RFC 202 - USB Information Device Category

Draft

36 Pages

Abstract

This document defines the device category for USB devices information in OSGi.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design
The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

`Source code is shown in this typeface.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	April 10, 2013	Initial version Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.2	July 4, 2013	– added RFC number to title – added 5.1.1.1 Optional Device Access Category – modified 5.2.2 Service properties from USB Specification Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp

Revision	Date	Comments
v0.3	Sept. 9, 2013	– modified based on the F2F meeting in Paris Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.4	Nov. 19, 2013	– modified based on the F2F meeting in Hursley Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.5	Nov. 19, 2013	– Updated Javadoc section Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.6	Feb. 17, 2014	– Modified based on the F2F meeting in Sofia – restructured chapters – added Preconditions and Behavior Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.7	April 7, 2014	Modified based on the F2F meeting in Cologne. – Added operation example details and added USB serial example 2, USB storage example 3 – Added Considered Alternatives (8.1) – Modified some wording – Modified some service properties' Java type Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.8	May 1, 2014	– Modified about USBDevice service registration rule – Changed treatment about bInterfaceClass/ SubClass/ Protocol – Changed some service properties' Java types – Added some service properties – Modified Security Considerations – English corrections Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.9	Sept. 18, 2014	– Changed RFC title and category/interface/package names based on F2F meeting in Madrid – Changed match scale and some property key based on F2F meeting in Basel – Changed RFC templete to a new one – Replaced Fig. 3 (Class Diagram) – Add a match scale (MATCH_VERSION) Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp

1 Introduction

OSGi Device Access Specification defines a unified and sophisticated way to handle devices attached to a residential gateway or devices found in the home network by using various protocols such as USB, Zigbee, ZWave, KNX, UPnP etc. However, OSGi Device Access Specification clearly declare that Device Category must be defined outside of OSGi Device Access Specification.

Recently, OSGi is gaining popularity as enabling technology for building embedded system in residential market. It gets popular that a HGW has USB interfaces and the needs of handling USB devices attached to a residential gateway is increased.

This RFC defines a device category for USB devices.

2 Application Domain

Currently there are several standardization bodies such as OSGiA, HGI, BBF, which deal with the deployment of services in an infrastructure based on the usage of a Residential Gateway running OSGi as Execution Platform.

In order to realize services which access not only IP devices but also non-IP devices connected to the residential gateway, there are several protocols for home networks, such as ZigBee, Z-Wave, KNX/EHS, ECHONET, ECHONET-LITE, etc.. While some residential gateways support those protocols on themselves, others do not. Many residential gateways have USB interfaces and there exist USB dongles which support those protocols. Therefore, there is a need to support those protocols using USB dongles attached to a residential gateway (Fig. 1). In addition, most of USB dongles can be controlled through Serial Communication.

The existing OSGi specifications which address related topics are:

- Device Access Specification - focuses on the dynamic discovery of the proper driver when a new device it attached/connected to the residential gateway

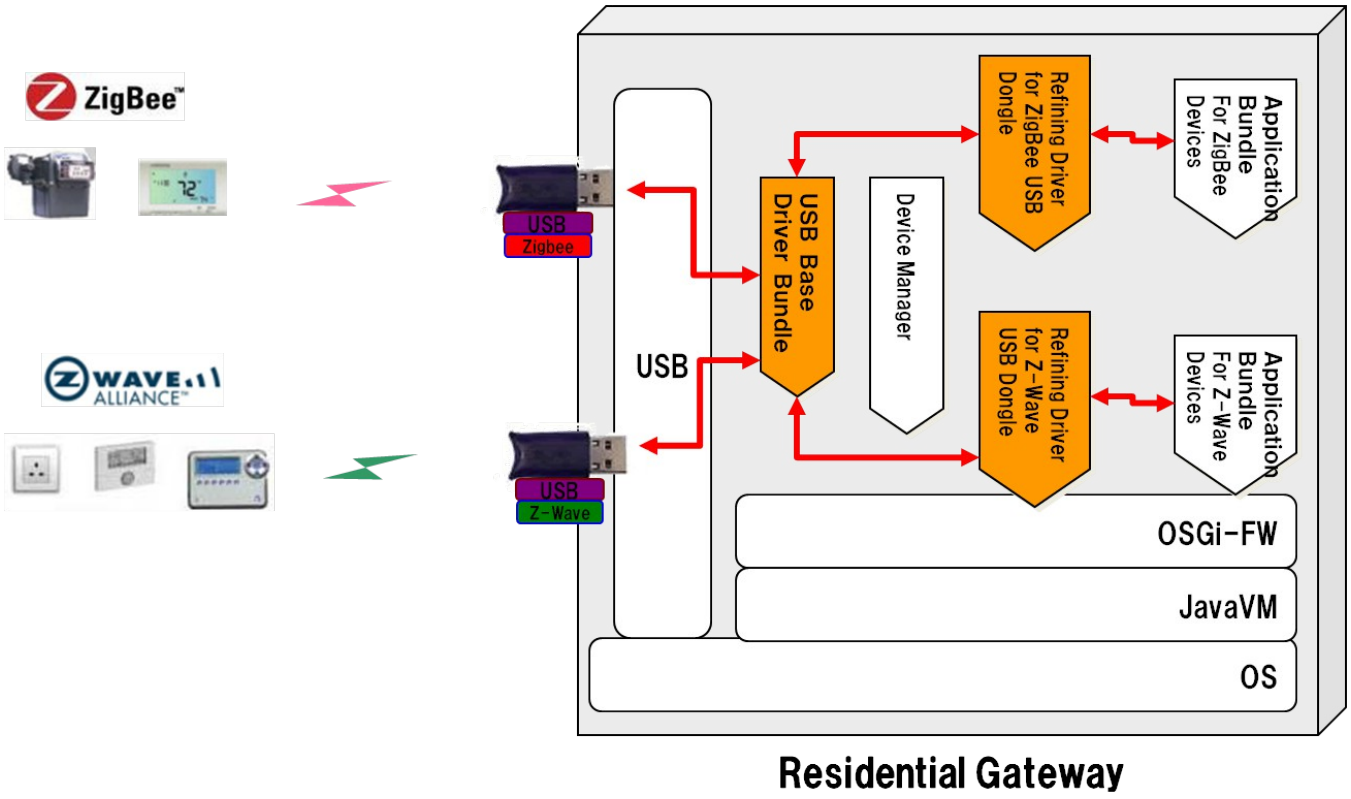

Fig 1 USB Dongles and Residential gateway

2.1 Terminology + Abbreviations

- Base Drivers: see "103.4.2.1" in OSGi Device Access Specification [3].
- Refining Drivers: see "103.4.2.2" in OSGi Device Access Specification [3].
- Match value: the value match() method of a Driver service registered by the refining driver bundle returns. Matching is explained in "103.7.2 The Device Attachment Algorithm" in OSGi Device Access Specification [3].
- Device Descriptor: see "9.6.1" in Universal Serial Bus Specification[4].

3 Problem Description

The existing OSGi Device Access Specification provides the unified way to installation and activation of driver bundles. However, the OSGi Device Access Specification declares the device category for specific devices must be defined outside of itself. Currently, no device category for USB devices has been defined yet.

The lack of the device category for USB devices causes the following problems.

[Problem 1] The developer of a refining driver bundle, which registers a Driver service at its activation, cannot design and implement Driver#attach(ServiceReference) method without knowledge of service properties set to the Device service registered by a USB base driver.

[Problem 2] The developer of a refining driver bundle, which registers a Driver service at its activation, cannot design and implement Driver#match(ServiceReference) method without knowledge of service properties set to the Device service registered by a USB base driver and without the definition of match values to be returned.

In other words, without the device category for USB devices, a refining driver bundle developed by developer A can cooperate with the USB base driver bundle developed by the same developer A but cannot cooperate with the USB base driver bundles developed by the different developer B.

4 Requirements

[REQ_1] The solution MUST be compatible with OSGi Device Access Specification .

[REQ_2] The solution MUST define the details of the registration of a Device service by a USB base driver bundle when a USB device is attached.

[REQ_2-1] The solution MUST define the service interface under which the Device service is registered.

[REQ_2-2] The solution MUST define the service properties with which the Device service is registered: A set of service properties, their data types, and semantics, each of which must be declared as either MANDATORY or OPTIONAL.

[REQ_3] The solution MUST define the way how a driver bundle controls an attached USB device which can be controlled through Serial communication.

[REQ_4] The solution MAY define a range of match values specific to this device category.

[REQ_5] The range of match values MUST be sufficient to describe the required range of native serial drivers specified by the HGI, especially the following ones:

- Class drivers for Human Interface Device (HID) and Communications Device Class (CDC) [1]
- Drivers for FTDI Virtual Com Ports with a variable list of supported USB Vendor Identifiers and Product Identifiers[2].
- Drivers for Silicon Labs CP210x USB to UART bridge and CP2110 HID USB to UART bridge[3].
- USB drivers for Prolific PL-2303 USB to Serial Bridge Controller[4].

5 Technical Solution

~~USB device category~~USB information device category defines the following elements:

1. An interface that all devices belonging to this category must implement.
2. A set of service registration properties, their data types, and semantics, each of which must be declared as either MANDATORY or OPTIONAL for this device category.
3. A range of match values specific to this device category.

This document defines other elements for some specific USB classes, because of there are clear use cases (Fig 2). This document does not define the details of all USB classes, they are roles of refining drivers. Otherwise, future specification may provide those definitions.

1 http://www.usb.org/developers/devclass_docs#approved for details of USB device classes

2 http://www.ftdichip.com/Drivers/VCP.htm

3 http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx.

4 http://www.prolific.com.tw

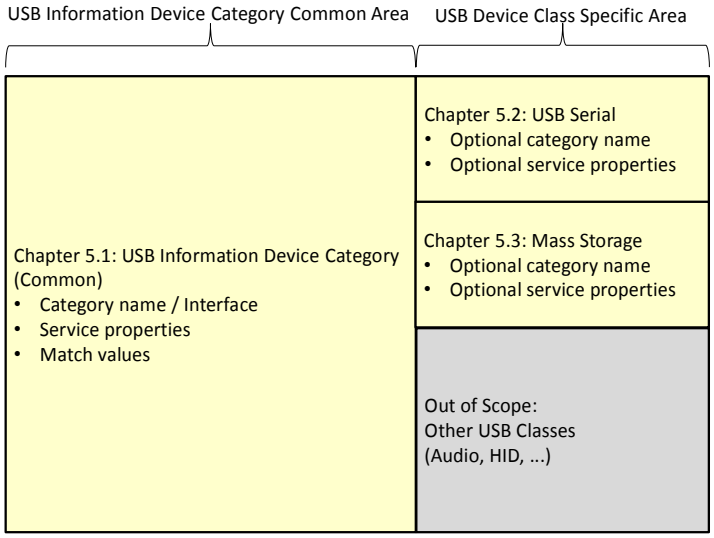

Fig 2: Structure of this document

5.1 ~~USBDevice~~USBInfoDevice Service

The device services are registered in the OSGi service registry with `org.osgi.service.~~usb.USBDevice~~usbinfo.USBInfoDevice` interface. The service is registered by a ~~USB base driver~~USB information base driver bundle when a USB device is attached. A ~~USB base driver~~USB information base driver bundle must implement `org.osgi.service.~~usb.USBDevice~~USBInfoDeviceusbinfo.USBInfoDevice` interface and register the OSGi service under `org.osgi.service.~~usb.USBDevice~~usbinfo.USBInfoDevice`. Refining drivers can find USB devices via ~~USBDevice~~USBInfoDevice services and identify the device. The ~~USBDevice~~USBInfoDevice service has a set of properties ~~(Fig 3)~~.

Universal Serial Bus Specification (USB Specification) [4]. defines that a USB device has USB interface(s). A ~~USB base driver~~USB information base driver bundle must register ~~USBDevice~~USBInfoDevice services number of USB interfaces. A ~~USBDevice~~USBInfoDevice service has information that contains a USB device information and a USB interface information.

Figure 3 shows the class diagram.

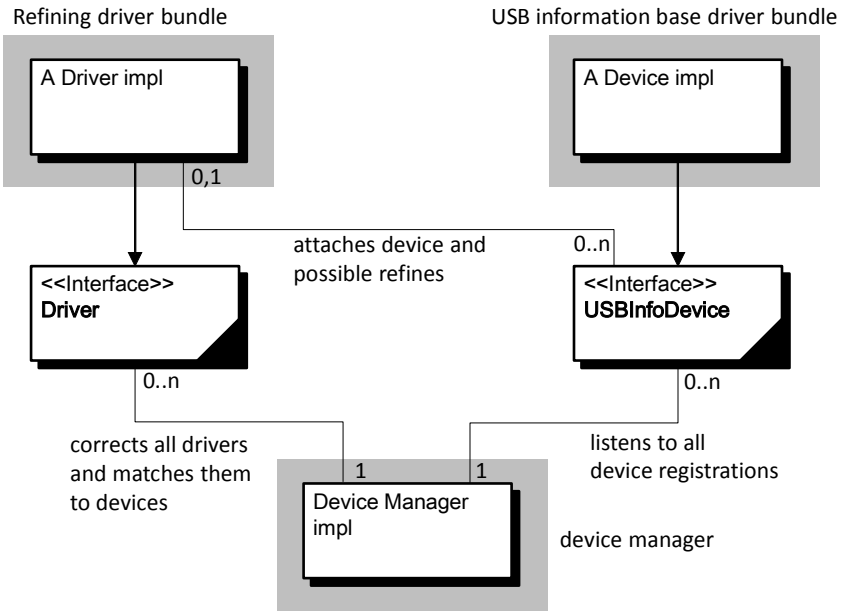

Fig 3:Class Diagram

5.1.1 Assumptions

The ~~USB base driver~~USB information base driver may need native drivers such as kernel drivers on Linux (Fig 4). This document has a precondition that there are native drivers. It is out of scope how to install native drivers.

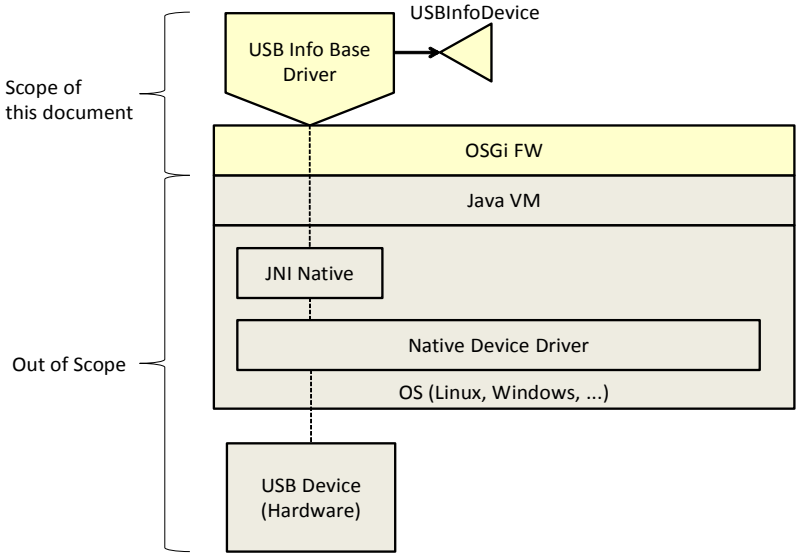

Fig 4:Software Structure and Scope

5.1.2 Device Access Category

The device access category is called "USBInfo". The category name is defined as a value of ~~USBDevice~~USBInfoDevice. `DEVICE_CATEGORY` constant. It can be used as a part of `org.osgi.service.device.Constants.DEVICE_CATEGORY` service key value. The category imposes the following specification rules.

- ~~USBDevice~~USBInfoDevice.`DEVICE_CATEGORY` - MANDATORY property. The value is "USBInfo". Constant for the value of the service property `DEVICE_CATEGORY` used for all USB devices. A ~~USB base driver~~USB information base driver bundle must set this property key.

5.1.3 Service properties from USB Specification

The USB Specification defines a device descriptor. USB devices report their attributes using descriptors. ~~USBDevice~~USBInfoDevice service has some properties from the USB device descriptor. Table 1 shows them.

Table 1: Device Descriptor and Service Property

Device Descriptor's Field from USB Spec.	**~~USB Device Category~~USBInfoDevice's service property**	**M/O**	**Java type**
bLength	`none`	-	-
bDescriptorType	`none`	-	-
bcdUSB	~~USB.device~~usbinfo.bcdUSB	O	`String`
bDeviceClass	~~USB.device~~usbinfo.bDeviceClass	M	`String`
bDeviceSubClass	~~USB.device~~usbinfo.bDeviceSubClass	M	`String`
bDeviceProtocol	~~USB.device~~usbinfo.bDeviceProtocol	M	`String`
bMaxPacketSize0	~~USB.device~~usbinfo.bMaxPacketSize0	O	`Integer`
idVendor	~~USB.device~~usbinfo.idVendor	M	`String`
idProduct	~~USB.device~~usbinfo.idProduct	M	`String`
bcdDevice	~~USB.device~~usbinfo.bcdDevice	M	`String`
iManufacturer	~~USB.device~~usbinfo.~~i~~Manufacturer	O	`String`
iProduct	~~USB.device~~usbinfo.~~i~~Product	O	`String`
iSerialNumber	~~USB.device~~usbinfo.~~i~~SerialNumber	O	`String`
bNumConfigurations	~~USB.device~~usbinfo.bNumConfigurat	O	`Integer`

	ions		

- ~~USB.device~~usbinfo.bcdUSB - OPTIONAL property key. The value is String, the 4-digit BCD format.
  - Example: ”0210”
- ~~USB.device~~usbinfo.bDeviceClass - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: “ff”
- ~~USB.device~~usbinfo.bDeviceSubClass - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: ”ff”
- ~~USB.device~~usbinfo.bDeviceProtocol - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: ”ff”
- ~~USB.device~~usbinfo.bMaxPacketSize0 – OPTIONAL property key. The value is Integer.
- ~~USB.device~~usbinfo.idVendor - MANDATORY property key. The value is String, hexadecimal, 4-digits.
  - Example: “0403”
- ~~USB.device~~usbinfo.idProduct - MANDATORY property key. The value is String, hexadecimal, 4-digits.
  - Example: “8372”
- ~~USB.device~~usbinfo.bcdDevice - MANDATORY property key. The value is String, the 4-digit BCD format.
  - Example: ”0200”
- ~~USB.device~~usbinfo.~~i~~Manufacturer - OPTIONAL property key. The value is String of indicated in iManufacturer. (The value is not the index.)
  - Example: "Buffalo Inc."
- ~~USB.device~~usbinfo.~~i~~Product - OPTIONAL property key. The value is String of indicated in iProduct. (The value is not the index.)
  - Example: "USB2.0 PC Camera"

- ~~USB.device~~usbinfo.~~i~~SerialNumber - OPTIONAL property key. The value is String of indicated in iSerialNumber. (The value is not the index.)
  - Example: "57B0002600000001"
- ~~USB.device~~usbinfo.bNumConfigurations – OPTIONAL property key. The value is Integer.

According to the USB Specification, a device descriptor has some interface descriptors.

Refining drivers need each interface descriptors' *bInterfaceClass, bInterfaceSubClass* and *bInterfaceProtocol* to identify devices. So these fields add to the service properties (see Table 2).

Table 2: Interface Descriptor and Service Property

Interface Descriptor's Field from USB Spec.	**~~USB Device Category~~USBInfoDevice's service property**	**M/O**	**Java type**
bLength	none	-	-
bDescriptorType	none	-	-
bInterfaceNumber	~~USB.device~~usbinfo.bInterfaceNumber	M	Integer
bAlternateSetting	~~USB.device~~usbinfo.bAlternateSetting	O	Integer
bNumEndpoints	~~USB.device~~usbinfo.bNumEndpoints	O	Integer
bInterfaceClass	~~USB.device~~usbinfo.bInterfaceClass	M	String
bInterfaceSubClass	~~USB.device~~usbinfo.bInterfaceSubClass	M	String
bInterfaceProtocol	~~USB.device~~usbinfo.bInterfaceProtocol	M	String
iInterface	~~USB.device~~usbinfo.~~i~~Interface	O	String

- ~~USB.device~~usbinfo.bInterfaceNumber – MANDATORY property key. The value is Integer.
- ~~USB.device~~usbinfo.bAlternateSetting – OPTIONAL property key. The value is Integer.
- ~~USB.device~~usbinfo.bNumEndpoints – OPTIONAL property key. The value is Integer.
- ~~USB.device~~usbinfo.bInterfaceClass - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: “ff”

- ~~USB.device~~usbinfo.bInterfaceSubClass - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: ”ff”
- ~~USB.device~~usbinfo.bInterfaceProtocol - MANDATORY property key. The value is String, hexadecimal, 2-digits.
  - Example: ”ff”
- ~~USB.device~~usbinfo.~~i~~Interface - OPTIONAL property key. The value is String of indicated in iInterface. (The value is not the index.)

5.1.4 Other Service properties

Some other service properties are needed to identify and access a device by refining drivers.

Table 3: Other service properties

Service property	M/O	Java type
~~USB.device~~usbinfo.bus	M	Integer
~~USB.device~~usbinfo.address	M	Integer

- ~~USB.device~~usbinfo.bus - MANDATORY property key. The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB bus assigned when connecting the USB device. USB bus ID is integer. The USB bus ID does not change while the USB device remains connected.
  - Example: 3
- ~~USB.device~~usbinfo.address - MANDATORY property key. The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB address assigned when connecting the USB device. USB address is integer (001-127). The USB address does not change while the USB device remains connected.
  - Example: 2

5.1.5 Match scale

When the driver service is registered by the driver bundle, the Device Manager calls Driver#match() with the argument of the ~~USBDevice~~USBInfoDevice service's ServiceReference. The driver responds with the value based on below scale.

- MATCH_VERSION - Constant for the USB device match scale, indicating a match with usbinfo.idVendor, usbinfo.idProduct and usbinfo.bcdDevice. Value is 50.
- MATCH_MODEL - Constant for the USB device match scale, indicating a match with ~~USB.device~~usbinfo.idVendor and ~~USB.device~~usbinfo.idProduct. Value is 40~~10~~.

- MATCH_PROTOCOL - Constant for the USB device match scale, indicating a match with ~~USB.device~~usbinfo.bDeviceClass, ~~USB.device~~usbinfo.bDeviceSubClass and ~~USB.device~~usbinfo.bDeviceProtocol, or a match with usbinfo.bInterfaceClass, usbinfo.bInterfaceSubClass and usbinfo.bInterfaceProtocol.~~*bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of USB.device.interfaceclasses.*~~ Value is 307.

- MATCH_SUBCLASS - Constant for the USB device match scale, indicating a match ~~USB.device~~usbinfo.bDeviceClass and ~~USB.device~~usbinfo.bDeviceSubClass, or a match with usbinfo.bInterfaceClass and usbinfo.bInterfaceSubClass.~~*bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses.*~~ Value is 205.

- MATCH_CLASS - Constant for the USB device match scale, indicating a match with ~~USB.device~~usbinfo.bDeviceClass, or a match with usbinfo.bInterfaceClass.~~*bInterfaceClass in one of USB.device.interfaceclasses.*~~ Value is 103.

5.1.6 Operations

Figure 5 describes a mechanism to handle USB devices. When a USB device is attached, a ~~USB base driver~~USB information base driver bundle recognizes it via native device drivers and gets information from the USB device. The ~~USB base driver~~USB information base driver bundle registers a ~~USBDevice~~USBInfoDevice service with service properties from the information gained.

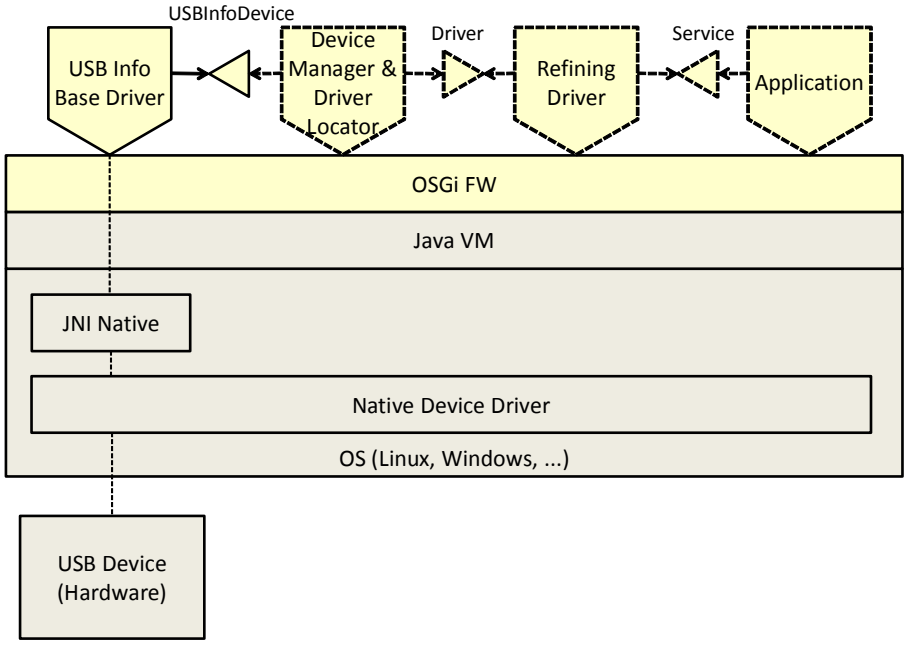

Fig 5:Device attachment example

5.2 USB Serial

This section is defined for USB devices have a serial communication function.

5.2.1 Assumptions

Many residential gateways have USB interfaces. And USB dongles support several protocols for home networks, such as ZigBee, Z-Wave, KNX/EHS, ECHONET, ECHONET-LITE, etc. Most USB dongles can be controlled through Serial Communication. When a USB dongle is connected, the device is mapped to a COM port automatically by native libraries in OS. This mechanism is out of scope, those libraries are preconditions.

5.2.2 Optional Device Access Category

In this section, an optional device access category is defined.

- ~~USBDevice~~USBInfoDevice.DEVICE_CATEGORY_SERIAL - OPTIONAL property. The value is "SerialInfo". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which has a serial communication function such as a USB dongle. The ~~USB base driver~~USB information base driver bundle must set this property key and serialinfo.comport property. This device category's value may be used independently of USB. This value is defined because some USB devices have a serial communication function.

5.2.3 Optional Service properties

Some other service properties are needed to identify and access a device by refining drivers.

Table 4: Optional service properties

Service property	M/O	Java type
serialinfo.comport	O	String

- serialinfo.comport - OPTIONAL Property key. The value is String. The USB Device has a serial communication function, set the value that represents the COM port. If the USB device does not have a serial communication function, this key and value are not set. The driver can communicate through Java Communications API with this value. Set this value "portName" of javax.comm.CommPortIdentifier#getPortIdentifier(String portName). Then serial communication is possible. Serial.comport value's format must be equal to the "portName" format. If a ~~USB base driver~~USB information base driver sets this property, ~~USBDevice~~USBInfoDevice.DEVICE_CATEGORY_SERIAL must be set to DEVICE_CATEGORY.
  - Example1: "/dev/ttyUSB0"
  - Example2: "COM5"
  - Example3: "/dev/tty.usbserial-XXXXXX"

5.2.4 Operations

5.2.4.1 USB serial example 1

Figure 6 describes a sample operation to handle USB serial devices.

0. A ~~USB base driver~~USB information base driver is tracking OS events. Native device driver such as kernel modules in Linux can detect and communicate with USB devices and allocate a USB device or a USB interface to a device file (COM port).

1. USB dongle 1 is connected, native device drivers allocate the device to /dev/ttyUSB0, the ~~USB base driver~~USB information base driver catches the event and gets information about the device, then the ~~USB base driver~~USB information base driver registers a ~~USBDevice~~USBInfoDevice service with service properties that have DEVICE_CATEGORY ({"USB", "Serial"}), serial.comport, bDeviceClass, idVendor and so on. NOTE; the ~~USB base driver~~USB information base driver must handle file.separator properly.

2. USB dongle 1 is removed, native device drivers remove it, the ~~USB base driver~~USB information base driver unregister the ~~USBDevice~~USBInfoDevice service.

3. USB dongle 1 (same device) is connected again, native device driver allocate it to /dev/ttyUSB2 (not same path), the ~~USB base driver~~USB information base driver should get information about that device again and registers ~~USBDevice~~USBInfoDevice service. NOTE; It is not guaranteed that the same device is the same COM port.

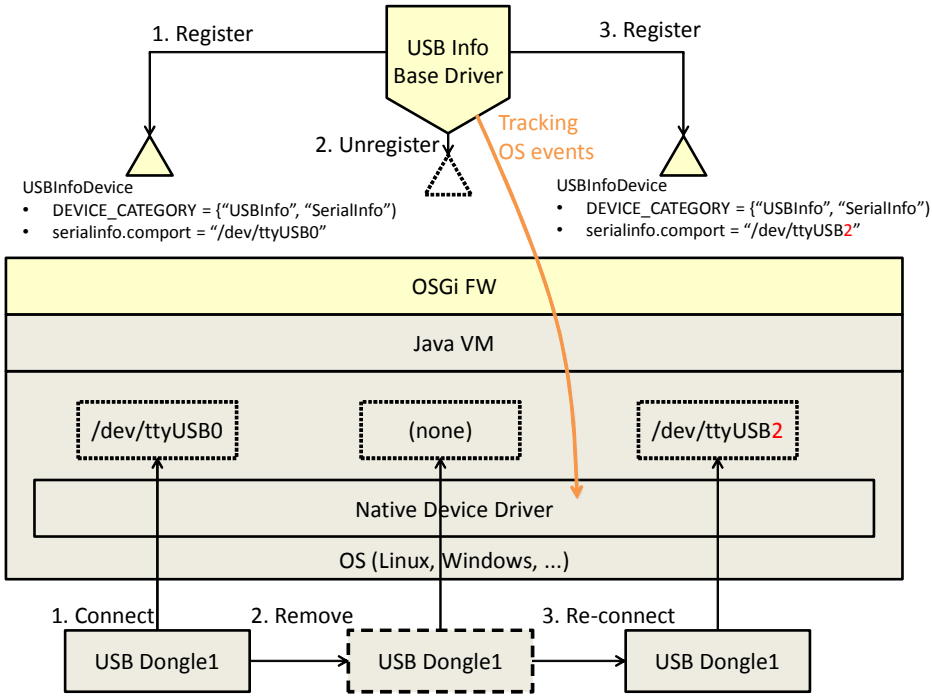

Fig 6: USB Serial Example 1

5.2.4.2 USB serial example 2

Figure 7 describes another sample operation to handle USB Serial devices.

0. Same as example 1.

1. Same as example 1.

2. Same as example 1.

3. USB dongle 2 (not same device) is connected, native device driver allocate it to /dev/ttyUSB0 (same path), the ~~USB base driver~~USB information base driver registers ~~USBDevice~~USBInfoDevice service. NOTE; It is not guaranteed that same COM port is same device.

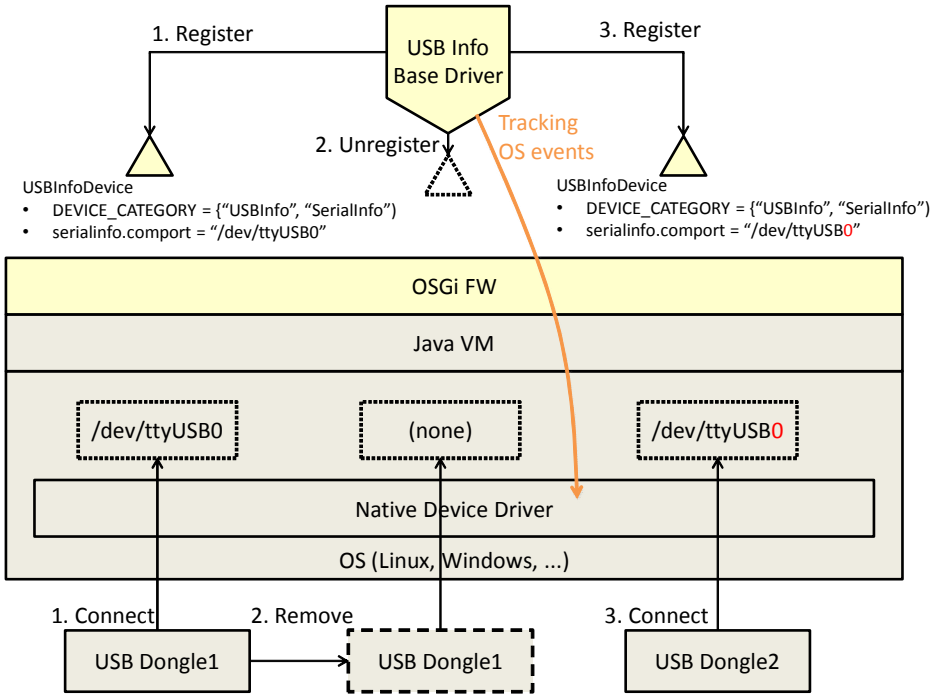

Fig 7: USB Serial Example 2

5.3 Mass Storage

This section is defined for USB devices that are Mass Storage.

5.3.1 Assumptions

When a USB storage is connected, the directory is mounted automatically or manually via some native libraries in OS. This mechanism is out of scope, those libraries are preconditions.

5.3.2 Optional Device Access Category

In this section, an optional device access category is defined.

- ~~USBDevice~~USBInfoDevice.DEVICE_CATEGORY_MASSSTORAGE - OPTIONAL property. The value is "MassStorageInfo". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which is a MassStorage Class in USB Specification[4]. such as a USB storage. Such a ~~USB base driver~~USB information base driver bundle must set this property key and set massstorageinfo.mountpoints property while the device is mounted.

5.3.3 Optional Service properties

Some other service properties are needed to identify and access a device by refining drivers.

Table 5: Optional service properties

Service property	M/O	Java type
massstorageinfo.mountpoints	O	String[]

- massstorageinfo.mountpoints - OPTIONAL property key. The value is String[]. If the USB device is Mass Storage Class, set the value that represents the mount point (a path to the USB storage) in OS. If the USB device is not Mass Storage Class, this key and value are not set. The driver can read and write the USB storage through standard File API with this value. Set this value "pathname" of java.io.File(String pathname). Then file access is possible. mMassstorageinfo.mountpoints's format must be equal to the "pathname" format. If a USB base driverUSB information base driver sets this property, USBDeviceUSBInfoDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY.
  - Example1: {"/mnt/media/usb-storage-01/"}
  - Example2: {"D:\\Java"}

5.3.4 Operations

5.3.4.1 USB storage example 1

Figure 8 describes a sample operation to handle USB storage devices.

0. A USB base driverUSB information base driver is tracking OS events. There are native device drivers and native libraries in OS. Native device drivers such as kernel modules in Linux can detect and communicate with USB device or USB and allocate it to a device file. Native libraries such as autofs in Linux auto-mount certain virtual devices to mount points.
1. USB storage 1 is connected, native device drivers allocate it to /dev/sda and native libraries auto-mount the virtual device to /mnt/usb0, the USB base driverUSB information base driver catches the mount event and gets information about the device, then the USB base driverUSB information base driver registers a USBDeviceUSBInfoDevice service with service properties that have DEVICE_CATEGORY ({"USB", "MassStorage"}), massstorage.mountpoints, bDeviceClass, idVendor and so on. NOTE; the USB base driverUSB information base driver must handle file.separator properly.
2. A user unmount /mnt/usb0 manually, the USB base driverUSB information base driver modifies the service property, massstorage.mountpoints is removed.
3. USB storage 1 is removed, native libraries remove the virtual device, the USB base driverUSB information base driver unregister the USBDeviceUSBInfoDevice service.
4. USB storage 1 (same device) is connected again, native device drivers allocate it to /dev/sdb (not same path) and native libraries mount it to /mnt/usb3 (not same path), the USB base driverUSB information base driver should get information about that device again and register USBDeviceUSBInfoDevice service. NOTE; It is not guaranteed that same device is same mountpoint.
5. A user re-mounts /mnt/usb3 to /mnt/myusb, the USB base driverUSB information base driver modifies the service property, massstorage.mountpoints is changed from {"/mnt/usb3"} to {"/mnt/myusb"}.

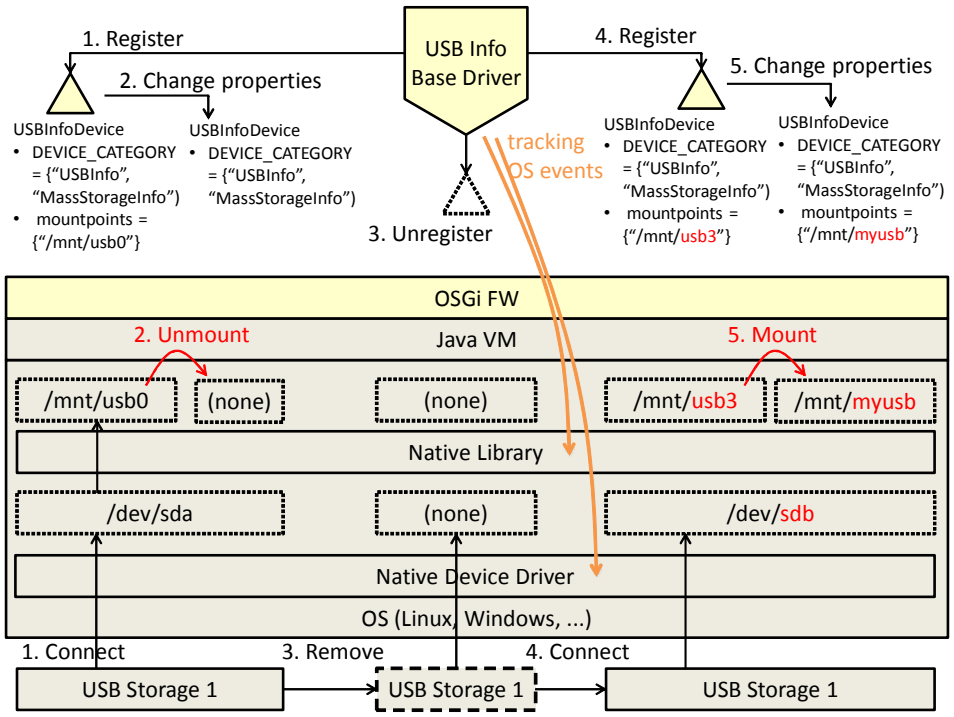

Fig 8: USB Storage Example 1

5.3.4.2 USB storage example 2

Figure 9 describes another sample operation to handle USB storage devices.

0. Same as example 1.

1. USB storage 1 that has 3 partitions is connected, native device drivers allocate each partition to /dev/sda1, /dev/sda2 and /dev/sda3 and native libraries auto-mount each virtual device to /mnt/usb0, none (not mounted) and /mnt/mmyusb, the ~~USB base driver~~USB information base driver catches the mount event and gets information about the device, then the ~~USB base driver~~USB information base driver registers a ~~USBDevice~~USBInfoDevice service with service properties that have DEVICE_CATEGORY, massstorage.mountpoints (`{"/mnt/usb0", "/mnt/myusb"}`), bDeviceClass, idVendor and so on. NOTE; the ~~USB base driver~~USB information base driver must handle file.separator properly.

2. A user unmount /mnt/myusb (/dev/sda3) manually, the ~~USB base driver~~USB information base driver modifies the service property, `massstorage.mountpoints` is changed to `{"/mnt/usb0"}`.

3. A user mount /dev/sda2 (not same partition) to /mnt/myusb (same mountpoint), the ~~USB base driver~~USB information base driver modifies the service property, `massstorage.mountpoints` is changed to `{"/mnt/usb0", "/mnt/myusb"}`. NOTE; It is not guaranteed that same mountpoint is same device. If USB Storage has multiple partitions, `massstorage.mountpoints` does not indicates which mountpoint represents which partition.

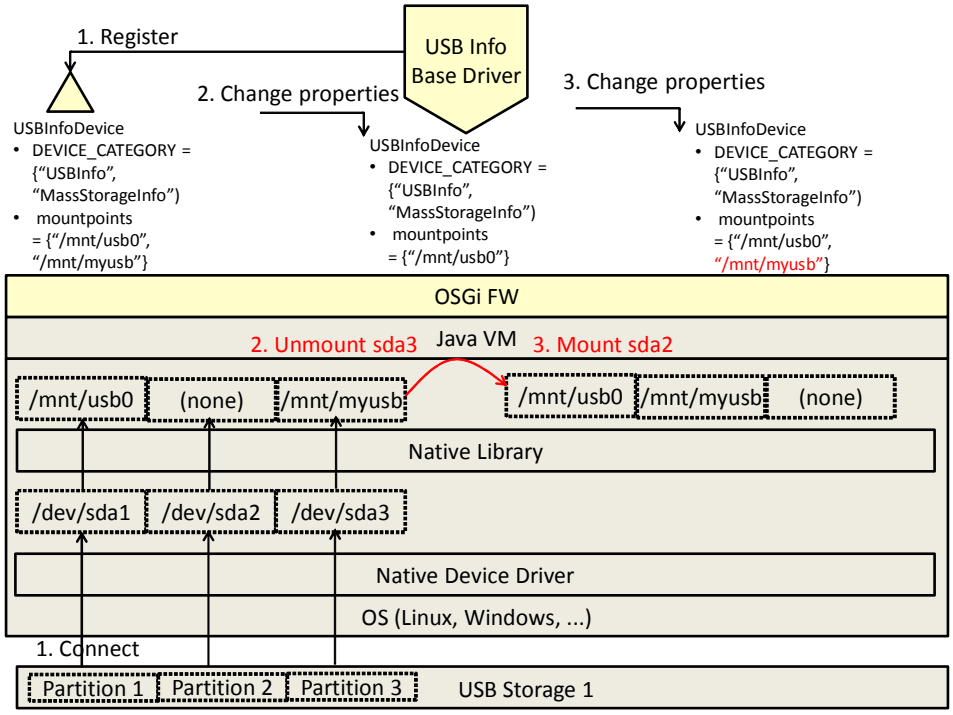

Fig 9: USB Storage Example 2

5.3.4.3 USB storage example 3

Figure 10 describes another sample operation to handle USB storage devices.

0. A ~~USB base driver~~USB information base driver is tracking OS events. Native device drivers such as kernel modules in Linux can detect and communicate with USB device and allocate it to a device file. Native libraries such as autofs in Linux auto-mount certain virtual devices to mount points. In this example, there are native device drivers in OS and native libraries are not installed.

1. USB storage 1 is connected, native device drivers allocate it to /dev/sda, the ~~USB base driver~~USB information base driver catches the event and gets information about the device, then the ~~USB base driver~~USB information base driver registers a ~~USBDevice~~USBInfoDevice service with service properties that have DEVICE_CATEGORY ({"USB", "MassStorage"}), bDeviceClass, idVendor and so on. NOTE; the ~~USB base driver~~USB information base driver must handle file.separator properly.

2. A user installs and starts the native libraries.

3. Those libraries mount the virtual device to /mnt/usb0, the ~~USB base driver~~USB information base driver catches the mount event and modifies the service property, `massstorage.mountpoints` (`{"/mnt/usb0"}`) is added.

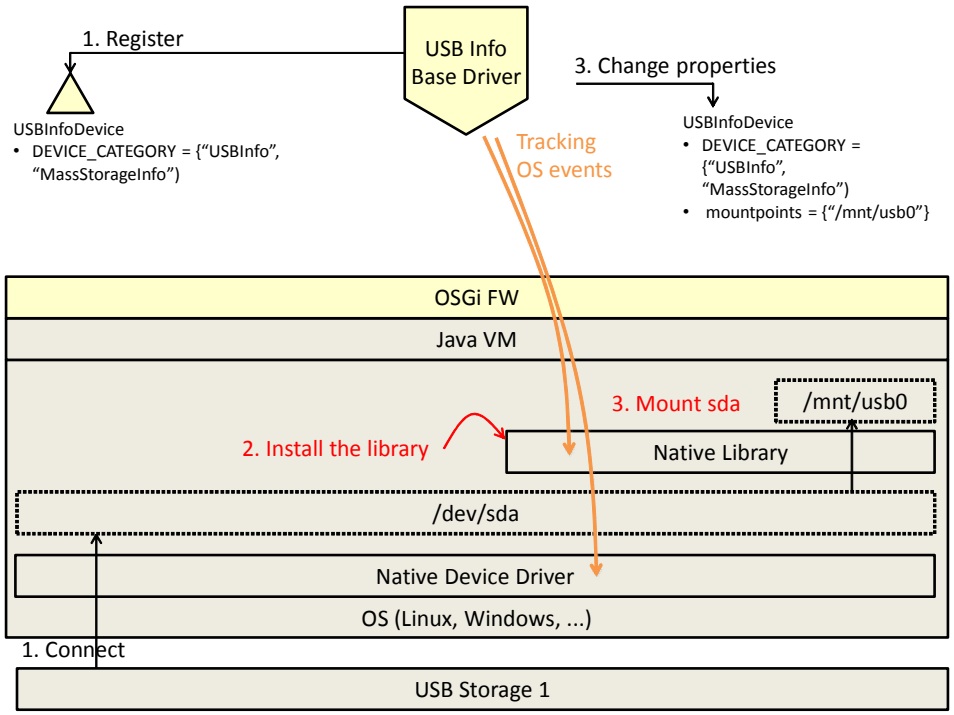

Fig 10: USB Storage Example 3

6 Data Transfer Objects

This RFC will not provide Data Transfer Objects.

7 Javadoc

OSGi Javadoc

9/18/14 5:06 PM

Package Summary		Page
org.osgi.service.usbinfo	-	25

Package org.osgi.service.usbinfo

Interface Summary		Page
USBInfoDevice	Represents a USB device.	*26*

Interface USBInfoDevice

org.osgi.service.usbinfo

---

public interface **USBInfoDevice**

Represents a USB device. For each USB device, an object is registered with the framework under the USBInfoDevice interface. A USB information base driver must implement this interface. The values of the USB property names are defined by the USB Implementers Forum, Inc. The package name is org.osgi.service.usbinfo.

Field Summary		Page
String	**ALTERNATE_SETTING** OPTIONAL property key.	29
String	**COM_PORT** OPTIONAL Property key.	31
String	**DEVICE_CATEGORY** MANDATORY property.	27
String	**DEVICE_CATEGORY_MASSSTORAGE** OPTIONAL Property.	32
String	**DEVICE_CATEGORY_SERIAL** OPTIONAL Property.	31
String	**DEVICE_CLASS** MANDATORY property key.	28
String	**DEVICE_MAXPACETSIZE0** OPTIONAL property key.	28
String	**DEVICE_PROTOCOL** MANDATORY property key.	28
String	**DEVICE_SUBCLASS** MANDATORY property key.	28
String	**INTERFACE_CLASS** MANDATORY property key.	29
String	**INTERFACE_DESCRIPTION** OPTIONAL property key.	30
String	**INTERFACE_NUMBER** MANDATORY property key.	29
String	**INTERFACE_PROTOCOL** MANDATORY property key.	30
String	**INTERFACE_SUBCLASS** MANDATORY property key.	30
String	**MANUFACTURER** OPTIONAL property key.	29
int	**MATCH_CLASS** Constant for the USB device match scale, indicating a match with usbinfo.bDeviceClass, or a match with bInterfaceClass in one of usbinfo.interfaceclasses.	31
int	**MATCH_MODEL** Constant for the USB device match scale, indicating a match with usbinfo.idVendor and usbinfo.idProduct.	31

int	**MATCH_PROTOCOL** Constant for the USB device match scale, indicating a match with usbinfo.bDeviceClass, usbinfo.bDeviceSubClass and usbinfo.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of usbinfo.interfaceclasses.	31
int	**MATCH_SUBCLASS** Constant for the USB device match scale, indicating a matchusbinfo.bDeviceClass and usbinfo.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of usbinfo.interfaceclasses.	31
int	**MATCH_VERSION** Constant for the USB device match scale, indicating a match with usbinfo.idVendor, usbinfo.idProduct and usbinfo.bcdDevice.	30
String	**MOUNTPOINTS** OPTIONAL Property key.	32
String	**NUM_CONFIGURATIONS** OPTIONAL property key.	29
String	**NUM_ENDPOINTS** OPTIONAL property key.	29
String	**PID** MANDATORY property key.	28
String	**PRODUCT** OPTIONAL property key.	29
String	**RELEASE_NUMBER** MANDATORY property key.	28
String	**SERIALNUMBER** OPTIONAL property key.	29
String	**USB_ADDR** MANDATORY property key.	30
String	**USB_BUS** MANDATORY property key.	30
String	**USB_RELEASE_NUMBER** OPTIONAL property key.	27
String	**VID** MANDATORY property key.	28

Field Detail

DEVICE_CATEGORY

```
public static final String DEVICE_CATEGORY = "USB"
```

MANDATORY property. The value is "USBInfo". Constant for the value of the service property DEVICE_CATEGORY used for all USB devices. A USB information base driver bundle must set this property key. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

USB_RELEASE_NUMBER

```
public static final String USB_RELEASE_NUMBER = "usbinfo.bcdUSB"
```

OPTIONAL property key. Value is "usbinfo.bcdUSB". The value is String, the 4-digit BCD format. Example: "0210"

DEVICE_CLASS

```
public static final String DEVICE_CLASS = "usbinfo.bDeviceClass"
```

MANDATORY property key. Value is "usbinfo.bDeviceClass". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_SUBCLASS

```
public static final String DEVICE_SUBCLASS = "usbinfo.bDeviceSubClass"
```

MANDATORY property key. Value is "usbinfo.bDeviceSubClass". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_PROTOCOL

```
public static final String DEVICE_PROTOCOL = "usbinfo.bDeviceProtocol"
```

MANDATORY property key. Value is "usbinfo.bDeviceProtocol". The value is String, hexadecimal, 2-digits. Example: "ff"

DEVICE_MAXPACETSIZE0

```
public static final String DEVICE_MAXPACETSIZE0 = "usbinfo.bMaxPacketSize0"
```

OPTIONAL property key. Value is "usbinfo.bMaxPacketSize0". The value is Integer.

VID

```
public static final String VID = "usbinfo.idVendor"
```

MANDATORY property key. Value is "usbinfo.idVendor". The value is String, hexadecimal, 4-digits. Example: "0403"

PID

```
public static final String PID = "usbinfo.idProduct"
```

MANDATORY property key. Value is "usbinfo.idProduct". The value is String, hexadecimal, 4-digits. Example: "8372"

RELEASE_NUMBER

```
public static final String RELEASE_NUMBER = "usbinfo.bcdDevice"
```

MANDATORY property key. Value is "usbinfo.bcdDevice". The value is String, the 4-digit BCD format. Example: "0200"

MANUFACTURER

```
public static final String MANUFACTURER = "usbinfo.Manufacturer"
```

OPTIONAL property key. Value is "usbinfo.Manufacturer". The value is String of indicated in iManufacturer. (The value is not the index.) Example: "Buffalo Inc."

PRODUCT

```
public static final String PRODUCT = "usbinfo.Product"
```

OPTIONAL property key. Value is "usbinfo.Product". The value is String of indicated in iProduct. (The value is not the index.) Example: "USB2.0 PC Camera"

SERIALNUMBER

```
public static final String SERIALNUMBER = "usbinfo.SerialNumber"
```

OPTIONAL property key. Value is "usbinfo.SerialNumber". The value is String of indicated in iSerialNumber. (The value is not the index.) Example: "57B0002600000001"

NUM_CONFIGURATIONS

```
public static final String NUM_CONFIGURATIONS = "usbinfo.bNumConfigurations"
```

OPTIONAL property key. Value is "usbinfo.bNumConfigurations". The value is Integer.

INTERFACE_NUMBER

```
public static final String INTERFACE_NUMBER = "usbinfo.bInterfaceNumber"
```

MANDATORY property key. Value is "usbinfo.bInterfaceNumber". The value is Integer.

ALTERNATE_SETTING

```
public static final String ALTERNATE_SETTING = "usbinfo.bAlternateSetting"
```

OPTIONAL property key. Value is "usbinfo.bAlternateSetting". The value is Integer.

NUM_ENDPOINTS

```
public static final String NUM_ENDPOINTS = "usbinfo.bAlternateSetting"
```

OPTIONAL property key. Value is "usbinfo.bAlternateSetting". The value is Integer.

INTERFACE_CLASS

```
public static final String INTERFACE_CLASS = "usbinfo.bInterfaceClass"
```

MANDATORY property key. Value is "usbinfo.bInterfaceClass". The value is String, hexadecimal, 2-digits. Example: "ff"

INTERFACE_SUBCLASS

```
public static final String INTERFACE_SUBCLASS = "usbinfo.bInterfaceSubClass"
```

MANDATORY property key. Value is "usbinfo.bInterfaceSubClass". The value is String, hexadecimal, 2-digits. Example: "ff"

INTERFACE_PROTOCOL

```
public static final String INTERFACE_PROTOCOL = "usbinfo.bInterfaceProtocol"
```

MANDATORY property key. Value is "usbinfo.bInterfaceProtocol". The value is String, hexadecimal, 2-digits. Example: "ff"

INTERFACE_DESCRIPTION

```
public static final String INTERFACE_DESCRIPTION = "usbinfo.Interface"
```

OPTIONAL property key. Value is "usbinfo.Interface". The value is String of indicated in iInterface. (The value is not the index.)

USB_BUS

```
public static final String USB_BUS = "usbinfo.bus"
```

MANDATORY property key. Value is "usbinfo.bus". The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB bus assigned when connecting the USB device. USB bus ID is integer. The USB bus ID does not change while the USB device remains connected. Example: 3

USB_ADDR

```
public static final String USB_ADDR = "usbinfo.address"
```

MANDATORY property key. Value is "usbinfo.address". The value is Integer. Used to identify USB devices with same VID / PID. The value is the ID of the USB address assigned when connecting the USB device. USB address is integer (001-127). The USB address does not change while the USB device remains connected. Example: 2

MATCH_VERSION

```
public static final int MATCH_VERSION = 50
```

Constant for the USB device match scale, indicating a match with usbinfo.idVendor, usbinfo.idProduct and usbinfo.bcdDevice. Value is 50.

MATCH_MODEL

```
public static final int MATCH_MODEL = 40
```

Constant for the USB device match scale, indicating a match with usbinfo.idVendor and usbinfo.idProduct. Value is 40.

MATCH_PROTOCOL

```
public static final int MATCH_PROTOCOL = 30
```

Constant for the USB device match scale, indicating a match with usbinfo.bDeviceClass, usbinfo.bDeviceSubClass and usbinfo.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of usbinfo.interfaceclasses. Value is 30.

MATCH_SUBCLASS

```
public static final int MATCH_SUBCLASS = 20
```

Constant for the USB device match scale, indicating a matchusbinfo.bDeviceClass and usbinfo.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of usbinfo.interfaceclasses. Value is 20.

MATCH_CLASS

```
public static final int MATCH_CLASS = 10
```

Constant for the USB device match scale, indicating a match with usbinfo.bDeviceClass, or a match with bInterfaceClass in one of usbinfo.interfaceclasses. Value is 10.

DEVICE_CATEGORY_SERIAL

```
public static final String DEVICE_CATEGORY_SERIAL = "SerialInfo"
```

OPTIONAL Property. The value is "SerialInfo". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which has a serial communication function such as a USB dongle. The USB information base driver bundle must set this property key and serialinfo.comport property. This device category's value may be used independently of USB. This value is defined because some USB devices have a serial communication function. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

COM_PORT

```
public static final String COM_PORT = "serialinfo.comport"
```

OPTIONAL Property key. Value is "serialinfo.comport". The property value is String. The USB Device has a serial communication function, set the value that represents the COM port. If the USB device does not have a serial communication function, this key and value are not set. The driver can communicate through Java Communications API with this value. Set this value "portName" of javax.comm.CommPortIdentifier#getPortIdentifier(String portName). Then serial communication is possible. serialinfo.comport value's format must be equal to the "portName" format. If a USB information base driver set this property, USBInfoDevice.DEVICE_CATEGORY_SERIAL must be set to

DEVICE_CATEGORY. Example1: "/dev/ttyUSB0". Example2: "COM5". Example3: "/dev/tty.usbserial-XXXXXX".

DEVICE_CATEGORY_MASSSTORAGE

```
public static final String DEVICE_CATEGORY_MASSSTORAGE = "MassStorageInfo"
```

OPTIONAL Property. The value is "MassStorageInfo". Constant for the value of the service property DEVICE_CATEGORY used for a USB device which is a MassStorage Class in USB Specification such as a USB storage. Such a USB information base driver bundle must set this property key and massstorageinfo.mountpoints property while the device is mounted. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

MOUNTPOINTS

```
public static final String MOUNTPOINTS = "massstorageinfo.mountpoints"
```

OPTIONAL Property key. Value is "massstorageinfo.mountpoints". The property value is String[]. If the USB device is Mass Storage Class, set the value that represents the mount point (a path to the USB storage) in OS. If the USB device is not Mass Storage Class, this key and value are not set. The driver can read and write the USB storage through standard File API with this value. Set this value "pathname" of java.io.File(String pathname). Then file access is possible. massstorageinfo.mountpoints's format must be equal to the "pathname" format. If a USB information base driver set this property, USBInfoDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY. Example1: {"/mnt/media/usb-storage-01/"}. Example2: {"D:\\Java"}.

---

8 Considered Alternatives

8.1 ~~USB.device~~usbinfo.interfaceclasses

The alternative format of ~~USB.device~~usbinfo.interfaceclasses is below.

- ~~USB.device~~usbinfo.interfaceclasses - MANDATORY property key. The property value is an array of String for each interface descriptor. Each String value is connected with "_" interface descriptor's *bInterfaceClass*, *bInterfaceSubClass*, and *bInterfaceProtocol*. If there is no subclass code associated with the class code, does not connect subclass code and protocol code. If there is no protocol code associated with the class code, the protocol code is not connected. In addition, if the class code is vendor-specific class, does not connect subclass code and protocol code. Set only the class code. (See Table 6.)

- Example: An example of a USB device that has 2 interfaces. The first class code is CDC, subclass code is ACM(without protocol code). The second class code is CDC-Data (no subclass code and protocol code).

  - Value: "CDC_ACM", "CDC-Data"

Table 6: Class Code and Service Property

	Class Code	SubClass Code	Protocol Code	Representation in Service Property
1	01	01	any	Audio_AudioControl
2	01	02	any	Audio_AudioStreaming
3	01	03	any	Audio_MidiStreaming
4	01	other than above	any	Audio
5	02	01	01	CDC_DLCM_V.250
6	02	01	other than above	CDC_DLCM
7	02	02	01	CDC_ACM_V.250
8	02	02	other than above	CDC_ACM
9	02	03	01	CDC_TCM_V.250
10	02	03	other than above	CDC_TCM
11	02	04	any	CDC_MCCM
12	02	05	any	CDC_CAPI
13	02	06	any	CDC_ENCM
14	02	07	any	CDC_ATM
15	02	08	02	CDC_WHCM_PCCA-101
16	02	08	03	CDC_WHCM_PCCA-101-AnnexO
17	02	08	04	CDC_WHCM_GSM7.07
18	02	08	05	CDC_WHCM_3GPP27.007
19	02	08	06	CDC_WHCM_TIA-CDMA
20	02	08	FE	CDC_WHCM_ExternalProtocol
21	02	08	other than above	CDC_WHCM
22	02	09	02	CDC_DM_PCCA-101
23	02	09	03	CDC_DM_PCCA-101-AnnexO
24	02	09	04	CDC_DM_GSM7.07
25	02	09	05	CDC_DM_3GPP27.007
26	02	09	06	CDC_DM_TIA-CDMA
27	02	09	FE	CDC_DM_ExternalProtocol
28	02	09	other than above	CDC_DM
29	02	0A	02	CDC_MDLM_PCCA-101
30	02	0A	03	CDC_MDLM_PCCA-101-AnnexO
31	02	0A	04	CDC_MDLM_GSM7.07
32	02	0A	05	CDC_MDLM_3GPP27.007
33	02	0A	06	CDC_MDLM_TIA-CDMA
34	02	0A	FE	CDC_MDLM_ExternalProtocol
35	02	0A	other than above	CDC_MDLM
36	02	0B	02	CDC_OBEX_PCCA-101
37	02	0B	03	CDC_OBEX_PCCA-101-AnnexO
38	02	0B	04	CDC_OBEX_GSM7.07
39	02	0B	05	CDC_OBEX_3GPP27.007
40	02	0B	06	CDC_OBEX_TIA-CDMA
41	02	0B	FE	CDC_OBEX_ExternalProtocol
42	02	0B	other than above	CDC_OBEX
43	02	0C	07	CDC_EEM_EEM
44	02	0C	other than above	CDC_EEM
45	02	0D	FE	CDC_NCM_ExternalProtocol
46	02	0D	other than above	CDC_NCM
47	02	other than above	any	CDC
48	03	01	01	HID_Boot_KeyBoard
49	03	01	02	HID_Boot_Mouse
50	03	01	other than above	HID_Boot
51	03	other than above	any	HID

52	05	any	any	Physical
53	06	01	01	Image_Capture_PIMA15740
54	06	01	other than above	Image_Capture
55	06	other than above	any	Image
56	07	01	01	Printer_Printer_Unidirectional
57	07	01	02	Printer_Printer_Bi-directional
58	07	01	03	Printer_Printer_1284.4
59	07	01	FF	Printer_Printer_VendorSpecific
60	07	01	other than above	Printer_Printer
61	07	other than above	any	Printer
62	08	00	any	MassStorage_SCSI-nr
63	08	01	any	MassStorage_RBC
64	08	02	00	MassStorage_MMC-5_CBI-cci
65	08	02	01	MassStorage_MMC-5_CBI-ncci
66	08	02	50	MassStorage_MMC-5_BBB
67	08	02	other than above	MassStorage_MMC-5
68	08	04	00	MassStorage_UFI_CBI-cci
69	08	04	01	MassStorage_UFI_CBI-ncci
70	08	04	other than above	MassStorage_UFI
71	08	06	50	MassStorage_SCSI_BBB
72	08	06	62	MassStorage_SCSI_UAS
73	08	06	other than above	MassStorage_SCSI
74	08	07	any	MassStorage_LSDFS
75	08	08	any	MassStorage_IEEE1667
76	08	FF	00	MassStorage_VendorSpecific_CBI-cci
77	08	FF	01	MassStorage_VendorSpecific_CBI-ncci
78	08	FF	50	MassStorage_VendorSpecific_BBB
79	08	FF	other than above	MassStorage_VendorSpecific
80	08	other than above	any	MassStorage
81	0A	00	01	CDC-Data_Data_NTB
82	0A	00	30	CDC-Data_Data_I.430
83	0A	00	31	CDC-Data_Data_HDLC
84	0A	00	32	CDC-Data_Data_Transparent
85	0A	00	50	CDC-Data_Data_Q.921M
86	0A	00	51	CDC-Data_Data_Q.921
87	0A	00	52	CDC-Data_Data_Q.921TM
88	0A	00	90	CDC-Data_Data_V.42bis
89	0A	00	91	CDC-Data_Data_Euro-ISDN
90	0A	00	92	CDC-Data_Data_V.120
91	0A	00	93	CDC-Data_Data_CAPI2.0
92	0A	00	FD	CDC-Data_Data_HostBasedDriver
93	0A	00	FE	CDC-Data_Data_PUFD
94	0A	00	FF	CDC-Data_Data_VendorSpecific
95	0A	00	other than above	CDC-Data_Data
96	0A	other than above	any	CDC-Data
97	0B	00	00	SmartCard_SmartCard_CCID
98	0B	00	01	SmartCard_SmartCard_ICC-A
99	0B	00	02	SmartCard_SmartCard_ICC-B
100	0B	00	other than above	SmartCard_SmartCard
101	0B	other than above	any	SmartCard
102	0D	00	00	ContentSecurity
103	0E	01	any	Video_VideoControl
104	0E	02	any	Video_VideoStreaming
105	0E	03	any	Video_VideoInterfaceCollection
106	0E	other than above	any	Video
107	0F	any	any	PersonalHealthcareDevice
108	10	01	any	AudioVideoDevice_AVControlInterface
109	10	02	any	AudioVideoDevice_ AVDataVideoStreamingInterface
110	10	03	any	AudioVideoDevice_

				AVDataAudioStreamingInterface
111	10	other than above	any	AudioVideoDevice
112	DC	any	any	DiagnosticDevice
113	E0	01	01	WirelessController_Wireless_Bluetooth
114	E0	01	02	WirelessController_Wireless_UWB
115	E0	01	03	WirelessController_Wireless_RemoteNDIS
116	E0	01	04	WirelessController_Wireless_ BluetoothAMPController
117	E0	01	other than above	WirelessController_Wireless
118	E0	02	01	WirelessController_WireAdapter_Host
119	E0	02	02	WirelessController_WireAdapter_Device
120	E0	02	03	WirelessController_WireAdapter_ DeviceIsochronous
121	E0	02	other than above	WirelessController_WireAdapter
122	E0	other than above	any	WirelessController
123	EF	01	any	Miscellaneous_Sync
124	EF	03	any	Miscellaneous_CBAF
125	EF	other than above	any	Miscellaneous
126	FE	01	any	ApplicationSpecific_FirmwareUpgrade
127	FE	02	any	ApplicationSpecific_IrdaBridge
128	FE	03	01	ApplicationSpecific_TestMeasurement_ USB488
129	FE	03	other than above	ApplicationSpecific_TestMeasurement
130	FE	other than above	any	ApplicationSpecific
131	FF	any	any	VendorSpecific

9 Security Considerations

ServicePermission is needed when a bundle get ~~USBDevice~~USBInfoDevice service.

10 Document Support

10.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. OSGi Service Platform Service Compendium Release 4, Version 4.3 Device Access Specification, Version 1.1.

[4]. Universal Serial Bus Specification Revision 1.1, September 23, 1998.

10.2 Author's Address

Name	Yukio KOIKE
Company	NTT Corporation
Address	1-1, Hikari-no-oka, Yokosuka-shi, 238-0847, Kanagawa, Japan
Voice	+81 46 859 5142
e-mail	koike.yukio@lab.ntt.co.jp

10.3 Acronyms and Abbreviations

10.4 End of Document