLAN) networking on/off
	bluetooth		Kill switch is for turning Bluetooth on/off
	wwan		Kill switch is for turning Wireless WAN (WWAN) networking on/off
`killswitch.state` (int)		No	Current state of the killswitch (as reported by the kernel)
	0		Radio output is blocked
	1		Radio output is allowed/enabled
	2		Radio output is (hard) blocked, non-overrideable via SetPower()
`killswitch.name` (string)		No	Name of the kill switch (as reported by the kernel).
`killswitch.access_method` (string)		Yes	How HAL should program the switch

laptop_panel namespace

Device objects with the capability `laptop_panel` represent devices capable of changing the brightness of the display.

Key (type)	Values	Mandatory	Description
`laptop_panel.num_levels` (int)		Yes	The brightness levels supported by the adaptor.
`laptop_panel.access_method` (string)		Yes	The access method to use in scripts, e.g. pmu, toshiba, ibm, sony.
`laptop_panel.brightness_in_hardware` (bool)		No	On some laptops, the brightness control is all done in hardware but the hardware also synthesizes keypresses when the brightness is changed. If this key is set true, then any power manager software should not attempt to set any new values on brightness keypress, as it may cause the panel to flash uncontrollably.

The following methods exist on the interface `org.freedesktop.Hal.Device.LaptopPanel`.

Method (parameter types)	Parameters	Mandatory	Description
`SetBrightness` (integer)	The hardware brightness state, which should be between 0 and	No	This method adjusts the brightness on an laptop screen. The values are returned as hardware values rather than percentages as we

Method (parameter types)	Parameters	Mandatory	Description
	laptop_panel.num_levels - 1.		cannot easily to floating point rounding in shell code and therefore use the raw values to prevent integer rounding errors.
integer GetBrightness (void)	Returns the hardware brightness state, which should be between 0 and laptop_panel.num_levels - 1.	No	This method gets the hardware brightness of the laptop screen, which we may need to do fairly regually on hardware that changes the values in hardware without a software event.

light_sensor namespace

Device objects with the capability sensor represent light sensors in the system.

Key (type)	Values	Mandatory	Description
light_sensor.sensor_locations (strlist)		Yes	The locations of the sensors
light_sensor.num_sensors (int)		Yes	Number of physical sensors
light_sensor.num_levels (int)		Yes	The number of levels of the sensors

net namespace

This namespace is used to describe networking devices and their capabilities.Such device objects will have the capability net and they will export the properties below. This namespace only describe the generic aspect of networking devices; specific networking technologies such as IEEE 802.3, IEEE 802.11 and Bluetooth have separate namespaces.

Key (type)	Values	Mandatory	Description
net.address (string)		Yes	Hardware address as a string. Is hardware dependant
net.arp_proto_hw_id (string)		Yes	ARP protocol hardware identifier
net.interface (string)		Yes	Name of the interface; may change if an interface is renamed
net.interface_up (bool)		No	Whether the interface is up
net.linux.ifindex (string)		Yes (only on Linux)	Index of the interface
net.originating_device (string)		Yes	UDI of the device the network device is bound to.
net.media (string)	example: Ethernet	Yes	Textual description of the networking media

net.80203 namespace

Ethernet networking devices is described in this namespace for device objects with the capability net.80203. Note that device objects

can only have the net.80203 capability if they already have the capability net.

Key (type)	Values	Mandatory	Description
net.80203.link (bool)		Only if the net.80203 capability is set and net.interface_up is TRUE.	True if the ethernet adaptor is connected to a another transceiver. NOTE: property not implemented yet.
net.80203.rate (uint64)	example: 100000000	Only if the net.80203 capability is set and net.80203.link is TRUE.	Bandwidth of connection, in bits/s. NOTE: property not implemented yet.
net.80203.mac_address (uint64)	example: 0x0010605d8ef4	Only if the net.80203 is set	48-bit address

net.80211 namespace

Wireless ethernet networking devices is described in this namespace for device objects with the capability net.80211. Note that device objects can only have the net.80211 capability if they already have the capability net.

Key (type)	Values	Mandatory	Description
net.80211.mac_address (uint64)	example: 0x0010605d8ef4	Only if the net.80211 capability is set	48-bit address

net.80211control namespace

Control devices for Wireless ethernet networking devices are described in this namespace for device objects with the capability net.80211control. Note that device objects can only have the net.80211control capability if they already have the capability net. Warning: You should know what you do if you touch this devices. They are not always stable and can cause (kernel) crashes (on linux).

net.bluetooth namespace

Bluetooth ethernet networking devices is described in this namespace for device objects with the capability net.bluetooth. Note that device objects can only have the net.bluetooth capability if they already have the capability net.

Key (type)	Values	Mandatory	Description
net.bluetooth.mac_address (uint64)	example: 0x0010605d8ef4	Only if the net.bluetooth capability is set	48-bit address
net.bluetooth.name (string)	example: Network Access Point Service	Only if the net.bluetooth capability is set and Bluez is being used.	Displayable Name for network connection

Key (type)	Values	Mandatory	Description
net.bluetooth.uuid (string)	example: 00001116-0000-1000-8000-00805f9b34fb	Only if the net.bluetooth capability is set and Bluez is being used.	Universal Unique Identifier for network connection

net.bridge namespace

Bridge ethernet networking devices is described in this namespace for device objects with the capability net.bridge. Note that device objects can only have the net.bridge capability if they already have the capability net.

Key (type)	Values	Mandatory	Description
net.bridge.mac_address (uint64)	example: 0x0010605d8ef4	Only if the net.bridge capability is set	48-bit address

net.irda namespace

IrDA (Infrared Data Association) Networking devices are described in this namespace for device objects with the capability net.irda. Note that device objects can only have the net.irda capability if they already have the capability net.

net.loopback namespace

Loopback networking devices are described in this namespace for device objects with the capability net.loopback. Note that device objects can only have the net.loopback capability if they already have the capability net. No namespace specific properties.

obex namespace

Device objects with the capability obex represent devices that have implemented OBject EXchange protocol for communication. Typically such devices are mobile phones and the protocol is used to transfer files, synchronize data and manage mobile phone configuration. No namespace specific properties.

Key (type)	Values	Mandatory	Description
obex.type (sting)	example: pcsuite, syncml	No	
	pcsuite	No	Device is meant to be used with Nokia PC Suite application. Standardized OBEX file transfer is supported and possibly proprietary extensions.
	syncml	No	Device supports SyncML over OBEX protocol.
	syncml-sync	No	Device supports SyncML over OBEX protocol for data synchronization.
	syncml-dm	No	Device supports SyncML over OBEX protocol for data management.

of_platform namespace

Devices on the virtual 'of_platform' bus are represented by device objects where `info.subsystem` equals `of_platform`. The following properties are available for such device objects.

Key (type)	Values	Mandatory	Description
`of_platform.id` (string)	example: f0003000.ethernet	Yes	Device identification

oss namespace

Device objects with the capability `oss` represent all the streams available through OSS on a soundcard. OSS devices could be emulated by ALSA.

Key (type)	Values	Mandatory	Description
`oss.card` (int)		Yes	Card number in system as registered by OSS (and/or ALSA).
`oss.card_id` (string)	Examples: `I82801DBICH4`, `MP3`	No	Textual description of the card.
`oss.device` (int)		Yes	Device number assigned by OSS/ALSA for a current card.
`oss.device_file` (string)		Yes	The device node to access the OSS device.
`oss.device_id` (string)	Examples: `Intel 82801DB-ICH4` `MIC2 ADC`	No	Textual description of the specific device for a card
`oss.originating_device` (string)		Yes	UDI of the device the OSS device is bound to.
`oss.type` (string)		Yes	The type of the stream.
	`mixer`		Stream is control/mixer device.
	`pcm`		Stream is PCM device.
	`midi`		Stream is MIDI device.
	`sequencer`		Stream is a global OSS sequencer device. This means, the device is for all OSS devices/cards.
	`unknown`		Stream is unknown device.

pcmcia_socket namespace

Device objects with the capability `pcmcia_socket` represent bridge devices (the actual bus of the device may differ) that PCMCIA cards can be attached to. The following properties are available.

Key (type)	Values	Mandatory	Description

Key (type)	Values	Mandatory	Description
pcmcia_socket.number (int)		Yes	PCMCIA socket number, starting from zero

pda namespace

Device objects with the capability pda represent Personal Digital Assistant (PDA) devices.

Key (type)	Values	Mandatory	Description
pda.platform (string)	e.g. palm or pocketpc	Yes	The type of the PDA platform
pda.palm.hotsync_interface (string)		Yes	Path to the Palm hotsync interface e.g. a linux device file (e.g. USB) or a serial device.
pda.pocketpc.hotsync_interface (string)		Yes	Path to the Pocket PC (e.g. HP iPAQ) hotsync interface e.g. a linux device file (e.g. USB) or a serial device.

power_management namespace

Keys with the prefix power_management provide information about power management supported by the system.

Key (type)	Values	Mandatory	Description
power_management.type (string)	Examples: apm, acpi, pmu	Yes	The power management subsystem used on the computer.
power_management.can_suspend (bool)		Yes	If suspend support is compiled into the kernel. NB. This may not mean the machine is able to suspend successfully.
power_management.can_suspend_hybrid (bool)		Yes	If the system is capable of hybrid suspend.
power_management.can_hibernate (bool)		Yes	If hibernation support is compiled into the kernel. NB. This may not mean the machine is able to hibernate successfully.
power_management.is_powersave_set (bool)		Yes	Is the last value passed to the SetPowerSave method.
power_management.quirk.s3_bios (bool)		No	Use the S3_BIOS kernel command for suspend.
power_management.quirk.s3_mode (bool)		No	Use the S3_MODE kernel command for suspend.
power_management.quirk.dpms_suspend (bool)		No	Suspend the video card via DPMS on suspend.
power_management.quirk.vga_mode_3 (bool)		No	Reset the VGA text mode to mode 3 on resume.
power_management.quirk.dpms_on (bool)		No	Reactivate the screen with DPMS on resume.
power_management.quirk.vbe_post (bool)		No	Run the VGA BIOS Power On Self Test (POST) on resume.

Key (type)	Values	Mandatory	Description
power_management.quirk.vbestate_restore (bool)		No	Save the VGA BIOS state before suspend, and restore it on resume.
power_management.quirk.vbemode_restore (bool)		No	Save the VGA BIOS mode before suspend, and restore it on resume.
power_management.quirk.pci_save (bool)		No	saving the PCI config space of the graphic card before suspend, restoring it after
power_management.quirk.radeon_off (bool)		No	Turn off the Radeon DAC off before suspend.
power_management.quirk.reset_brightness (bool)		No	Reset the brightness state after resume.
power_management.quirk.no_fb (bool)		No	True if the machine can only suspend when not using framebuffer.
power_management.quirk.none (bool)		No	No quirks are necessary for suspend or resume.

portable_audio_player namespace

Device objects with the capability **portable_audio_player** represent portable audio players that can be attached to a computer to exchange files. They can also playback audio. Sometimes they can also record audio. This capability can't, in general, be reliably probed from the hardware so the information needs to be merged from either device information files or callouts. Therefore this capability should be merged on the appropriate device object that represents the addressable piece of hardware that is the portable music player; for USB devices this would be the device object representing the appropriate USB interface. The following properties are available:

Key (type)	Values	Mandatory	Description
portable_audio_player.access_method.protocols (strlist)	example: storage ipod mtp pde iriver karma	Yes	Indicates transfer protocols that this device can speak. **storage** indicates USB Mass Storage (UMS) is an access protocol. **ipod** indicates UMS plus an iTunes-style database. **mtp** indicates a device using Microsoft's Media Transfer Protocol. Arbitrary values for newer or obscure protocols are allowed but entities providing this information should try to ensure that they are not duplicating protocols under a different name.
portable_audio_player.access_method.drivers (strlist)	example: libgpod, libmtp, libnjb, libifp, libkarma	No	Indicates installed device driver libraries that can handle this device. These drivers can export information in **portable_audio_player.[drivername]** sub-namespaces. Can also be used by libraries or

Key (type)	Values	Mandatory	Description
			programs providing extra device information to indicate the presence of this information in the appropriate sub-namespace.
portable_audio_player.[drivername].protocol (strlist)	example: mtp	If portable_audio_player.drivers is set	This entry is required for drivers listed in portable_audio_player.access_method.drivers. Indicates which protocol in portable_audio_player.access_method.protocols a particular driver will use. If the driver is providing information only, this should be set to information.
portable_audio_player.output_formats (strlist)	example: audio/mpeg audio/x-ms-wma	Yes	A string list of MIME-types representing the kind of audio formats that the device can play back.
portable_audio_player.input_formats (strlist)	example: audio/x-wav	Yes	A string list of MIME-types representing the kind of audio formats that the device can record. If empty, it means that the device is not capable of recording.
portable_audio_player.folder_depth (int)	example: 1 (If the device only supports one sub-folder)	No	If portable_audio_player.access_method.protocols contains "storage", this tells applications exactly how deep of directory hierarchies files should be placed in. If all files are put in a sub-folder (with the audio_folders property), only the depth within that sub-folder should be entered here. If the device does not have a limit, do not set this property.
portable_audio_player.audio_folders (strlist)	example: music/ voice/ linein/	No	If portable_audio_player.access_method.protocols contains "storage", this may contain a string list of folders in which music can be found. Paths are relative to the mount point of the device. If there is one or more entry in this property, the first one is where files will be written to by applications. Do not enter a folder and a parent of that folder. If the device places files in its root directory, then do not set this property.

Key (type)	Values	Mandatory	Description
portable_audio_player.playlist_format (strlist)	example: audio/x-mpegurl audio/x-somethingelse	No	A string list of the MIME-type of the playlist formats accepted by this device. Leave blank if none.
portable_audio_player.playlist_path (string)	examples: playlists/%File or Playlist.m3u	No	Set to the path to which playlists should be written. Leave blank if playlist files are not supported. If the device supports a single playlist with a specific name/path, set this to the path relative to the mount point that it should be saved to. If it supports multiple playlists, use the %File variable as needed. Applications are responsible for substituting %File with the desired playlist file name, noting that it's use in this string is optional.
portable_audio_player.storage_device (string)	examples: a udi	No	Set to the udi of the portable_audio_player device if portable_audio_player.access_method.protocols contains storage for a USB Mass Storage (UMS) music player. Mandatory for storage.

printer namespace

Device objects with the capability printer represent printers. The following properties are available.

Key (type)	Values	Mandatory	Description
printer.device (string)		Yes	Special device file to interact with the printer device.
printer.vendor (string)		Yes	Name of the device vendor
printer.product (string)		Yes	Name of the product.
printer.serial (string)		Yes	A string uniquely identifying the instance of the device; ie. it will be different for two devices of the same type. Note that the serial number is broken on some USB devices.
printer.description (string)		No	Description for the device.

Key (type)	Values	Mandatory	Description
printer.commandset (strlist)		No	List of supported commands / printer languages.

processor namespace

Device objects with the capability processor represent CPU's in the system.

Key (type)	Values	Mandatory	Description
processor.number (int)		Yes	The internal processor number in the system, starting from zero
processor.can_throttle (bool)		No	Whether the processor supports throttling to decrease it's own clock speed
processor.maximum_speed (long)	example: 2200	No	The maximum speed of the processor in units of MHz

scanner namespace

Device objects with the capability scanner represent image scanners. This capability should be merged on the appropriate device object that represents the addressable piece of hardware that is the digital still camera; for USB devices this would be the device object representing the appropriate USB interface. The following properties are available:

Key (type)	Values	Mandatory	Description
scanner.access_method (string)		Yes	This property defines how the device is accessed
	proprietary		The device is accessed from userspace through a userspace driver such as SANE.

smart_card_reader namespace

Device objects with the capability smart_card_reader represent a smart card device/systems (e.g. smart card reader) . No namespace specific properties.

smart_card_reader.card_reader namespace

Device objects with the capabilities smart_card_reader and smart_card_reader.card_reader represent a smart card reader. No namespace specific properties.

smart_card_reader.crypto_token namespace

Device objects with the capabilities smart_card_reader and smart_card_reader.crypto_token represent a smart token, a device where the

smart card and the smart card reader are in one device. No namespace specific properties.

storage namespace

This namespace is used to describe storage devices and their capabilities. Such device objects will have the capability `storage` and they will export the properties below. Note that device objects can only have the `storage` capability if they already got capability `block` and the property `block.is_volume` set to FALSE. One significant between the `storage` and `block` namespace is that the properties exported in the `storage` represents constant vital product information, whereas the properties in the `block` namespace represent variable system-dependent information.

Key (type)	Values	Mandatory	Description
`storage.bus` (string)		Yes	Interface the storage device is attached to
	cciss		HP Smart Array CCISS interface
	ccw		IBM s390/s390x ccw interface
	ide		IDE or ATA interface
	ieee1394		IEEE 1394 interface
	linux_raid		Linux MD (multi disk) RAID device
	memstick		Sony MemoryStick device/interface
	mmc		MultiMediaCard (MMC) interface
	pci		PCI interface
	pcmcia		PCMCIA interface
	platform		Legacy device that is part of the platform
	sata		SATA interface
	scsi		SCSI interface
	usb		USB interface
	vio		IBM pSeries/iSeries Vio interface
`storage.drive_type` (string)		Yes	The type of the drive. Note that it may not be possible to probe for some of these properties so in some cases memory card readers may appear as harddisks. Device information files can be used to override this value.
	disk		The device is a harddisk
	cdrom		The device is an optical drive. The device object will also have the capability `storage.cdrom` in this case.
	floppy		The device is a floppy disk drive

Key (type)	Values	Mandatory	Description
	tape		The device is a tape drive
	compact_flash		The device is a card reader for Compact Flash memory cards
	memory_stick		The device is a card reader for MemoryStick memory cards
	smart_media		The device is a card reader for SmartMedia memory cards
	sd_mmc		The device is a card reader for SecureDigital/MultiMediaCard memory cards
storage.removable (bool)		Yes	Media can be removed from the storage device
storage.removable.media_available (bool)		Yes	true, if and only if, media have been detected in storage device
storage.removable.media_size (uint64)		Yes	Size of media in storage device. Available only if media have been detected in storage device.
storage.removable.support_async_notification (bool)		Yes	Whether the drive reports asynchronous notification for media change.
storage.partitioning_scheme (string)		Only when media is inserted and is partitioned	The partitioning scheme of the media.
	mbr		Master Boot Record partitioning scheme used in most PC's
	gpt		GUID Partitioning Table as defined by UEFI
	apm		Apple Partition Map, used in non-Intel Apple computers
storage.size (uint64)		No	size in bytes of the storage device - only meaningful if storage.removable is FALSE
storage.requires_eject (bool)		Yes	The eject ioctl is required to properly eject the media
storage.hotpluggable (bool)		Yes	The storage device can be removed while the system is running
storage.media_check_enabled (bool)		Yes	If this property is set to FALSE then HAL will not continuosly poll for media changes.
storage.automount_enabled_hint (bool)		Yes	This property is a hint to desktop file managers that they shouldn't automount volumes of the storage device when they appear.
storage.no_partitions_hint (bool)		Yes	This property is a hint to programs that maintain the /etc/fstab file to signal, when TRUE, that the storage drive (such as floppy or optical drives) is used for media with no partition

Key (type)	Values	Mandatory	Description
			table so an entry can be added ahead of media insertion time. Note that this is only a hint; media may be inserted that has partition tables that the kernel may respect. Conversely, when this is FALSE media without partition tables may be inserted (an example is a Compact Flash card; this media is normally formatted with a PC style partition table and a single FAT partition. However, it may be formatted with just a single FAT partition and no partition table).
`storage.originating_device` (string)		Yes	This contains the UDI of the device object representing the device or blank if there is no such device.
`storage.model` (string)		Yes	The name of the drive
`storage.vendor` (string)		Yes	The vendor of the drive
`storage.serial` (string)		No	The serial number of the drive
`storage.firmware_revision` (string)		No	The revision of the firmware of the drive
`storage.icon.drive` (string)		No	Name of icon to use for displaying the drive. The name must comply with freedesktop.org icon-theme specification and must not be an absolute path. This property exists such that e.g. OEM's can install icons in `/usr/share/icons/hicolor` a device information file matching their device.
`storage.icon.volume` (string)		No	Name of icon to use for displaying volumes from the drive. The name must comply with freedesktop.org icon-theme specification and must not be an absolute path. This property exists such that e.g. OEM's can install icons in `/usr/share/icons/hicolor` a device information file matching their device.

storage.cdrom namespace

This namespace is used to describe optical storage drives and their capabilities.Such device objects will have the capability `storage.cdrom` and they will export the properties below. Note that device objects can only have the `storage.cdrom` capability if they already got the capability `storage`.

Key (type)	Values	Mandatory	Description
`storage.cdrom.cdr` (bool)		Yes	TRUE when the optical drive can write CD-R discs
`storage.cdrom.cdrw` (bool)		Yes	TRUE when the optical drive can blank and write to CD-RW discs
`storage.cdrom.dvd` (bool)		Yes	TRUE when the optical drive can read DVD-ROM discs
`storage.cdrom.dvdr` (bool)		Yes	TRUE when the optical drive can write to DVD-R discs

Key (type)	Values	Mandatory	Description
storage.cdrom.dvdrw (bool)		Yes	TRUE when the optical drive can blank and write to DVD-RW discs
storage.cdrom.dvdrdl (bool)		Yes	TRUE when the optical drive can write to DVD-R Dual-Layer discs
storage.cdrom.dvdram (bool)		Yes	TRUE when the optical drive can write to DVD-RAM discs
storage.cdrom.dvdplusr (bool)		Yes	TRUE when the optical drive can write to DVD+R discs
storage.cdrom.dvdplusrw (bool)		Yes	TRUE when the optical drive can blank and write to DVD+RW discs
storage.cdrom.dvdplusrwdl (bool)		Yes	TRUE when the optical drive can blank and write to DVD+RW Dual-Layer discs
storage.cdrom.dvdplusrdl (bool)		Yes	TRUE when the optical drive can write to DVD+R Dual-Layer discs
storage.cdrom.bd (bool)		Yes	TRUE when the optical drive can read Blu-ray ROM discs
storage.cdrom.bdr (bool)		Yes	TRUE when the optical drive can write to Blu-ray Recordable discs
storage.cdrom.bdre (bool)		Yes	TRUE when the optical drive can write to Blu-ray Rewritable discs
storage.cdrom.hddvd (bool)		Yes	TRUE when the optical drive can read Read-only HD DVD discs
storage.cdrom.hddvdr (bool)		Yes	TRUE when the optical drive can write to Write-once HD DVD discs
storage.cdrom.hddvdrw (bool)		Yes	TRUE when the optical drive can write to Rewritable HD DVD discs
storage.cdrom.mrw (bool)		Yes	TRUE when the optical drive can read MRW (Mount Rainier Rewrite) discs
storage.cdrom.mrw_w (bool)		Yes	TRUE when the optical drive can write MRW (Mount Rainier Rewrite) discs
storage.cdrom.mo (bool)		No	TRUE when the optical drive is a MO (Magneto Optical) device.
storage.cdrom.support_multisession (bool)		Yes	TRUE if the drive can read multisession discs
storage.cdrom.support_media_changed (bool)		Yes	TRUE if the drive can generate media changed events
storage.cdrom.read_speed (int)		Yes	The maximum reading speed, in kb/s
storage.cdrom.write_speed (int)		Yes	The maximum writing speed, in kb/s
storage.cdrom.write_speeds (strlist)		No	By the device supported write speeds in kb/s

storage.linux_raid namespace

This namespace is used to describe logical Software RAID devices under Linux using the md driver. By and large, all the same properties under the storage name space applies except that storage.serial is set to the UUID of the RAID set, storage.firmware_version is set to the version of the md driver and the value of storage.hotpluggable is taken from the enclosing drive of the first RAID component encountered. In addition, the following properties are available.

Key (type)	Values	Mandatory	Description
storage.linux_raid.level (string)		Yes	the RAID level of the device as reported by the kernel (linear, raid0, raid1, raid4, raid5, raid6, raid10)

Key (type)	Values	Mandatory	Description
storage.linux_raid.sysfs_path (string)		Yes	sysfs path of device, e.g. /sys/block/md0
storage.linux_raid.num_components (int)		Yes	Number of components in the RAID array
storage.linux_raid.num_components_active (int)		Yes	Number of active components in the RAID array. If less than storage.linux_raid.num_components it means that the RAID array is running in degraded mode.
storage.linux_raid.components (strlist)		Yes	UDI's of the volumes constituting the array.
storage.linux_raid.is_syncing (bool)		Yes	TRUE if, and only if, the array is currently syncing
storage.linux_raid.sync.action (string)		only if .is_syncing is TRUE	The syncing mechanism as reported by the kernel (idle, resync, check, repair, recover)
storage.linux_raid.sync.progress (double)		only if .is_syncing is TRUE	Number between 0 and 1 representing progress of the sync operation. This is updated regulary when syncing is happening.
storage.linux_raid.sync.speed (uint64)		only if .is_syncing is TRUE	Speed of the sync operation, in kB/s. This is updated regulary when syncing is happening.

system namespace

This namespace is found on the toplevel "Computer" device, and represents information about the system and the currently running kernel.

Key (type)	Values	Mandatory	Description
system.kernel.name (string)	example: Linux	No	The name of the kernel, usually the equivalent of uname -s.
system.kernel.version (string)	example: 2.6.5-7.104-default	No	The version of the currently running kernel. Usually the equivalent of uname -r.
system.kernel.version.major (int)	example: 2	No	The major version number of the running kernel.
system.kernel.version.minor (int)	example: 6	No	The minor version number of the running kernel.
system.kernel.version.micro (int)	example: 28	No	The micro version number of the running kernel.
system.kernel.machine (string)	example: i686	No	The "machine hardware name" of the currently running kernel. Usually the equivalent of uname -m.
system.formfactor (string)	example: laptop, desktop, server, unknown	Yes	The formfactor of the system. Usually the equivalent of system.chassis.type or set from information about ACPI/APM/PMU properties.

Key (type)	Values	Mandatory	Description
system.hardware.vendor (string)		No	The name of the manufacturer of the machine.
system.hardware.product (string)		No	The product name of the machine.
system.hardware.version (string)		No	The version of the machine.
system.hardware.serial (string)		No	The serial number of the machine.
system.hardware.uuid (string)		No	The unique ID of the machine.
system.hardware.primary_video.vendor (int)		No	The PCI vendor ID of the primary graphics card in the system.
system.hardware.primary_video.product (int)		No	The PCI device ID of the primary graphics card in the system.
system.firmware.vendor (string)		No	The firmware vendor.
system.firmware.version (string)		No	The firmware version.
system.firmware.release_date (string)		No	The release date of the firmware.
system.chassis.manufacturer (string)		No	The manufacturer of the chassis.
system.chassis.type (string)		No	The chassis type of the machine.
system.board.vendor (string)		No	The name of the manufacturer of the base board.
system.board.product (string)		No	The product name of the base board.
system.board.version (string)		No	The version of the base board.
system.board.serial (string)		No	The serial number of the base board.

tape namespace

Device objects with the capability **tape** represent tape devices.

Key (type)	Values	Mandatory	Description
tape.major (int)	example: 254	Yes	The device's major number
tape.minor (int)	example: 0	Yes	The device's minor number

video4linux namespace

Device objects with the capability **video4linux** represent Video4Linux devices.

Key (type)	Values	Mandatory	Description
video4linux.device (string)	Example: /dev/video0	Yes	The device node to access the Video4Linux device.

Key (type)	Values	Mandatory	Description
video4linux.version (string)	Example: 2	Yes	The highest Video4Linux API version supported by the device.

video4linux.audio namespace

The video4linux device has audio inputs or outputs. No namespace specific properties.

video4linux.radio namespace

The video4linux device is a radio device. No namespace specific properties.

video4linux.tuner namespace

The video4linux device has some sort of tuner or modulator to receive or emit RF-modulated video signals. No namespace specific properties.

video4linux.video_capture namespace

The video4linux device can capture video. No namespace specific properties.

video4linux.video_output namespace

The video4linux device can output video. No namespace specific properties.

video4linux.video_overlay namespace

The video4linux device can overlay video. No namespace specific properties.

volume namespace

This namespace is for device objects that represent storage devices with a filesystem that can be mounted. Such device objects will have the capability **volume** and they will export the properties below. Note that device objects can only have the **volume** capability if they already have the capability **block** and the property **block.is_volume** set to TRUE.

Key (type)	Values	Mandatory	Description
volume.ignore (bool)		Yes	This is a hint to software higher in the stack that this volume should be ignored. If TRUE, the volume should be invisible in the UI and

Key (type)	Values	Mandatory	Description
			mount wrappers should refuse to mount it on behalf on an unprivileged user. This is useful for hiding e.g. firmware partitions (e.g. bootstrap on Mac's) and OS reinstall partitions on e.g. OEM systems.
volume.is_mounted (bool)		Yes	This property is TRUE if and only if the volume is mounted
volume.is_mounted_read_only (bool)		Yes	This property is TRUE if and only if the volume is mounted and the volume's file-system is read-only.
volume.mount_point (string)	example: /media/compact_flash1	Yes (is blank only when volume.is_mounted is FALSE)	A fully qualified path to the mount point of the volume
volume.fsusage (string)	example: filesystem	Yes	This property specifies the expected usage of the volume
	filesystem		The volume is a mountable filesystem
	partitiontable		The volume contains a partitiontable.
	raid		The volume is a member of a raid set and not mountable
	other		The volume is not mountable like a swap partition
	unused		The volume is marked a unused or free
volume.fstype (string)	examples: ext3, vfat	Yes (is blank if the type is unknown)	The specific type of either the file system or what the volume is used for, cf. volume.fsusage
volume.mount.valid_options (string list)	examples: ro, sync, noexec	Yes, if the org.freedesktop.Hal.Device.Volume interface is defined.	List of the allowed (valid) mount options for the defined fstype.
volume.unmount.valid_options (string list)	examples: lazy	No (only available if the org.freedesktop.Hal.Device.Volume interface is defined)	List of the allowed (valid) unmount options for the defined fstype.
volume.fstype.alternative (string list)	examples: ntfs-3g, ntfs-fuse	No (only if there is a alternative mount handler)	The allowed (and installed) alternative mount handler/filesystem(s).
volume.fstype.alternative.preferred (string)	examples: ntfs-3g	No	The preferred alternative mount handler/filesystem. Allows e.g. a distributor do define a preferred of the alternative handlers. If not set, the default filesystem

Key (type)	Values	Mandatory	Description
			(defined by volume.fstype) should be used (or let the fstype parameter of Mount() empty), otherwise it must be the name of one of the alternative handlers, as in volume.fstype.alternative.
volume.mount.[alternative].valid_options (string list)	examples: locale=, uid=	Yes, if volume.fstype.alternative is set and if the org.freedesktop.Hal.Device.Volume interface is defined.	List of the allowed (valid) mount options for the defined alternative filesystem handler. Replace [alternative] in the property string with the name of the alternative filesystem handler. For e.g. ntfs-3g the property would be: volume.mount.ntfs-3g.valid_options. NOTE: To mount the volume with the alternative handler/filesystem you have to pass the correct (alternative) fstype parameter to the Mount() method of the org.freedesktop.Hal.Device.Volume interface
volume.unmount.[alternative].valid_options (string list)	examples: lazy	No, only available if volume.fstype.alternative is set and if the org.freedesktop.Hal.Device.Volume interface is defined.	List of the allowed (valid) unmount options for the defined alternative filesystem handler. Replace [alternative] in the property string with the name of the alternative filesystem handler. For e.g. ntfs-3g the property would be: volume.unmount.ntfs-3g.valid_options
volume.fsversion (string)	example: FAT32		Version number or subtype of the filesystem
volume.label (string)	example: 'Fedora Core 1.90'	Yes (is blank if no label is found)	The label of the volume
volume.uuid (string)	example: 4060-6C11	Yes (is blank if no UUID is found)	The Universal Unique Identifer for the volume
volume.is_disc (bool)		Yes	If the volume stems from an optical disc, this property is true and the device object will also have the capability volume.disc
volume.block_size (string)		No	The block size of the volume
volume.num_blocks (string)		No	Number of blocks on the volume
volume.size (uint64)		No	Size of the volume in bytes
volume.is_partition (bool)		Yes	If the volume stems from a partition on e.g. a hard disk, this property is set to TRUE.

Key (type)	Values	Mandatory	Description
volume.linux.is_device_mapper (bool)		Yes, but only on Linux	If the volume stems from the Linux Device Mapper this property is set to TRUE.
volume.partition.number (int)		If, and only if, volume.is_partition is set to TRUE.	The number of the partition.
volume.partition.label (string)		If, and only if, volume.is_partition is set to TRUE.	Label of partition. Only available for "apm" and "gpt" partition tables. Note that this is not the same as the file system label defined in volume.label.
volume.partition.uuid (string)		If, and only if, volume.is_partition is set to TRUE.	The UUID or GUID of the partition table entry. Only available for "gpt" partition tables.
volume.partition.scheme (string)		If, and only if, volume.is_partition is set to TRUE.	The scheme of the partition table this entry is part of. Note that this is not necessarily the same as storage.partitioning_scheme as some partition tables can embed other partition tables.
	mbr		Master Boot Record
	embr		Extended Master Boot Record
	gpt		GUID Partition Table as defined by EFI
	apm		Apple Partition Map
volume.partition.type (string)		If, and only if, volume.is_partition is set to TRUE.	The type of the partition table entry. Depends on volume.partition.scheme.
	mbr and embr entries		The hexadecimal encoding of the 8-bit partition type, see http://www.win.tue.nl/~aeb/partitions/partition_types-1.html for a list. For example the Linux partition type is represented as the string "0x83".
	gpt entries		The GUID encoded as a string. See http://en.wikipedia.org/wiki/GUID_Partition_Table for a list of well-known GUID's.
	apm entries		Defined in http://developer.apple.com/documentation/mac/Devices/Devices-126.html. Also note that for FAT file systems, it appears that "DOS_FAT_32", "DOS_FAT_16" and "DOS_FAT_12" are also recognized under Mac OS X (I've tested this too) cf.

Key (type)	Values	Mandatory	Description
			http://lists.apple.com/archives/Darwin-drivers/2003/May/msg00021.html
volume.partition.flags (strlist)		If, and only if, volume.is_partition is set to TRUE.	Flags conveying specific information about the partition entry. Dependent on the partitioning scheme.
	mbr and embr entries		Only one flag, "boot", is defined. This is used by some BIOS'es and boot loaders to populate a boot menu. It means that a partition is bootable.
	gpt entries		Only the flag "required" is recognized. This corresponds to bit 0 of the attibutes (at offset 48), meaning "Required for the platform to function. The system cannot function normally if this partition is removed. This partition should be considered as part of the hardware of the system, and if it is removed the system may not boot. It may contain diagnostics, recovery tools, or other code or data that is critical to the functioning of a system independent of any OS."
	apm entries		The following flags are recognized: "allocated" if the partition is already allocated; and "in_use" if the partition is in use; may be cleared after a system reset; and "boot" if partition contains valid boot information; and "allow_read" if partition allows reading; and "allow_write"; if partition allows writing; and "boot_code_is_pic"; if boot code is position independent
volume.partition.media_size (uint64)	example: 500107862016	If, and only if, volume.is_partition is set to TRUE.	If available, size of the current media or the fixed disk in the storage device.
volume.partition.start (uint64)	example: 32256	If, and only if, volume.is_partition is set to TRUE.	If available, the offset where the partition starts on the media or the fixed disk in the storage device.

Device objects with this capability may emit the following device conditions

Condition Name	Parameters	Example	Description

Condition Name	Parameters	Example	Description
VolumeMount	block.device (string), volume.mount_point (string)	/dev/sda1, /media/compact_flash	Emitted when a volume is mounted
VolumeUnmount	block.device (string), volume.mount_point (string)	/dev/sda1, /media/compact_flash	Emitted when a volume is unmounted
VolumeUnmountForced	block.device (string), volume.mount_point (string)	/dev/sda1, /media/compact_flash	Emitted when a volume is forcibly unmounted because the media backing the volume was removed.

volume.disc namespace

This namespace is for device objects that represent optical discs, e.g. device objects with the capability volume.disc. Such device objects will also have the capability volume.

Key (type)	Values	Mandatory	Description
volume.disc.has_audio (bool)		Yes	Is true only if the disc contains audio
volume.disc.has_data (bool)		Yes	Is true only if the disc contains data
volume.disc.is_vcd (bool)		Yes	Is true only if the disc is a Video CD
volume.disc.is_svcd (bool)		Yes	Is true only if the disc is a Super Video CD
volume.disc.is_videodvd (bool)		Yes	Is true only if the disc is a Video DVD
volume.disc.is_appendable (bool)		Yes	Is true only if it's possible to write additional data
volume.disc.is_blank (bool)		Yes	Is true only if the disc is blank
volume.disc.is_rewritable (bool)		Yes	Is true only if the disc is rewritable
volume.disc.capacity (uint64)		No	Capacity of disc, in bytes
volume.disc.type (string)		Yes	This property specifies the physical type of the disc
	cd_rom		CD-ROM disc
	cd_r		CD-R disc
	cd_rw		CD-RW disc
	dvd_rom		DVD-ROM disc
	dvd_ram		DVD-RAM disc
	dvd_r		DVD-R disc
	dvd_rw		DVD-RW disc
	dvd_r_dl		DVD-R dual layer disc
	dvd_plus_r		DVD+R disc

Key (type)	Values	Mandatory	Description
	dvd_plus_r_dl		DVD+R dual layer disc
	dvd_plus_rw		DVD+RW disc
	dvd_plus_rw_dl		DVD+RW dual layer disc
	bd_rom		BD-ROM disc
	bd_r		BD-R disc
	bd_re		BD-RE disc
	hddvd_rom		HD DVD-ROM disc
	hddvd_r		HD DVD-R disc
	hddvd_rw		HD DVD-Rewritable disc
	unknown		Unknown type or lack of support from drive to determine the type

Misc. Properties

access_control namespace

Device objects with the capability **access_control** represent devices where access to a special device file can be granted/revoked to unprivileged users.

Key (type)	Values	Mandatory	Description
access_control.file (string)	Example: /dev/snd/pcmC0D1p	Yes	Path to the special device file that access can be granted to.
access_control.type (string)	Example: cdrom	Yes	Type of access - only makes sense when PolicyKit support is enabled; it's used by PolicyKit to compute what privilege to check for by prepending **org.freedesktop.hal.device-access.** to the value.
access_control.grant_user (strlist)	Example: "gdm, flumotion"	No	List of UNIX user names to always grant access to the device. This is useful for 3rd party system-wide packages that need access to a device to function properly.
access_control.grant_group (strlist)	Example: "pvr_software, staff"	No	List of UNIX group names to always grant access to the device. This is useful for 3rd party system-wide packages that need access to a device to function properly.

See also the section called “org.freedesktop.Hal.Device.AccessControl interface”.

Deprecated Properties

The section represents properties that are deprecated and should be no longer used. The properties/keys will be removed, but not before the date in the following table:

Key (type)	Replacement	Remove (date)	Notes
()	()	xxxx-xx-xx	

Chapter 6. D-Bus interfaces

Table of Contents

All of the HAL D-Bus interfaces are introspectable using the standard D-Bus introspection methods (e.g. they all implement the `org.freedesktop.DBus.Introspectable` interface). For example, a command like

```
$ dbus-send --system --print-reply --dest=org.freedesktop.Hal \
            /org/freedesktop/Hal/devices/computer             \
            org.freedesktop.DBus.Introspectable.Introspect
```

will print out the introspection XML for what interfaces (ie. methods and signals) the given hal device object supports. For brevity, the `org.freedesktop.Hal` prefix have been stripped from the exceptions listed in the following sections.

Also note that other exceptions than the ones listed may be thrown; for example the `org.freedesktop.Hal.Device.InterfaceLocked` exception may be thrown regardless of how the interface is implemented (depending on if some other process is holding a lock on the device cf.

Chapter 4, *Locking*); if PolicyKit support is enabled, the `org.freedesktop.Hal.Device.PermissionDeniedByPolicy` exception may be thrown (the two first words in the exception detail is resp. a) the privilege the caller didn't have; b) the textual result code from PolicyKit specifying if the caller can obtain the privilege) if the caller is not privileged and so on.

org.freedesktop.Hal.Device interface

Every hal device object (e.g. objects where the object path is prefixed with `/org/freedesktop/Hal/devices/`) implements this interface. It provides generic functionality. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetProperty	Variant	String key	NoSuchProperty	Get property.
GetPropertyString	String	String key	NoSuchProperty, TypeMismatch	Get property.
GetPropertyStringList	String[]	String key	NoSuchProperty, TypeMismatch	Get property.
GetPropertyInteger	Int	String key	NoSuchProperty, TypeMismatch	Get property.
GetPropertyUInt64	UInt64	String key	NoSuchProperty, TypeMismatch	Get property.
GetPropertyBoolean	Bool	String key	NoSuchProperty, TypeMismatch	Get property.
GetPropertyDouble	Double	String key	NoSuchProperty, TypeMismatch	Get property.
SetProperty	Variant	String key	PermissionDenied	Set property.
SetPropertyString	String	String key	PermissionDenied, TypeMismatch	Set property.
SetPropertyStringList	String[]	String key	PermissionDenied, TypeMismatch	Set property.
SetPropertyInteger	Int	String key	PermissionDenied, TypeMismatch	Set property.
SetPropertyUInt64	UInt64	String key	PermissionDenied, TypeMismatch	Set property.
SetPropertyBoolean	Bool	String key	PermissionDenied, TypeMismatch	Set property.
SetPropertyDouble	Double	String key	PermissionDenied, TypeMismatch	Set property.
RemoveProperty		String key	NoSuchProperty, PermissionDenied	Remove a property.
GetPropertyType	Int	String key	NoSuchProperty	Get the type of a property. Returns the D-Bus type as an integer.
PropertyExists	Bool	String key		Determine if a property exists.

Method	Returns	Parameters	Throws	Description
AddCapability		String capability	PermissionDenied	Adds a capability to a device.
QueryCapability	Bool	String capability		Determine if a device have a capability.
Lock	Bool	String reason	DeviceAlreadyLocked	Acquires an advisory lock on the device. Returns TRUE if the lock was acquired.
Unlock	Bool		DeviceNotLocked	Releases an advisory lock on the device. Returns TRUE if the lock was released.
AcquireInterfaceLock		String interface_name, Bool exclusive	PermissionDenied, Device.InterfaceAlreadyLocked	Acquires a lock on an interface for a specific device. See Chapter 4, *Locking* for details.
ReleaseInterfaceLock		String interface_name	PermissionDenied, Device.InterfaceNotLocked	Releases a lock on an interface for a specific device. See Chapter 4, *Locking* for details.
IsCallerLockedOut	Bool	String interface_name, String caller_unique_name	PermissionDenied	Determines whether a given process on the system message bus is locked out from an interface on a specific device. Only HAL helpers are privileged to use this method. See Chapter 4, *Locking* for details.
IsCallerPrivileged	String	String privilege, String caller_unique_name	PermissionDenied, Error	Determines whether a given process on the system message bus is authorized according to PolicyKit on a specific device for a specific PolicyKit privilege. Unprivileged callers (e.g. with a non-zero uid) can only ask about `caller_unique_name` that matches their own uid; if this is violated `PermissionDenied` will be thrown. This can be used ahead of time to see if a given call will succeed or if it requires privilege elevation (TODO: clarify this once PolicyKit can auth over D-Bus). Returns the textual representation of a PolKitResult value on success. If HAL is not built with PolicyKit support, this method always throws the `org.freedesktop.Hal.Device.Error` exception.
IsLockedByOthers	Bool	String interface_name		Determines whether a determines other processes than the caller holds a lock on the given device. See Chapter 4, *Locking* for details.

Method	Returns	Parameters	Throws	Description
StringListAppend		String key, String value	PermissionDenied, TypeMismatch	Appends an item to a string list.
StringListPrepend		String key, String value	PermissionDenied, TypeMismatch	Prepends an item to a string list.
StringListRemove		String key, String value	PermissionDenied, TypeMismatch	Removes an item from a string list.
EmitCondition	Bool	String name, String details	PermissionDenied	Emit a condition on a device object.
Rescan	Bool		PermissionDenied	Force an updates of the properties of a device object by rereading data that is not monitored for changes.
Reprobe	Bool		PermissionDenied	Cause a synthetic remove and subsequent add of the given device object including all children beneath it. Will generate at least one pair of DeviceRemoved() and DeviceAdded() signals on the Manager interface.
ClaimInterface	Bool	String name, String introspection_xml	PermissionDenied	An addon can use this method for making the HAL daemon route all D-Bus calls on the given interface to the addon via the peer to peer D-Bus connection between the addon and the HAL daemon.
AddonIsReady	Bool		PermissionDenied	An addon needs to call this method when it's ready for the device to be announced on D-Bus. The rationale for this method is to allow an addon to modify the device object and claim interfaces before the device is announced on D-Bus.

The following signals are emitted:

Signal	Parameters	Description
PropertyModified	Int num_changes, Array of struct {String property_name, Bool removed, Bool added}	One or more properties on the device object have changed.
Condition	String name, String details	A generic mechanism used to specify a device condition that cannot be expressed in device properties. (NOTE: newly written code should use dedicated signals on a dedicated interface.).

Signal	Parameters	Description
InterfaceLockAcquired	String lock_name, String lock_owner, Int num_holders	Sent when a process acquires an interface lock on the device.
InterfaceLockReleased	String lock_name, String lock_owner, Int num_holders	Sent when a process releases an interface lock on the device.

org.freedesktop.Hal.Device.AccessControl interface

This interface provides a mechanism for discovering when an ACL is added or removed for a device file. The following signals are available:

Method	Parameters	Description
ACLAdded	UInt unix_user_id	Emitted when an ACL have been added for a UNIX user on a device object with the `access_control` capability.
ACLRemoved	UInt unix_user_id	Emitted when an ACL have been removed for a UNIX user on a device object with the `access_control` capability.

This interface does not export any methods.

org.freedesktop.Hal.Device.CPUFreq interface

This interface provides a mechanism to configure CPU frequency scaling. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetCPUFreqAvailableGovernors	String[]			Retrieves a list of available CPU scaling governors.
GetCPUFreqGovernor	String			Get currently selected CPU Frequency governor.
SetCPUFreqGovernor	Void	String governor	CPUFreq.UnknownGovernor	Selects a CPU frequency scaling governor for all CPUFreq interfaces the kernel provides. If the userspace governor is set, this interface also contains a proper scaling mechanism.
SetCPUFreqPerformance	Void	Int (1 to 100)	CPUFreq.NoSuitableGovernor	Sets the performance of the dynamic scaling mechanism. This method summarizes and abstracts all the different settings which can be taken for dynamic frequency

Method	Returns	Parameters	Throws	Description
				adjustments, like at which load to switch up frequency or how many steps the mechanism should traverse until reaching the maximum frequency. The higher the value, the more performance you get. Respectively, the higher the value, the sooner and the more often the frequency is switched up.
GetCPUFreqPerformance	Int		CPUFreq.NoSuitableGovernor	Get the tuning value for the governor.
SetCPUFreqConsiderNice	Void	Bool consider_niced_processes	CPUFreq.NoSuitableGovernor	Whether or not niced processes should be considered on CPU load calculation. If niced processes are considered, they can cause a frequency increment although their absolute load percentage wouldn't trigger the scaling mechanism to switch up the frequency. The default setting is 'false'.
GetCPUFreqConsiderNice	Bool		CPUFreq.NoSuitableGovernor	Whether nice'ed processes are considered by the governor.

This interface does not emit any signals.

org.freedesktop.Hal.Device.DockStation interface

This interface provides a mechanism to deal with dock stations.

Method	Returns	Parameters	Throws	Description
Undock	Int	Void	DockStation.NotDocked	Undock the system from a dock station

This interface does not emit any signals.

org.freedesktop.Hal.Device.KeyboardBacklight interface

This interface provides a mechanism to get/set the brightness of the keyboard backlight. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetBrightness	Int			Get the current brightness.

Method	Returns	Parameters	Throws	Description
SetBrightness		Int brightness		Set the current brightness.

This interface does not emit any signals.

org.freedesktop.Hal.Device.KillSwitch interface

This interface provides a mechanism for both querying whether a radio is on as well as turning it on and off. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetPower	Int			Returns 1 if, and only if, the power is on.
SetPower		Bool		Set the power of the radio.

This interface does not emit any signals.

org.freedesktop.Hal.Device.Leds interface

This interface provides a mechanism to get/set the brightness of a LED (light-emitting diode). The following methods are available:

Method	Returns	Parameters	Throws	Description
GetBrightness	Int			Get the current brightness of the LED.
SetBrightness		Int brightness		Set the current brightness of the LED. The value 0 deactivate normally the LED, any higher value active the LED again.

This interface does not emit any signals.

org.freedesktop.Hal.Device.LaptopPanel interface

This interface provides a mechanism to get/set the brightness of a laptop panel. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetBrightness	Int			Get the current brightness.
SetBrightness		Int brightness		Set the current brightness.

This interface does not emit any signals.

org.freedesktop.Hal.Device.LightSensor interface

This interface provides a mechanism to get information from a light sensor. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetBrightness	Int[]			The current brightness as measured by the light sensor.

This interface does not emit any signals.

org.freedesktop.Hal.Device.Storage interface

This interface provides a mechanism to interact with a storage device. The following methods are available:

Method	Returns	Parameters	Throws	Description
Eject	Int	String[] options	Volume.PermissionDenied, Volume.NotMounted, Volume.NotMountedByHal, Volume.Busy, Volume.InvalidEjectOption	Unmounts all volumes and possibly ejects the media. Note that this method is not only restricted to optical drives.
CloseTray	Int	String[] options	Storage.InvalidCloseTrayOption	Attempts to close the tray. Only really makes sense for (optical) drives that uses a tray loading mechanism.

This interface does not emit any signals.

org.freedesktop.Hal.Device.Storage.Removable interface

This interface provides a mechanism to interact with a storage device that uses removable media. The following methods are available:

Method	Returns	Parameters	Throws	Description
CheckForMedia	Bool			Polls a storage device for media.

This interface does not emit any signals.

org.freedesktop.Hal.Device.SystemPowerManagement interface

This interface provides a mechanism to affect system-wide power management. Normally only the root computer device object (`/org/freedesktop/Hal/devices/computer`) implements this interface. The following methods are available:

Method	Returns	Parameters	Throws	Description
Suspend	Int	Int num_secs_to_wakeup	SystemPowerManagement.NotSupported, SystemPowerManagement.AlarmNotSupported	Puts the system in a suspended state (typically ACPI S3) for num_secs_to_wakeup seconds. If the given time is zero, the system is put in the suspended state indefinitely. If wake-up isn't supported the the AlarmNotSupported exception is thrown. Latency for the system to return to an operational state is in the order of magnitude of 5 seconds.
Hibernate	Int		SystemPowerManagement.NotSupported	Save system state to persistent storage and power off the system (typically ACPI S4). Latency for the system to return to an operational state is in the order of magnitude of one minute.
SuspendHybrid	Int	Int num_secs_to_wakeup	SystemPowerManagement.NotSupported, SystemPowerManagement.AlarmNotSupported	Puts the system in a suspended state (typically ACPI S3) for num_secs_to_wakeup seconds but also write the system state to persistent storage so the system can resume even if power is removed. Like with Suspend(), if the given time is zero, the system is put in the suspended state indefinitely. If wake-up isn't supported the the AlarmNotSupported exception is thrown.
Shutdown	Int		SystemPowerManagement.NotSupported	Shut down the system.
Reboot	Int		SystemPowerManagement.NotSupported	Reboot the system.
SetPowerSave	Int	Bool should_save_power	SystemPowerManagement.NotSupported	If the boolean passed is TRUE, the system will be configured to save as much power as possible by e.g. enabling laptop mode to avoid spinning up disks. Typically, power management daemons will call this method when it determines that the system is running on battery power.

This interface does not emit any signals.

Implementors of power management daemons should make sure that their software respects the locking guidelines described in Chapter 6, *D-Bus interfaces*.

org.freedesktop.Hal.Device.Volume interface

This interface provides a mechanism to interact with a volume that has a mountable file system. The following methods are available:

Method	Returns	Parameters	Throws	Description
Mount	Int	String mount_point, String fstype, String[] options	Volume.PermissionDenied, Volume.AlreadyMounted, Volume.InvalidMountOption, Volume.UnknownFilesystemType, Volume.InvalidMountPoint, Volume.MountPointNotAvailable, Volume.CannotRemount	Mounts a volume.
Unmount	Int	String[] options	Volume.PermissionDenied, Volume.NotMounted, Volume.NotMountedByHal, Volume.Busy, Volume.InvalidUnmountOption	Unmount a volume.
Eject	Int	String[] options	Volume.PermissionDenied, Volume.NotMounted, Volume.NotMountedByHal, Volume.Busy, Volume.InvalidEjectOption	Unmounts all volumes from the originating drive and possibly ejects the media. Note that this method is not only restricted to optical drives.

This interface does not emit any signals.

If a volume originates from a storage device (and all volumes do), it also is checked whether the caller is locked out of the `org.freedesktop.Hal.Device.Storage` interface of the originating storage device. As a corollary, it is sufficient to just either a) lock the storage device; or b) globally lock the `org.freedesktop.Hal.Device.Storage` interface if one wants to lock out callers from mounting volumes from either a specific drive or all drives.

org.freedesktop.Hal.Device.Volume.Crypto interface

This interface provides a mechanism to interact with a volume that is encrypted at the block layer. The following methods are available:

Method	Returns	Parameters	Throws	Description
Setup	Int	String passphrase	Volume.Crypto.CryptSetupMissing, Volume.Crypto.SetupError, Volume.Crypto.SetupPasswordError	Unlocks an encrypted file system. If successful, a cleartext volume will appear.
Teardown	Int		Volume.Crypto.TeardownError	Teardown the cleartext volume.

This interface does not emit any signals.

For objects implementing this interface, it will also be checked if the caller is locked out of the Volume interface on the device (and per the section called “org.freedesktop.Hal.Device.Volume interface” this includes checking whether the caller is locked out of the `org.freedesktop.Hal.Device.Storage` interface for the storage device that the volume originates from).

org.freedesktop.Hal.Device.WakeOnLan interface

This interface provides a mechanism to configure Wake On LAN capabilities. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetSupported	Bool		WakeOnLan.NoEthtool	Get if device supports Wake On LAN
GetEnabled	Bool		WakeOnLan.NoEthtool	Get if Wake On LAN is enabled
SetEnabled	Void	Bool	WakeOnLan.NoEthtool	Enable or disable Wake On LAN

This interface does not emit any signals.

org.freedesktop.Hal.Manager interface

Only the `/org/freedesktop/Hal/Manager` object implements this interface. It's primarily used to discover devices. The following methods are available:

Method	Returns	Parameters	Throws	Description
GetAllDevices	Objref[]			Get all UDI's in the database.
DeviceExists	Bool			Determines if a given object exists.
FindDeviceStringMatch	Objref[]	String key, String value		Find devices for which the given string property assumes the given value.
FindDeviceByCapability	Objref[]	String capability		Finds devices of the given capability.
NewDevice	Objref		PermissionDenied	Creates a new device object in the temporary device list (TDL) and return the UDI. Caller must be uid 0.
Remove		Objref tmp_udi	NoSuchDevice, PermissionDenied	Removes a device object that was created in the TDL. Caller must be uid 0.
CommitToGdl		Objref tmp_udi, Objref udi	NoSuchDevice, PermissionDenied	Moves a device from the temporary device list (TDL) to the global device list (GDL). Caller must be uid 0.
AcquireGlobalInterfaceLock		String interface_name, Bool exclusive	Device.InterfaceAlreadyLocked	Acquires a global lock on an interface. See Chapter 4, *Locking* for details.
ReleaseGlobalInterfaceLock		String interface_name	Device.InterfaceNotLocked	Releases a global lock on an interface. See Chapter 4, *Locking* for details.

Method	Returns	Parameters	Throws	Description
SingletonAddonIsReady		String command_line	PermissionDenied, SyntaxError	Called by singleton addons to signal that they are ready to handle devices. A singleton addon should implement the org.freedesktop.Hal.Singleton interface.

The following signals are emitted:

Signal	Parameters	Description
DeviceAdded	Objref obj	A device was added to the global device list (GDL).
DeviceRemoved	Objref obj	A device was removed from the global device list (GDL).
NewCapability	Objref obj, String cap	A device gained a new capability.
GlobalInterfaceLockAcquired	String lock_name, String lock_owner, Int num_holders	Sent when a process acquires a global interface lock.
GlobalInterfaceLockReleased	String lock_name, String lock_owner, Int num_holders	Sent when a process releases a global interface lock.

org.freedesktop.Hal.SingletonAddon interface

This interface is provided by singleton addons to allow the Manager to request handling of new devices and removal of old ones. This differs from other HAL interface definitions in that it is provided by addon processes, rather than the HAL daemon itself. It should be exported on the path `/org/freedesktop/Hal/Manager`.

Method	Returns	Parameters	Throws	Description
DeviceAdded	String udi Dict(String,Variant) property_set			Called by the HAL Manager when a device is added that has this singleton listed in info.addons.singleton An addon implementing this function should start handling the device before returning, and keep track that is is handling this udi.
DeviceRemoved	String udi Dict(String,Variant) property_set			Called by the HAL Manager when a device is added that has this singleton listed in info.addons.singleton The implementer of this function should keep track of which devices it is still handling and exit when no longer handling any devices.

This interface does not export any methods.