RFC 213 - Serial Device Service

Draft

38 Pages

Set the release level by selecting one from: Draft, Final Draft, Release.

Abstract

This document defines the Java API to communicate with Serial devices on the OSGi platform. Moreover this RFC defines additional specifications for USB-Serial dongles.

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	August 22, 2014	Initial version Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.2	August 26, 2014	Revised version Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.3	August 26, 2014	Added the RFC number Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.4	Sept. 10, 2014	- Modified based on ML comments - Edited some parts 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 residential gateways or devices found in the home network by using various protocols such as USB, Zigbee, ZWave, KNX, and UPnP etc. OSGi Device Access Specification clearly declares that Device Category must be defined outside of OSGi Device Access Specification.

Recently, OSGi is gaining popularity as an enabling technology for building embedded system in residential market. It is expected that USB devices attached to residential gateways on OSGi has been processed since USB interfaces have been introduced into such gateways.

2 Application Domain

Currently there are several standardization bodies such as OSGi Alliance, HGI, and BBF which deal with the deployment of services in an infrastructure based on the usage of residential gateways running OSGi as Execution Platform.

In order to realize the services which access not only IP devices but also non-IP devices connected to the residential gateway, various protocols for home networks, such as ZigBee, Z-Wave, KNX/EHS, and ECHONET-LITE etc, have to be properly taken care of. While some residential gateways originally support those protocols, others do not. Such issue can be solved when such gateways can support USB interfaces and there exist USB dongles which support those protocols. As shown in Fig. 1, the residential gateway with USB dongles can handle various protocols by the way of “add-on”. The point is that such USB dongles can be usually 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 is attached/connected to the residential gateway

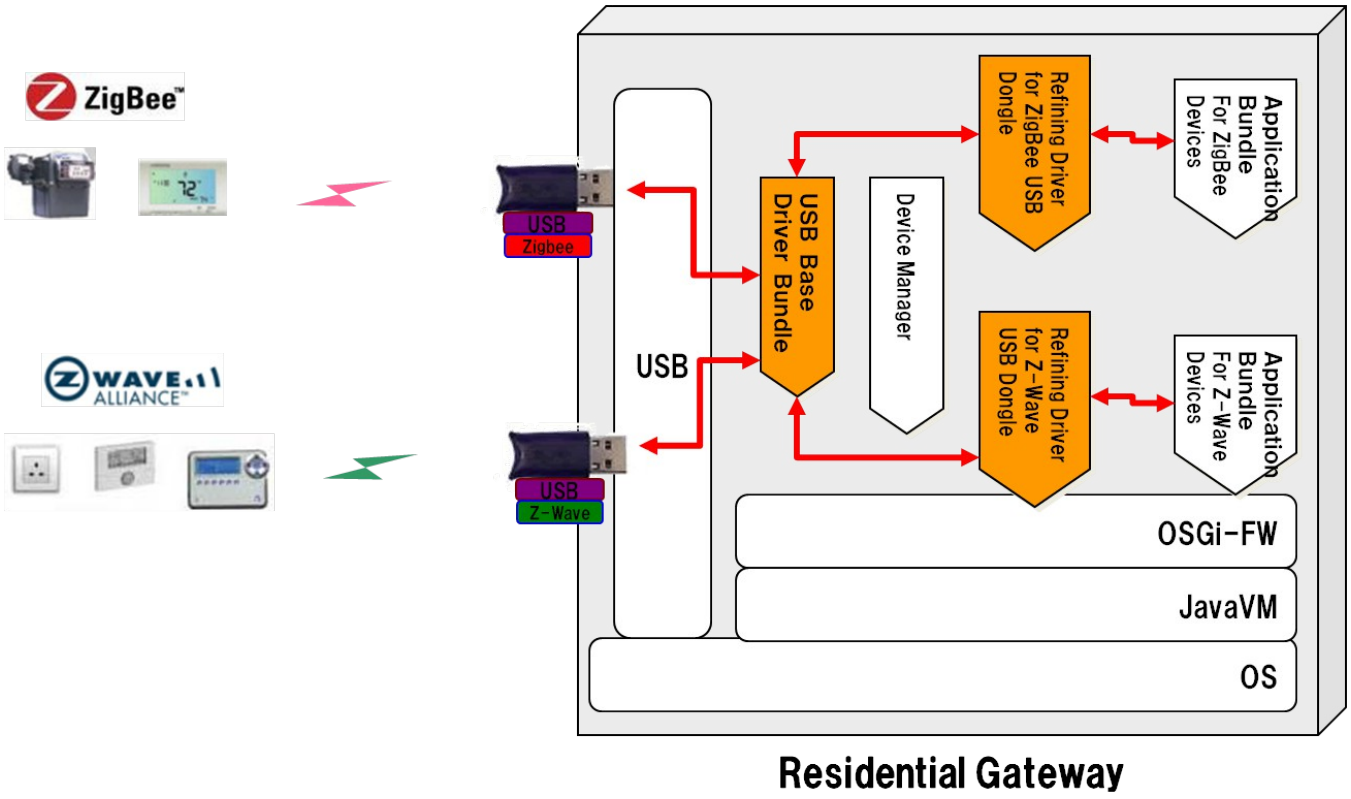

Fig 1 USB Dongles and Residential gateway

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

5.1 Introduction

RFP 149 “USB Device Category” describes the requirements regarding what to be defined as an OSGi Specification when handling USB devices with OSGi. Among various use cases described in this RFP, we would like to focus on such a typical use case as USB-Serial dongle that can be controlled through Serial Communication.

Such communication can be implemented by means of serial connection when using non-IP devices based on ZigBee and Z-wave protocols. The most typical case arises when a USB dongle that supports such protocols is connected to the USB port in the devices such as residential gateways. OS on the gateways will recognize the dongle as a virtual serial device, and initiate a serial communication with the application process.

In order to realize such a case on OSGi platform, this RFC defines a device category and a service for Serial devices. Moreover this RFC defines additional specifications for USB-Serial dongles. This document explains specifications required for establishing communication between OSGi bundle and serial devices.

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

5.2 Entities

- SerialDevice: This is an OSGi service that is used to represent a serial device. This OSGi service stores information regarding serial device and its status as a service property and provides communication function with the device as a SerialConnection. Refining driver bundles can obtain a SerialConnection instance from the SerialDevice service.
- SerialConnection: This is an interface to represent communication with a serial device. Only the refining driver bundles that acquire and maintain this instance can communicate with the serial device.
- Serial base driver bundle: The bundle that implements SerialDevice and SerialConnection. Serial base driver bundle registers SerialDevice services with the Framework. It provides communication function with the (physical) serial devices.
- Refining driver bundle: Refining drivers provide a refined view of a physical device that is already represented by another Device service registered with the Framework (see the details for Device Access Specification).

Figure 2 shows a class diagram of Serial Device Service.

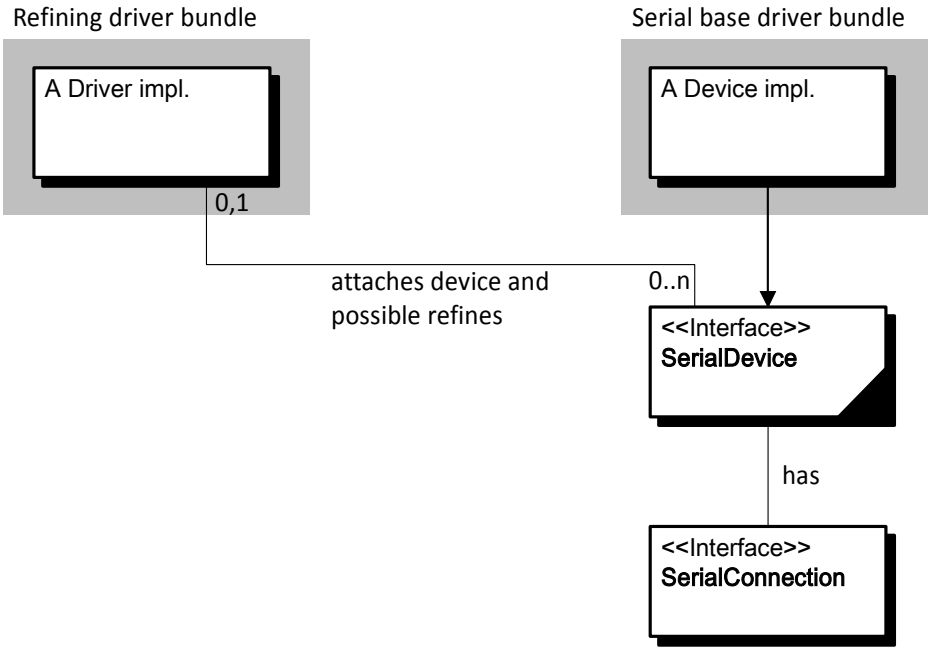

Fig 2: Serial Device Service class diagram

5.3 Assumptions

When a serial device is connected to the gateway, it is mapped to a COM port automatically by native libraries in OS. Those libraries are installed.

5.4 Operation Summary

5.4.1 Serial base driver bundle

A Serial base driver is tracking OS events. Native device driver such as kernel modules in Linux can detect a serial device, communicate with it and allocate it to the corresponding device file (COM port).

When a serial device is connected, native device drivers allocate the device to /dev/ttyS0. Subsequently the serial base driver catches event and gets information about the device. Then the Serial base driver registers a SerialDevice service with service properties.

When the serial device is disconnected, the Serial base driver catches the event and unregisters the SerialDevice service.

5.4.2 Refining driver bundle

The refining driver bundle determines which SerialDevice service is suitable to establish a communication based on service properties. This process is carried out by the device manager based on device access specifications.

The refining driver bundle will get the SerialDevice service then call the SerialDevice#open() method and acquire the SerialConnection.

The bundle executes the necessary settings to the SerialConnection. After this execution, it will acquire the communication stream using SerialConnection#getInputStream(), and/or the SerialConnection#getOutputStream() and initiate a communication with the serial device.

The refining driver bundle invokes SerialConnection#close() when the communication is over.

5.5 SerialDevice Service

SerialDevice is the interface expressing a serial device. It maintains information and state of the serial device as a service property. It provides the communication facility with the serial device as SerialConnection. Each SerialDevice expresses each serial device.

SerialDevice service is registered with the service repository with service properties as shown in the following table.

Table 1: Service properties of SerialDevice Service

The key of service property	M/O	Description
DEVICE_CATEGORY	M	Constant for the value of the service property DEVICE_CATEGORY used for all Serial devices. Value is "Serial".
serial.comport	M	Represents the name of the port. The value is String. Example1: "/dev/ttyUSB0" Example2: "COM5" Example3: "/dev/tty.usbserial-XXXXXX"
~~current.owner~~	~~M~~	~~Represents the owner of the port. The value is String.~~
bus.type	O	Optional. Represents underlying technology such as USB-Serial. The value is String.

When the refining driver bundle calls SerialDevice#open() method, the SerialDevice Service will return the (new) SerialConnection ~~and change the status of the current.owner~~. A SerialDevice instance returns PortInUseException when some bundle calls SerialDevice#open() method and that method was already called.

5.6 SerialConnection

This is an interface to represent communication with a serial device. Only the refining driver bundles that acquire this instance can communicate with the serial device.

If a refining driver invokes SerialConnection#close(), any refining drivers cannot use the SerialConnection instance (cannot open streams).

5.7 SerialDevice service properties for USB-Serial devices

This clause explains SerialDevice service properties and its usage when an USB-Serial device is used as the serial device.

Table 2: Additional service properties for USB-Serial devices

The key of service property	M/O	Description
bus.type	M	Must be set "USB".
usb.bus	M	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.address	M	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

Universal Serial Bus Specification (USB Specification) defines USB Interface(s). OS maps each USB interface to the corresponding virtual serial device. A Serial base driver bundle must register SerialDevice service to the corresponding virtual serial device. A SerialDevice service has such information as contains USB device information and USB interface information.

5.7.1 Service properties from USB Specification

The USB Specification defines a device descriptor. USB devices report their attributes using descriptors. SerialDevice service has some properties from the USB device descriptors. Table 3 shows the mapping between the device descriptors and service properties of SerialDevice.

Table 3: Device Descriptor and Service Property

Device Descriptor's Field from USB Spec.	**Service Property of SerialDevice**	**M/O**	**Java type**
bLength	`none`	-	-
bDescriptorType	`none`	-	-

bcdUSB	`usb.bcdUSB`	O	`String`
bDeviceClass	`usb.bDeviceClass`	M	`String`
bDeviceSubClass	`usb.bDeviceSubClass`	M	`String`
bDeviceProtocol	`usb.bDeviceProtocol`	M	`String`
bMaxPacketSize0	`usb.bMaxPacketSize0`	O	`Integer`
idVendor	`usb.idVendor`	M	`String`
idProduct	`usb.idProduct`	M	`String`
bcdDevice	`usb.bcdDevice`	M	`String`
iManufacturer	`usb.Manufacturer`	O	`String`
iProduct	`DEVICE_DESCRIPTION`	O	`String`
iSerialNumber	`DEVICE_SERIAL`	O	`String`
bNumConfigurations	`usb.bNumConfigurations`	O	`Integer`

- `usb.bcdUSB` - OPTIONAL property key. The value is `String`, the 4-digit BCD format.
  - Example: ”0210”
- `usb.bDeviceClass` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: “ff”
- `usb.bDeviceSubClass` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: ”ff”
- `usb.bDeviceProtocol` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: ”ff”
- `usb.bMaxPacketSize0` – OPTIONAL property key. The value is Integer.
- `usb.idVendor` - MANDATORY property key. The value is `String`, hexadecimal, 4-digits.
  - Example: “0403”
- `usb.idProduct` - MANDATORY property key. The value is `String`, hexadecimal, 4-digits.

  - Example: “8372”

- `usb.bcdDevice` - MANDATORY property key. The value is `String`, the 4-digit BCD format.
  - Example: ”0200”
- `usb.Manufacturer` - OPTIONAL property key. The value is `String` of indicated in iManufacturer. (The value is not the index.)
  - Example: "Buffalo Inc."
- `DEVICE_DESCRIPTION` - OPTIONAL property key. The value is `String` of indicated in iProduct. (The value is not the index.)
  - Example: "USB2.0 PC Camera"
- `DEVICE_SERIAL` - OPTIONAL property key. The value is `String` of indicated in iSerialNumber. (The value is not the index.)
  - Example: "57B0002600000001"
- `usb.bNumConfigurations` – OPTIONAL property key. The value is Integer.

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

So these fields add to the service properties (see Table 4).

Table 4: Interface Descriptor and Service Property

Interface Descriptor's Field from USB Spec.	**Service Property of SerialDevice**	**M/O**	**Java type**
bLength	`none`	-	-
bDescriptorType	`none`	-	-
bInterfaceNumber	`usb.bInterfaceNumber`	M	`Integer`
bAlternateSetting	`usb.bAlternateSetting`	O	`Integer`
bNumEndpoints	`usb.bNumEndpoints`	O	`Integer`
bInterfaceClass	`usb.bInterfaceClass`	M	`String`
bInterfaceSubClass	`usb.bInterfaceSubClass`	M	`String`
bInterfaceProtocol	`usb.bInterfaceProtocol`	M	`String`
iInterface	`usb.Interface`	O	`String`

- `usb.bInterfaceNumber` – MANDATORY property key. The value is Integer.
- `usb.bAlternateSetting` – OPTIONAL property key. The value is Integer.
- `usb.bNumEndpoints` – OPTIONAL property key. The value is Integer.
- `usb.bInterfaceClass` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: “ff”
- `usb.bInterfaceSubClass` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: ”ff”
- `usb.bInterfaceProtocol` - MANDATORY property key. The value is `String`, hexadecimal, 2-digits.
  - Example: ”ff”
- `usb.Interface` - OPTIONAL property key. The value is `String` of indicated in iInterface. (The value is not the index.)

5.7.2 Match scale

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

- MATCH_VERSION – Constant for the USB-Serial device match scale, indicating a match with usb.idVendor, usb.idProduct and usb.bcdDevice. Value is 20.
- MATCH_MODEL - Constant for the USB-Serial device match scale, indicating a match with usb.idVendor and usb.idProduct. Value is 10.

6 Data Transfer Objects

This RFC does not provide Data Transfer Objects.

7 Javadoc

OSGi Javadoc

9/10/14 11:29 AM

Package Summary		Page
org.osgi.service.serial	Serial Device Service Specification Package Version 1.0.	17
org.osgi.service.serial.usb	SerialDevice service properties for USB-Serial devices Specification Package Version 1.0.	30

Package org.osgi.service.serial

Serial Device Service Specification Package Version 1.0.

See:
Description

Interface Summary		Page
SerialConnection	SerialConnection is an open communications port.	*19*
SerialDevice	SerialDevice is an interface to express a device performing serial communication.	*27*

Exception Summary		Page
PortInUseException	Thrown when the specified port is in use.	*18*
UnsupportedCommOperationException	Thrown when a driver doesn't allow the specified operation.	*29*

Package org.osgi.service.serial Description

Serial Device Service Specification Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.serial; version="[1.0,2.0)"
```

Class PortInUseException

org.osgi.service.serial

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ org.osgi.service.serial.PortInUseException
```

All Implemented Interfaces:

Serializable

```
public class PortInUseException
extends Exception
```

Thrown when the specified port is in use.

Constructor Summary	Page
PortInUseException(String currentOwner) Constructor.	18

Method Summary		Page
String	**currentOwner**() Describes the current owner of the communications port.	18

Constructor Detail

PortInUseException

```
public PortInUseException(String currentOwner)
```

Constructor.

Method Detail

currentOwner

```
public String currentOwner()
```

Describes the current owner of the communications port.

Returns:

current owner

Interface SerialConnection

org.osgi.service.serial

---

public interface **SerialConnection**

SerialConnection is an open communications port.

Field Summary		Page
int	**DATABITS_5** 5 data bit format.	20
int	**DATABITS_6** 6 data bit format.	20
int	**DATABITS_7** 7 data bit format.	21
int	**DATABITS_8** 8 data bit format.	21
int	**FLOWCONTROL_NONE** Flow control off.	21
int	**FLOWCONTROL_RTSCTS_IN** RTS/CTS flow control on input.	21
int	**FLOWCONTROL_RTSCTS_OUT** RTS/CTS flow control on output.	21
int	**FLOWCONTROL_XONXOFF_IN** XON/XOFF flow control on input.	21
int	**FLOWCONTROL_XONXOFF_OUT** XON/XOFF flow control on output.	21
int	**PARITY_EVEN** EVEN parity scheme.	22
int	**PARITY_MARK** MARK parity scheme.	22
int	**PARITY_NONE** No parity bit.	22
int	**PARITY_ODD** ODD parity scheme.	22
int	**PARITY_SPACE** SPACE parity scheme.	22
int	**STOPBITS_1** Number of STOP bits - 1.	22
int	**STOPBITS_1_5** Number of STOP bits - 1-1/2.	22
int	**STOPBITS_2** Number of STOP bits - 2.	22

Method Summary		Page
void	**close**() Closes the communications port.	23

int	**getBaudRate**() Gets the currently configured baud rate.	23
int	**getDataBits**() Gets the currently configured number of data bits.	24
int	**getFlowControlMode**() Gets the currently configured flow control mode.	24
InputStream	**getInputStream**() Returns an input stream. This is the only way to receive data from the communications port. If the port is unidirectional and doesn't support receiving data, then getInputStream returns null.	23
OutputStream	**getOutputStream**() Returns an output stream. This is the only way to send data to the communications port.	23
int	**getParity**() Get the currently configured parity setting.	24
int	**getStopBits**() Gets the currently defined stop bits.	24
boolean	**isDTR**() Gets the state of the DTR (Data Terminal Ready) bit in the UART, if supported by the underlying implementation.	24
boolean	**isRTS**() Gets the state of the RTS (Request To Send) bit in the UART, if supported by the underlying implementation.	25
void	**setDTR**(boolean dtr) Sets or clears the DTR (Data Terminal Ready) bit in the UART, if supported by the underlying implementation.	25
void	**setFlowControlMode**(int flowcontrol) Sets the flow control mode.	25
void	**setRTS**(boolean rts) Sets or clears the RTS (Request To Send) bit in the UART, if supported by the underlying implementation.	25
void	**setSerialPortParams**(int baudrate, int dataBits, int stopBits, int parity) Sets serial port parameters.	26

Field Detail

DATABITS_5

```
public static final int DATABITS_5 = 5
```

5 data bit format.

DATABITS_6

```
public static final int DATABITS_6 = 6
```

6 data bit format.

DATABITS_7

```
public static final int DATABITS_7 = 7
```

7 data bit format.

DATABITS_8

```
public static final int DATABITS_8 = 8
```

8 data bit format.

FLOWCONTROL_NONE

```
public static final int FLOWCONTROL_NONE = 0
```

Flow control off.

FLOWCONTROL_RTSCTS_IN

```
public static final int FLOWCONTROL_RTSCTS_IN = 1
```

RTS/CTS flow control on input.

FLOWCONTROL_RTSCTS_OUT

```
public static final int FLOWCONTROL_RTSCTS_OUT = 2
```

RTS/CTS flow control on output.

FLOWCONTROL_XONXOFF_IN

```
public static final int FLOWCONTROL_XONXOFF_IN = 4
```

XON/XOFF flow control on input.

FLOWCONTROL_XONXOFF_OUT

```
public static final int FLOWCONTROL_XONXOFF_OUT = 8
```

XON/XOFF flow control on output.

PARITY_NONE

```
public static final int PARITY_NONE = 0
```

No parity bit.

PARITY_ODD

```
public static final int PARITY_ODD = 1
```

ODD parity scheme. The parity bit is added so there are an odd number of TRUE bits.

PARITY_EVEN

```
public static final int PARITY_EVEN = 2
```

EVEN parity scheme. The parity bit is added so there are an even number of TRUE bits.

PARITY_MARK

```
public static final int PARITY_MARK = 3
```

MARK parity scheme.

PARITY_SPACE

```
public static final int PARITY_SPACE = 4
```

SPACE parity scheme.

STOPBITS_1

```
public static final int STOPBITS_1 = 1
```

Number of STOP bits - 1.

STOPBITS_2

```
public static final int STOPBITS_2 = 2
```

Number of STOP bits - 2.

STOPBITS_1_5

```
public static final int STOPBITS_1_5 = 3
```

Number of STOP bits - 1-1/2. Some UARTs permit 1-1/2 STOP bits only with 5 data bit format, but permit 1 or 2 STOP bits with any format.

Method Detail

close

```
void close()
```

Closes the communications port.

---

getInputStream

```
InputStream getInputStream()
                    throws IOException
```

Returns an input stream. This is the only way to receive data from the communications port. If the port is unidirectional and doesn't support receiving data, then getInputStream returns null.

Returns:
InputStream object that can be used to read from the port
Throws:
`IOException` - if an I/O error occurred

---

getOutputStream

```
OutputStream getOutputStream()
                     throws IOException
```

Returns an output stream. This is the only way to send data to the communications port. If the port is unidirectional and doesn't support sending data, then getOutputStream returns null.

Returns:
OutputStream object that can be used to write to the port
Throws:
`IOException` - if an I/O error occurred

---

getBaudRate

```
int getBaudRate()
```

Gets the currently configured baud rate.

Returns:
integer value indicating the baud rate

---

getDataBits

```
int getDataBits()
```

Gets the currently configured number of data bits.

Returns:
integer that can be equal to DATABITS_5, DATABITS_6, DATABITS_7, or DATABITS_8

---

getFlowControlMode

```
int getFlowControlMode()
```

Gets the currently configured flow control mode.

Returns:
an integer bitmask of the modes FLOWCONTROL_NONE, FLOWCONTROL_RTSCTS_IN, FLOWCONTROL_RTSCTS_OUT, FLOWCONTROL_XONXOFF_IN, and FLOWCONTROL_XONXOFF_OUT.

---

getParity

```
int getParity()
```

Get the currently configured parity setting.

Returns:
integer that can be equal to PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK or PARITY_SPACE.

---

getStopBits

```
int getStopBits()
```

Gets the currently defined stop bits.

Returns:
integer that can be equal to STOPBITS_1, STOPBITS_2, or STOPBITS_1_5

---

isDTR

```
boolean isDTR()
```

Gets the state of the DTR (Data Terminal Ready) bit in the UART, if supported by the underlying implementation.

Returns:
state of the DTR

isRTS

`boolean isRTS()`

Gets the state of the RTS (Request To Send) bit in the UART, if supported by the underlying implementation.

Returns:
state of the RTS

setDTR

`void setDTR(boolean dtr)`

Sets or clears the DTR (Data Terminal Ready) bit in the UART, if supported by the underlying implementation.

Parameters:
`dtr` -

- `true` set DTR
- `false` clear DTR

setFlowControlMode

`void setFlowControlMode(int flowcontrol)`
`throws UnsupportedCommOperationException`

Sets the flow control mode.

Parameters:
`flowcontrol` - Can be a bitmask combination of

- FLOWCONTROL_NONE: no flow control
- FLOWCONTROL_RTSCTS_IN: RTS/CTS (hardware) flow control for input
- FLOWCONTROL_RTSCTS_OUT: RTS/CTS (hardware) flow control for output
- FLOWCONTROL_XONXOFF_IN: XON/XOFF (software) flow control for input
- FLOWCONTROL_XONXOFF_OUT: XON/XOFF (software) flow control for output

Throws:
`UnsupportedCommOperationException` - if any of the flow control mode was not supported by the underline OS, or if input and output flow control are set to different values, i.e. one hardware and one software. The flow control mode will revert to the value before the call was made.

setRTS

`void setRTS(boolean rts)`

Sets or clears the RTS (Request To Send) bit in the UART, if supported by the underlying implementation.

Parameters:

`rts` -

- `true` set RTS
- `false` clear RTS

---

setSerialPortParams

```
void setSerialPortParams(int baudrate,
                         int dataBits,
                         int stopBits,
                         int parity)
                  throws UnsupportedCommOperationException
```

Sets serial port parameters. DEFAULT: 9600 baud, 8 data bits, 1 stop bit, no parity

Parameters:

`baudrate` - If the baudrate passed in by the application is unsupported by the driver, the driver will throw an UnsupportedCommOperationException

`dataBits` -

- DATABITS_5: 5 bits
- DATABITS_6: 6 bits
- DATABITS_7: 7 bits
- DATABITS_8: 8 bits

`stopBits` -

- STOPBITS_1: 1 stop bit
- STOPBITS_2: 2 stop bits
- STOPBITS_1_5: 1.5 stop bits

`parity` -

- PARITY_NONE: no parity
- PARITY_ODD: odd parity
- PARITY_EVEN: even parity
- PARITY_MARK: mark parity
- PARITY_SPACE: space parity

Throws:

`UnsupportedCommOperationException` - if any of the above parameters are specified incorrectly. All four of the parameters will revert to the values before the call was made.

Interface SerialDevice

org.osgi.service.serial

All Known Subinterfaces:
USBSerialDevice

---

public interface **SerialDevice**

SerialDevice is an interface to express a device performing serial communication.

Field Summary		Page
String	**BUS_TYPE** Optional. The key string of "bus.type" service property. Represents underlying technology such as USB-Serial. The value is String.	28
String	**DEVICE_CATEGORY** Constant for the value of the service property DEVICE_CATEGORY used for all Serial devices.	27
String	**SERIAL_COMPORT** The key string of "serial.comport" service property. Represents the name of the port. The value is String. Example1: "/dev/ttyUSB0" Example2: "COM5" Example3: "/dev/tty.usbserial-XXXXXX"	27

Method Summary		Page
SerialConnection	**open**() Opens the communications port. Open obtains exclusive ownership of the port.	28

Field Detail

DEVICE_CATEGORY

public static final String **DEVICE_CATEGORY** = "Serial"

Constant for the value of the service property DEVICE_CATEGORY used for all Serial devices. Value is "Serial".

SERIAL_COMPORT

public static final String **SERIAL_COMPORT** = "serial.comport"

The key string of "serial.comport" service property. Represents the name of the port.

The value is String.
Example1: "/dev/ttyUSB0"
Example2: "COM5"
Example3: "/dev/tty.usbserial-XXXXXX"

BUS_TYPE

```
public static final String BUS_TYPE = "bus.type"
```

Optional.
The key string of "bus.type" service property.
Represents underlying technology such as USB-Serial.
The value is String.

Method Detail

open

```
SerialConnection open()
                 throws PortInUseException
```

Opens the communications port.
Open obtains exclusive ownership of the port.

Returns:
SerialConnection
Throws:
`PortInUseException` - if the port is in use by some other application that is not willing to relinquish ownership

Class UnsupportedCommOperationException

org.osgi.service.serial

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ org.osgi.service.serial.UnsupportedCommOperationException
```

All Implemented Interfaces:

Serializable

```
public class UnsupportedCommOperationException
extends Exception
```

Thrown when a driver doesn't allow the specified operation.

Constructor Summary	Page
UnsupportedCommOperationException() Constructs an UnsupportedCommOperationException with no detail message.	29
UnsupportedCommOperationException(String message) Constructs an UnsupportedCommOperationException with the specified detail message.	29

Constructor Detail

UnsupportedCommOperationException

```
public UnsupportedCommOperationException()
```

Constructs an UnsupportedCommOperationException with no detail message.

UnsupportedCommOperationException

```
public UnsupportedCommOperationException(String message)
```

Constructs an UnsupportedCommOperationException with the specified detail message.

Parameters:

`message` - the detail message

Package org.osgi.service.serial.usb

SerialDevice service properties for USB-Serial devices Specification Package Version 1.0.

See:
Description

Interface Summary		Page
USBSerialDevice	Defines additional SerialDevice service properties for USB-Serial devices.	*31*

Package org.osgi.service.serial.usb Description

SerialDevice service properties for USB-Serial devices Specification Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.serial; version="[1.0,2.0)"
```

Interface USBSerialDevice

org.osgi.service.serial.usb

All Superinterfaces:
SerialDevice

```
public interface USBSerialDevice
extends SerialDevice
```

Defines additional SerialDevice service properties for USB-Serial devices.

Field Summary		Page
String	**BUS_TYPE_USB** The value string of "bus.type" service property.	32
String	**USB_ADDRESS** The key string of "usb.address" service property. Used to identify USB devices with same VID / PID.	33
String	**USB_BALTERNATESETTING** Optional. The key string of "usb.bAlternateSetting" service property. Service properties from USB Interface Descriptor.	35
String	**USB_BCDDEVICE** The key string of "usb.bcdDevice" service property. Service properties from USB Device Descriptor.	34
String	**USB_BCDUSB** Optional. The key string of "usb.bcdUSB" service property. Service properties from USB Device Descriptor.	33
String	**USB_BDEVICECLASS** The key string of "usb.bDeviceClass" service property. Service properties from USB Device Descriptor.	33
String	**USB_BDEVICEPROTOCOL** The key string of "usb.bDeviceProtocol" service property. Service properties from USB Device Descriptor.	33
String	**USB_BDEVICESUBCLASS** The key string of "usb.bDeviceSubClass" service property. Service properties from USB Device Descriptor.	33
String	**USB_BINTERFACECLASS** The key string of "usb.bInterfaceClass" service property. Service properties from USB Interface Descriptor.	36
String	**USB_BINTERFACENUMBER** The key string of "usb.bInterfaceNumber" service property. Service properties from USB Interface Descriptor.	35
String	**USB_BINTERFACEPROTOCOL** The key string of "usb.bInterfaceProtocol" service property. Service properties from USB Interface Descriptor.	36
String	**USB_BINTERFACESUBCLASS** The key string of "usb.bInterfaceSubClass" service property. Service properties from USB Interface Descriptor.	36

String	**USB_BMAXPACKETSIZE0** Optional. The key string of "usb.bMaxPacketSize0" service property. Service properties from USB Device Descriptor.	34
String	**USB_BNUMCONFIGURATIONS** Optional. The key string of "usb.bNumConfigurations" service property. Service properties from USB Device Descriptor.	35
String	**USB_BNUMENDPOINTS** Optional. The key string of "usb.bNumEndpoints" service property. Service properties from USB Interface Descriptor.	35
String	**USB_BUS** The key string of "usb.bus" service property. Used to identify USB devices with same VID / PID.	32
String	**USB_IDPRODUCT** The key string of "usb.idProduct" service property. Service properties from USB Device Descriptor.	34
String	**USB_IDVENDOR** The key string of "usb.idVendor" service property. Service properties from USB Device Descriptor.	34
String	**USB_INTERFACE** Optional. The key string of "usb.Interface" service property. Service properties from USB Interface Descriptor.	36
String	**USB_MANUFACTURER** Optional. The key string of "usb.Manufacturer" service property. Service properties from USB Device Descriptor.	34

Fields inherited from interface org.osgi.service.serial.SerialDevice
BUS_TYPE, DEVICE_CATEGORY, SERIAL_COMPORT

Methods inherited from interface org.osgi.service.serial.SerialDevice
open

Field Detail

BUS_TYPE_USB

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

The value string of "bus.type" service property.

USB_BUS

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

The key string of "usb.bus" service property. 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.

The value is Integer.
Example: 3

USB_ADDRESS

```
public static final String USB_ADDRESS = "usb.address"
```

The key string of "usb.address" service property. 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.
The value is Integer.
Example: 2

USB_BCDUSB

```
public static final String USB_BCDUSB = "usb.bcdUSB"
```

Optional.
The key string of "usb.bcdUSB" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bcdUSB".
The value is String, the 4-digit BCD format. Example: "0210"

USB_BDEVICECLASS

```
public static final String USB_BDEVICECLASS = "usb.bDeviceClass"
```

The key string of "usb.bDeviceClass" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bDeviceClass".
The value is String, hexadecimal, 2-digits.
Example: "ff"

USB_BDEVICESUBCLASS

```
public static final String USB_BDEVICESUBCLASS = "usb.bDeviceSubClass"
```

The key string of "usb.bDeviceSubClass" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bDeviceSubClass".
The value is String, hexadecimal, 2-digits.
Example: "ff"

USB_BDEVICEPROTOCOL

```
public static final String USB_BDEVICEPROTOCOL = "usb.bDeviceProtocol"
```

The key string of "usb.bDeviceProtocol" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bDeviceProtocol".
The value is String, hexadecimal, 2-digits. Example: "ff"

USB_BMAXPACKETSIZE0

```
public static final String USB_BMAXPACKETSIZE0 = "usb.bMaxPacketSize0"
```

Optional.
The key string of "usb.bMaxPacketSize0" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bMaxPacketSize0".
The value is Integer.

USB_IDVENDOR

```
public static final String USB_IDVENDOR = "usb.idVendor"
```

The key string of "usb.idVendor" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "idVendor".
The value is String, hexadecimal, 4-digits. Example: "0403"

USB_IDPRODUCT

```
public static final String USB_IDPRODUCT = "usb.idProduct"
```

The key string of "usb.idProduct" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "idProduct".
The value is String, hexadecimal, 4-digits. Example: "8372"

USB_BCDDEVICE

```
public static final String USB_BCDDEVICE = "usb.bcdDevice"
```

The key string of "usb.bcdDevice" service property. Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bcdDevice".
The value is String, the 4-digit BCD format. Example: "0200"

USB_MANUFACTURER

```
public static final String USB_MANUFACTURER = "usb.Manufacturer"
```

Optional.
The key string of "usb.Manufacturer" service property.
Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "iManufacturer".
The value is String of indicated in iManufacturer. (The value is not the index.)
Example: "Buffalo Inc."

USB_BNUMCONFIGURATIONS

```
public static final String USB_BNUMCONFIGURATIONS = "usb.bNumConfigurations"
```

Optional.
The key string of "usb.bNumConfigurations" service property.
Service properties from USB Device Descriptor. Device Descriptor's Field from USB Spec is "bNumConfigurations".
The value is Integer.

USB_BINTERFACENUMBER

```
public static final String USB_BINTERFACENUMBER = "usb.bInterfaceNumber"
```

The key string of "usb.bInterfaceNumber" service property.
Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bInterfaceNumber".
The value is Integer.

USB_BALTERNATESETTING

```
public static final String USB_BALTERNATESETTING = "usb.bAlternateSetting"
```

Optional.
The key string of "usb.bAlternateSetting" service property.
Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bAlternateSetting".
The value is Integer.

USB_BNUMENDPOINTS

```
public static final String USB_BNUMENDPOINTS = "usb.bNumEndpoints"
```

Optional.
The key string of "usb.bNumEndpoints" service property.
Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bNumEndpoints".
The value is Integer.

USB_BINTERFACECLASS

```
public static final String USB_BINTERFACECLASS = "usb.bInterfaceClass"
```

The key string of "usb.bInterfaceClass" service property. Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bInterfaceClass".
The value is String, hexadecimal, 2-digits. Example: "ff"

USB_BINTERFACESUBCLASS

```
public static final String USB_BINTERFACESUBCLASS = "usb.bInterfaceSubClass"
```

The key string of "usb.bInterfaceSubClass" service property. Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bInterfaceSubClass".
The value is String, hexadecimal, 2-digits. Example: "ff"

USB_BINTERFACEPROTOCOL

```
public static final String USB_BINTERFACEPROTOCOL = "usb.bInterfaceProtocol"
```

The key string of "usb.bInterfaceProtocol" service property. Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "bInterfaceProtocol".
The value is String, hexadecimal, 2-digits. Example: "ff"

USB_INTERFACE

```
public static final String USB_INTERFACE = "usb.Interface"
```

Optional.
The key string of "usb.Interface" service property. Service properties from USB Interface Descriptor. Interface Descriptor's Field from USB Spec is "iInterface".
The value is String of indicated in iInterface. (The value is not the index.)

8 Considered Alternatives

8.1 USB Category

RFC 202 tried to give a technical solution for RFP 149 at the beginning.

The current draft of RFC 202 did not describe the necessary communication functions and included some unclear definitions regarding protocols (USB, Serial devices, etc). During the discussion at REG WG we decided to take another approach instead of updating RFC 202.

9 Security Considerations

ServicePermission is needed when a bundle get SerialDevice 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, 239-0847, 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