RFC 202 – USB Device Category

Draft

25 Pages

Text in Red is here to help you. Delete it when you have followed the instructions.
The <RFC Title> and <Company> can be set from the File>Properties:User Defined menu. To update it onscreen, press F9. To update all of the fields in the document Select All (CTRL-A), then hit F9. Set the release level by selecting one from: Draft, Final Draft, Release. The date is set automagically when the document is saved. ***Note that FINAL DRAFT and RELEASE documents should only be released as PDF, NOT AS ODF DOCUMENTS***

Abstract

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

0 Document Information

0.1 Table of Contents

0.2 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.
```

0.3 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
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

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

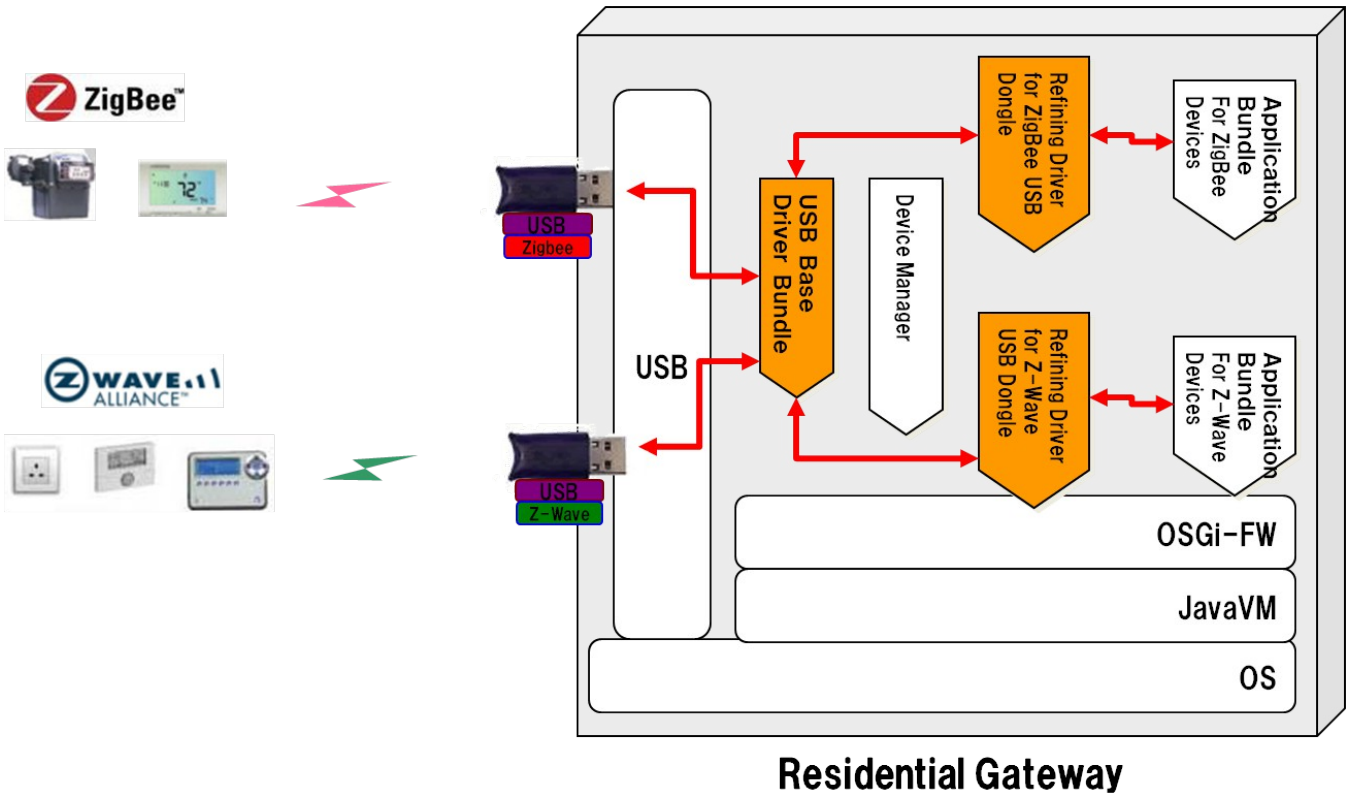


Fig 1. USB Dongles and Residential gateway

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].

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

5 Technical Solution

USB 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. This document does not define details of all USB classes, they are roles of refining drivers. Otherwise future specification about USB may define that.

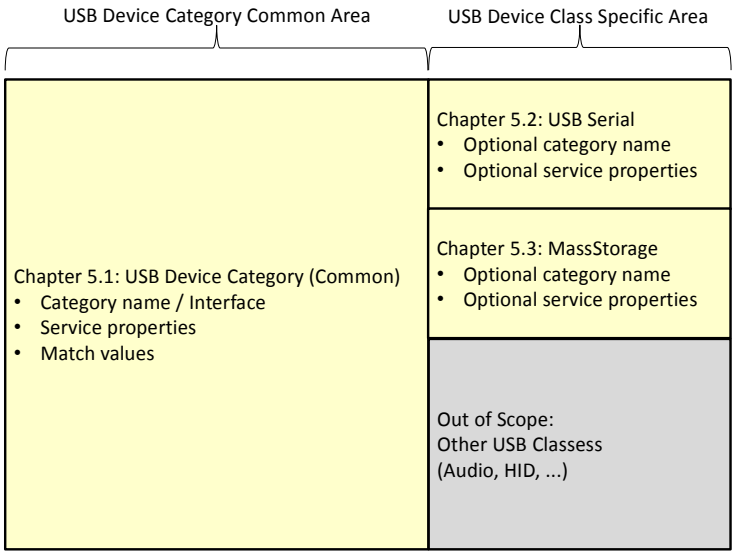


Fig 1:Structure of this document

5.1 USBDevice Service

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

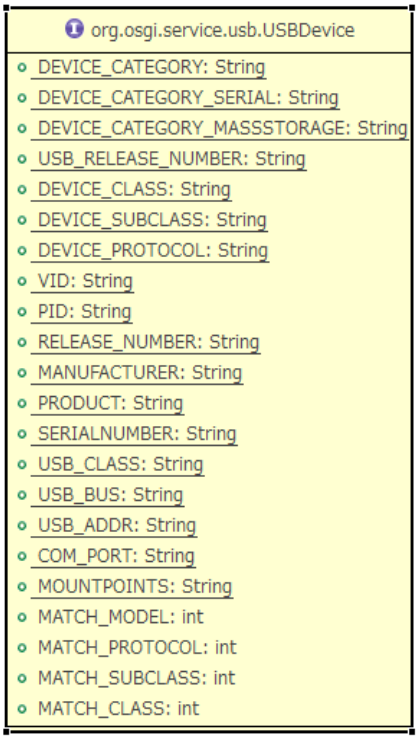


Fig 2:Class Diagram

5.1.1 Preconditions

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

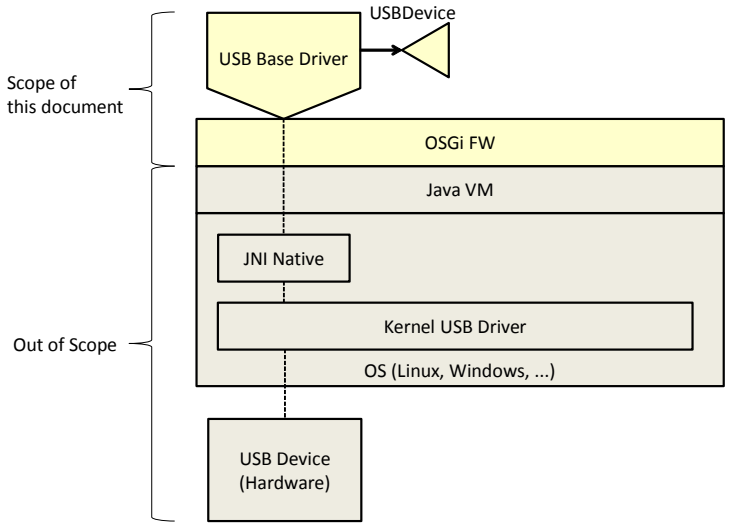


Fig 3:Software Structure

5.1.2 Device Access Category

The device access category is called “USB”. The category name is defined as a value of `USBDevice.DEVICE_CATEGORY` constant. It can be used as a part of `org.osgi.service.device.Constants.DEVICE_CATEGORY` service key value. The category impose this specification rules.

- USBDevice.DEVICE_CATEGORY – MANDATORY property. The value is “USB”. Constant for the value of the service property DEVICE_CATEGORY used for all USB devices. A USB base driver bundle must set this property key.

5.1.3 Service properties from USB Specification

Universal Serial Bus Specification (USB Specification) [4] defines a device descriptor. USB devices report their attributes using descriptors. USBDevice 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's service property	M/O	Java type
bLength	none	-	-
bDescriptorType	none	-	-
bcdUSB	USB.device.bcdUSB	M	int
bDeviceClass	USB.device.bDeviceClass	M	int
bDeviceSubClass	USB.device.bDeviceSubClass	M	int
bDeviceProtocol	USB.device.bDeviceProtocol	M	int
bMaxPacketSize0	none	-	-
idVendor	USB.device.idVendor	M	int
idProduct	USB.device.idProduct	M	int
bcdDevice	USB.device.bcdDevice	M	int
iManufacturer	USB.device.iManufacturer	O	String
iProduct	USB.device.iProduct	O	String
iSerialNumber	USB.device.iSerialNumber	O	String
bNumConfigurations	none	-	-

- `USB.device.bcdUSB` – MANDATORY property key. The value is int data type, the 4-digit BCD format.
  - Example: 0x0210
- `USB.device.bDeviceClass` – MANDATORY property key. The value is int data type, hexadecimal, 2-digits.
  - Example: 0xff
- `USB.device.bDeviceSubClass` – MANDATORY property key. The value is int data type, hexadecimal, 2-digits.
  - Example: 0xff
- `USB.device.bDeviceProtocol` – MANDATORY property key. The value is int data type, hexadecimal,

2-digits.
  - Example: 0xff
- `USB.device.idVendor` – MANDATORY property key. The value is int data type, hexadecimal, 4-digits.
  - Example: 0x0403
- `USB.device.idProduct` – MANDATORY property key. The value is int data type, hexadecimal, 4-digits.
  - Example : 0x8372
- `USB.device.bcdDevice` – MANDATORY property key. The value is int data type, the 4-digit BCD format.
  - Example : 0x0200
- `USB.device.iManufacturer` – OPTIONAL Property key. The value is string of indicated in *iManufacturer.* (The value is not the index.)
  - Example: “Buffalo Inc.”
- `USB.device.iProduct` – OPTIONAL Property key. The value is string of indicated in *iProduct.* (The value is not the index.)
  - Example : “USB2.0 PC Camera”
- `USB.device.iSerialNumber` – OPTIONAL Property key. The value is string of indicated in *iSerialNumber.* (The value is not the index.)
  - Example : “57B0002600000001”

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 service properties (see Table 2).

Table 2: Interface Descriptor and Service Property

Interface Descriptor's Field from USB Spec.	**USB Device Category's service property**	**M/O**	**Java type**
bLength	none	-	-
bDescriptorType	none	-	-
bInterfaceNumber	none	-	-
bAlternateSetting	none	-	-
bNumEndpoints	none	-	-
bInterfaceClass	USB.device.interfaceclasses	M	int+
bInterfaceSubClass			
bInterfaceProtocol			
iInterface	none	-	-

- `USB.device.interfaceclasses` – MANDATORY property key. The property value is int+, hexadecimal, 6-digits. Each int responds to each USB interface and is combinated the interface's bInterfaceClass (2-digits), bInterfaceSubClass (2-digits) and bInterfaceProtocol (2-digits).
  - Example: {0x080000, 0x0a00ff}

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.bus	M	int
USB.device.address	M	int

- `USB.device.bus` – MANDATORY property key. 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 int (001-127). The USB bus ID does not change while connecting the USB device.
  - Example: 3
- `USB.device.address` – MANDATORY property key. 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 int (001-127). The USB address does not change while connecting the USB device.
  - 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 service's ServiceReference. The driver answer the value based on below scale.

- `MATCH_MODEL` – Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct. Value is 10.

SUBCLASS – Constant for the USB device match scale, indicating a matchUSB.device.bDeviceClass and USB.device.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses. Value is 5.

- `MATCH_CLASS` – Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of `USB.device.interfaceclasses`. Value is 3.

5.1.6 Behavior

Figure 4 describes a mechanism to handle USB devices. When a USB device is attached, a USB base driver bundle recognizes it via native device drivers and gets information from the USB device. The USB base driver bundle registers a USBDevice service with service properties from got information. Then the Device Manager installs a driver bundle with the help of DriverLocator services, and starts it. When the driver service is registered by the driver bundle, the Device Manager calls Driver#match() and Driver#attach() with the argument of the USBDevice service's ServiceReference. Then the driver bundle finds out how to access the USB Device by the service properties and communicates with it through the associated API, such as Java Communications API and File API. When the driver bundle recognizes the USB device, the driver bundle may register services for applications such as ZigBee device service. Then, an application bundle can use those ZigBee Device services.

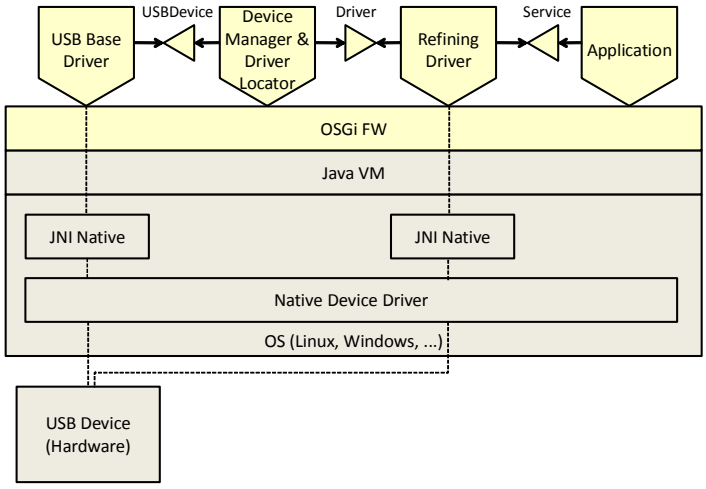


Fig 4:Device attachment example

5.2 USB Serial

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

5.2.1 Preconditions

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 of 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.DEVICE_CATEGORY_SERIAL – OPTIONAL property. The value is “Serial”. 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. Such a USB base driver bundle must set this property key and `USB.device.comport` property. This device category's value may be used independently of USB. This value is defined because of 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
serial.comport	O	String

- `serial.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 is 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. If a USB base driver set this property, USBDevice.DEVICE_CATEGORY_SERIAL must be set to DEVICE_CATEGORY.

- Example: "/dev/ttyUSB0"

5.2.4 Behavior

TBD.

- USB Base Driver tracks OS events.
- It is not guaranteed that same device is same comport.

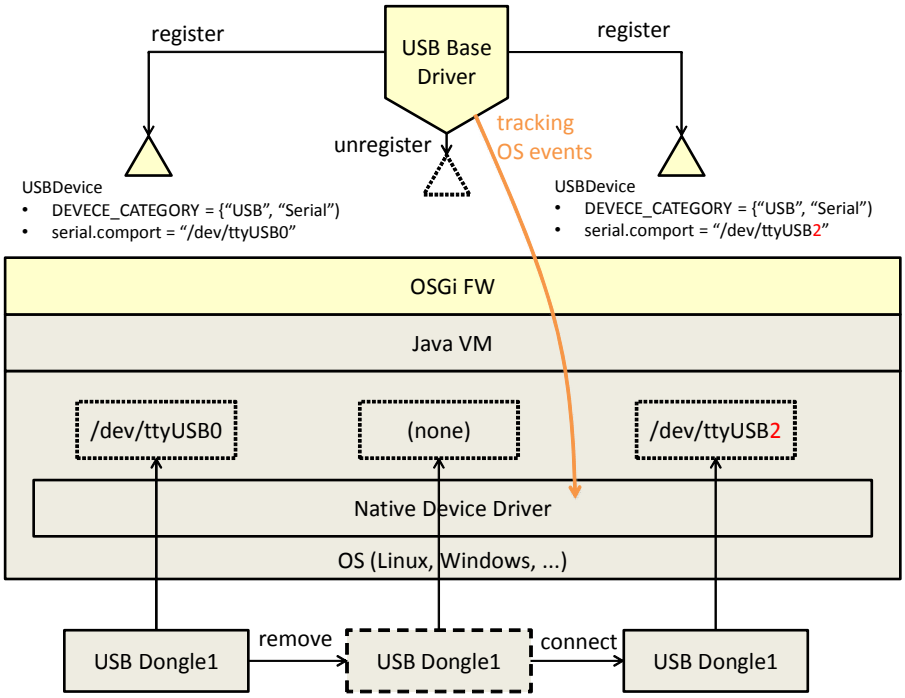


Fig 5:USB Serial Example

5.3 Mass Storage

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

5.3.1 Preconditions

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.DEVICE_CATEGORY_MASSSTORAGE – OPTIONAL property. The value is "MassStorage". 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 bundle must set this property key and `USB.device.mountpoint` property.

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

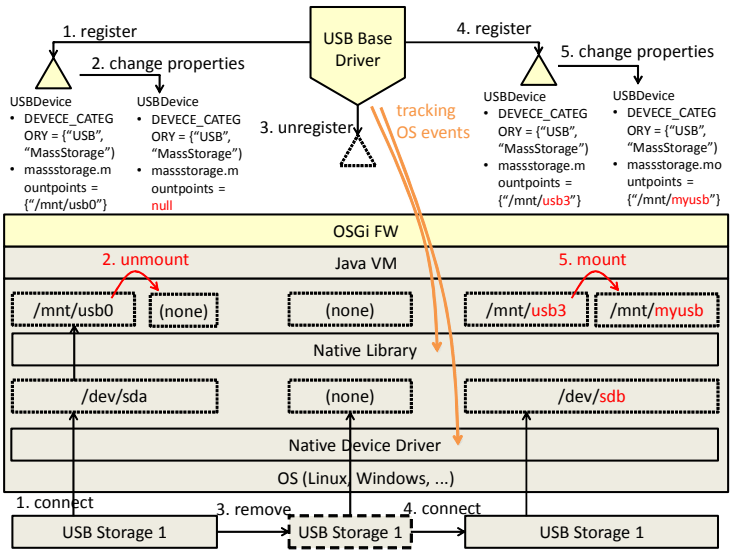
Service property	M/O	Java type
massstorage.mountpoints	O	String+

- `massstorage.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 is not set. The driver can read and write the USB storage through standard API such as File class. If a USB base driver set this property, USBDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY.
  - Example: {"/mnt/media/usb-storage-01/"}

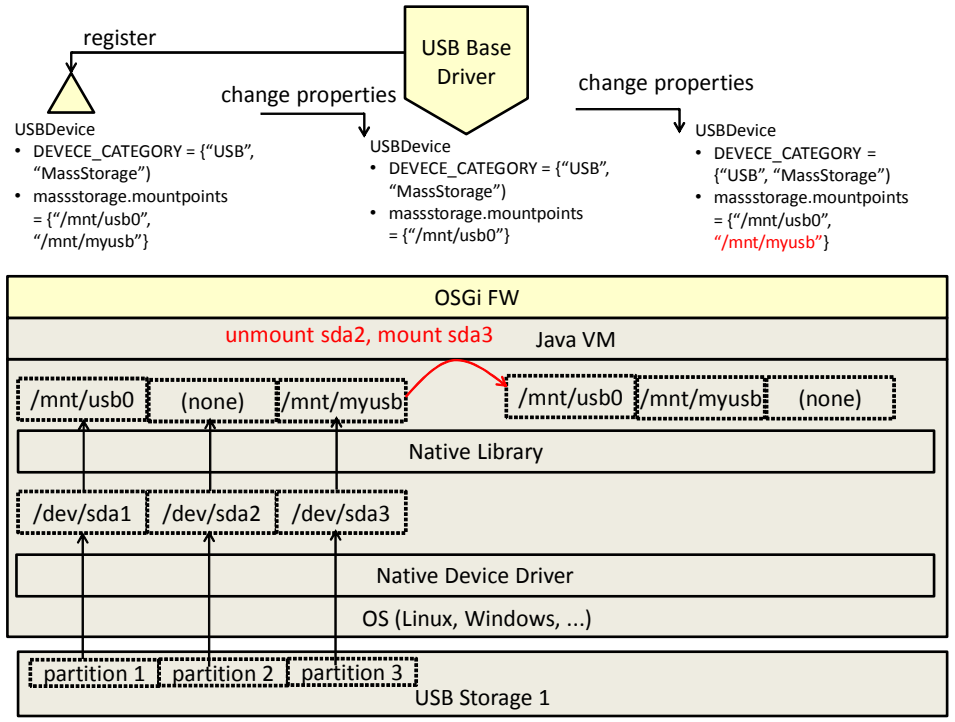
5.3.4 Behavior

TBD.

- USB Base Driver tracks OS events.
- USB Base Driver maps USB information (e.g. idvendor), OS device information (e.g. /dev/sda) and mountpoint information (e.g. /mnt/usb0).
- It is not guaranteed that same device is same mountpoint and same mountpoint is same device.



- If USB Storage has multiple partitions, massstorage.mountpoints does not indicates which mountpoint represents which partition.



6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

OSGi Javadoc

11/19/13 9:43 PM

Package Summary		Page
org.osgi.service.usb		18

Package org.osgi.service.usb

Interface Summary		Page
USBDevice	Represents a USB device.	19

Interface USBDevice

org.osgi.service.usb

public interface **USBDevice**

Represents a USB device. For each USB device, an object is registered with the framework under the USBDevice interface. A USB 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.usb.

Field Summary		Page
String	**COM_PORT** OPTIONAL Property key.	22
String	**DEVICE_CATEGORY** MANDATORY property.	20
String	**DEVICE_CATEGORY_MASSSTORAGE** OPTIONAL Property.	20
String	**DEVICE_CATEGORY_SERIAL** OPTIONAL Property.	20
String	**DEVICE_CLASS** MANDATORY property key.	20
String	**DEVICE_PROTOCOL** MANDATORY property key.	21
String	**DEVICE_SUBCLASS** MANDATORY property key.	21
String	**MANUFACTURER** OPTIONAL Property key.	21
int	**MATCH_CLASS** Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of USB.device.interfaceclasses.	23
int	**MATCH_MODEL** Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct.	23
int	**MATCH_PROTOCOL** Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, USB.device.bDeviceSubClass and USB.device.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of USB.device.interfaceclasses.	23
int	**MATCH_SUBCLASS** Constant for the USB device match scale, indicating a matchUSB.device.bDeviceClass and USB.device.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses.	23
String	**MOUNTPOINTS** OPTIONAL Property key.	23
String	**PID** MANDATORY property key.	21
String	**PRODUCT** OPTIONAL Property key.	22
String	**RELEASE_NUMBER** MANDATORY property key.	21
String	**SERIALNUMBER** OPTIONAL Property key.	22

String	**USB_ADDR** MANDATORY property key.	*22*
String	**USB_BUS** MANDATORY property key.	*22*
String	**USB_CLASS** MANDATORY property key.	*22*
String	**USB_RELEASE_NUMBER** MANDATORY property key.	*20*
String	**VID** MANDATORY property key.	*21*

Field Detail

DEVICE_CATEGORY

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

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

DEVICE_CATEGORY_SERIAL

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

OPTIONAL Property. The value is “Serial”. 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. Such a USB base driver bundle must set this property key and USB.device.comport property. This device category's value may be used independently of USB. This value is defined because of some USB devices have a serial communication function. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

DEVICE_CATEGORY_MASSSTORAGE

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

OPTIONAL Property. The value is “MassStorage”. 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 base driver bundle must set this property key and USB.device.mountpoint property. See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

USB_RELEASE_NUMBER

```
public static final String USB_RELEASE_NUMBER = "USB.device.bcdUSB"
```

MANDATORY property key. Value is "USB.device.bcdUSB". The value is int data type, the 4-digit BCD format. Example: 0x0210 *

DEVICE_CLASS

```
public static final String DEVICE_CLASS = "USB.device.bDeviceClass"
```

MANDATORY property key. Value is "USB.device.bDeviceClass". The value is int data type, hexadecimal, 2-digits. Example: 0xff

---

DEVICE_SUBCLASS

```
public static final String DEVICE_SUBCLASS = "USB.device.bDeviceSubClass"
```

MANDATORY property key. Value is "USB.device.bDeviceSubClass". The value is int data type, hexadecimal, 2-digits. Example: 0xff

---

DEVICE_PROTOCOL

```
public static final String DEVICE_PROTOCOL = "USB.device.bDeviceProtocol"
```

MANDATORY property key. Value is "USB.device.bDeviceProtocol". The value is int data type, hexadecimal, 2-digits. Example: 0xff

---

VID

```
public static final String VID = "USB.device.idVendor"
```

MANDATORY property key. Value is "USB.device.idVendor". The value is int data type, hexadecimal, 4-digits. Example: 0x0403

PID

```
public static final String PID = "USB.device.idProduct"
```

MANDATORY property key. Value is "USB.device.idProduct". The value is int data type, hexadecimal, 4-digits. Example : 0x8372

RELEASE_NUMBER

```
public static final String RELEASE_NUMBER = "USB.device.bcdDevice"
```

MANDATORY property key. Value is "USB.device.bcdDevice". The value is int data type, the 4-digit BCD format. Example : 0x0200

MANUFACTURER

```
public static final String MANUFACTURER = "USB.device.iManufacturer"
```

OPTIONAL Property key. Value is "iManufacturer". The value is string of indicated in iManufacturer. (The value is not the index.) Example: “Buffalo Inc.”

PRODUCT

```
public static final String PRODUCT = "USB.device.iProduct"
```

OPTIONAL Property key. Value is "iProduct". 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 = "USB.device.iSerialNumber"
```

OPTIONAL Property key. Value is "USB.device.iSerialNumber". The value is string of indicated in iSerialNumber. (The value is not the index.) Example : “57B0002600000001”

USB_CLASS

```
public static final String USB_CLASS = "USB.device.interfaceclassess"
```

MANDATORY property key. Value is "USB.device.interfaceclassess". The property value is int+, hexadecimal, 6-digits. Each int responds to each USB interface and is combinated the interface's bInterfaceClass (2-digits), bInterfaceSubClass (2-digits) and bInterfaceProtocol (2-digits). Example: {0x080000, 0x0a00ff}

USB_BUS

```
public static final String USB_BUS = "USB.device.bus"
```

MANDATORY property key. Value is "USB.device.bus". 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 int (001-127). The USB bus ID does not change while connecting the USB device. Example: 3

USB_ADDR

```
public static final String USB_ADDR = "USB.device.address"
```

MANDATORY property key. Value is "USB.device.address". 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 int (001-127). The USB address does not change while connecting the USB device. Example: 2

COM_PORT

```
public static final String COM_PORT = "USB.device.comport"
```

OPTIONAL Property key. Value is "USB.device.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 is 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. If a USB base driver set this property, USBDevice.DEVICE_CATEGORY_USBSERIAL must be set to DEVICE_CATEGORY. Example: "/dev/ttyUSB0"

MOUNTPOINTS

```
public static final String MOUNTPOINTS = "USB.device.mountpoints"
```

OPTIONAL Property key. Value is "USB.device.mountpoint". 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 is not set. The driver can read and write the USB storage through standard API such as File. If a USB base driver set this property, USBDevice.DEVICE_CATEGORY_MASSSTORAGE must be set to DEVICE_CATEGORY. Example: "/mnt/media/usb-storage-01/"

MATCH_MODEL

```
public static final int MATCH_MODEL = 10
```

Constant for the USB device match scale, indicating a match with USB.device.idVendor and USB.device.idProduct. Value is 10.

MATCH_PROTOCOL

```
public static final int MATCH_PROTOCOL = 7
```

Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, USB.device.bDeviceSubClass and USB.device.bDeviceProtocol, or a match with bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol in one of USB.device.interfaceclasses. Value is 7.

MATCH_SUBCLASS

```
public static final int MATCH_SUBCLASS = 5
```

Constant for the USB device match scale, indicating a matchUSB.device.bDeviceClass and USB.device.bDeviceSubClass, or a match with bInterfaceClass and bInterfaceSubClass in one of USB.device.interfaceclasses. Value is 5.

MATCH_CLASS

```
public static final int MATCH_CLASS = 3
```

Constant for the USB device match scale, indicating a match with USB.device.bDeviceClass, or a match with bInterfaceClass in one of USB.device.interfaceclasses. Value is 3.

8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

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