RFC 213 - Serial Device Service

Final

34 Pages

Abstract

This document defines the Java API to communicate with Serial devices on the OSGi platform.

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
v0.5	Nov. 6, 2014	- Removed USB properties - Changed design based on ML discussion Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
v0.6	Dec. 15, 2014	- Changed design based on review comments (Use a DTO for the configuration. Add an event listener.) - Removed mention the Service Factory - Add note for the USB-Serial use case Yukio Koike, NTT Corporation, koike.yukio@lab.ntt.co.jp
final	Dec. 25, 2014	- Finalized to vote 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

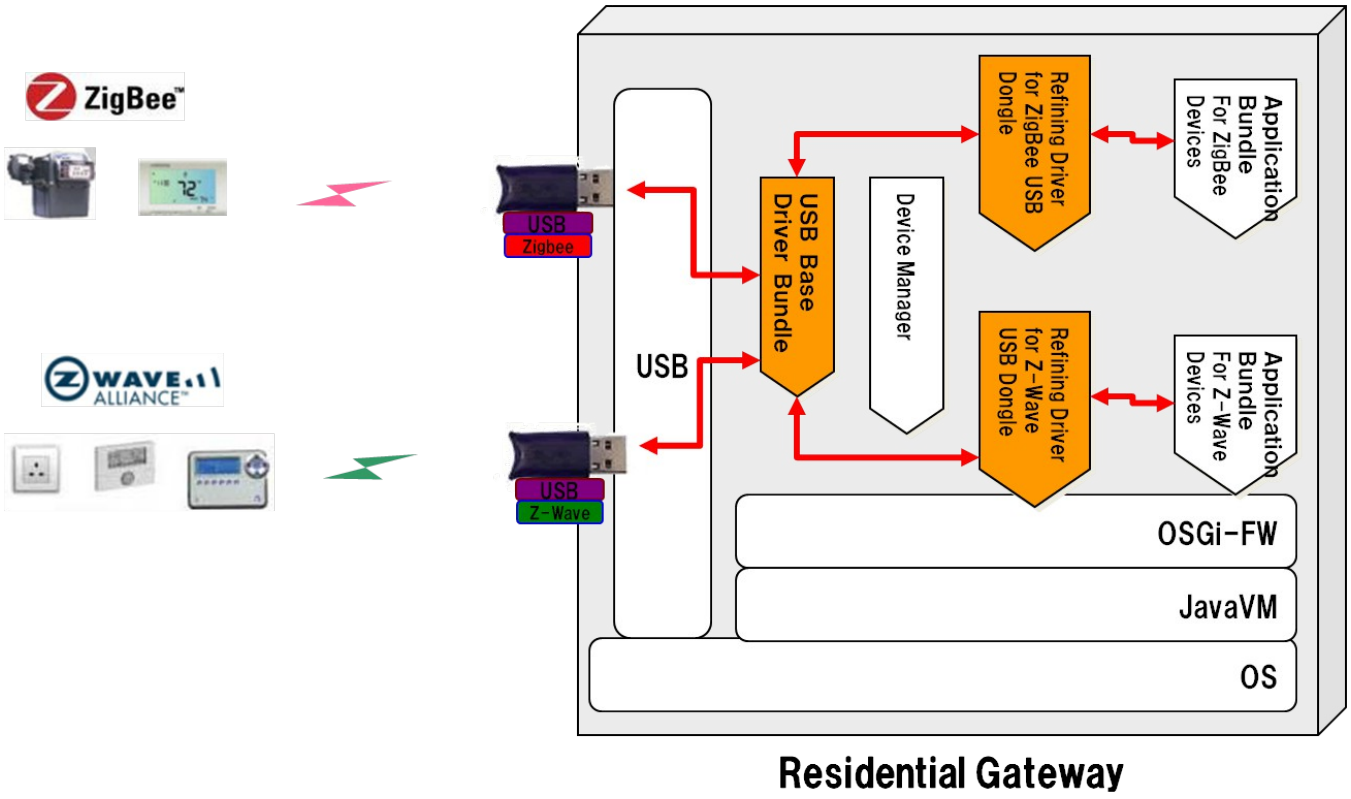

Fig 1 USB Dongles and Residential gateway

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. This document explains specifications required for establishing communication between OSGi bundle and serial devices.

RFC 202 “USB Information Device Category” defines a device category for USB devices. Therefore RFC 202 and this RFC are the solution for RFP 149.

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.
- SerialEventListener: A listener to events coming from Serial devices.
- Serial base driver bundle: The bundle that implements SerialDevice. 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.

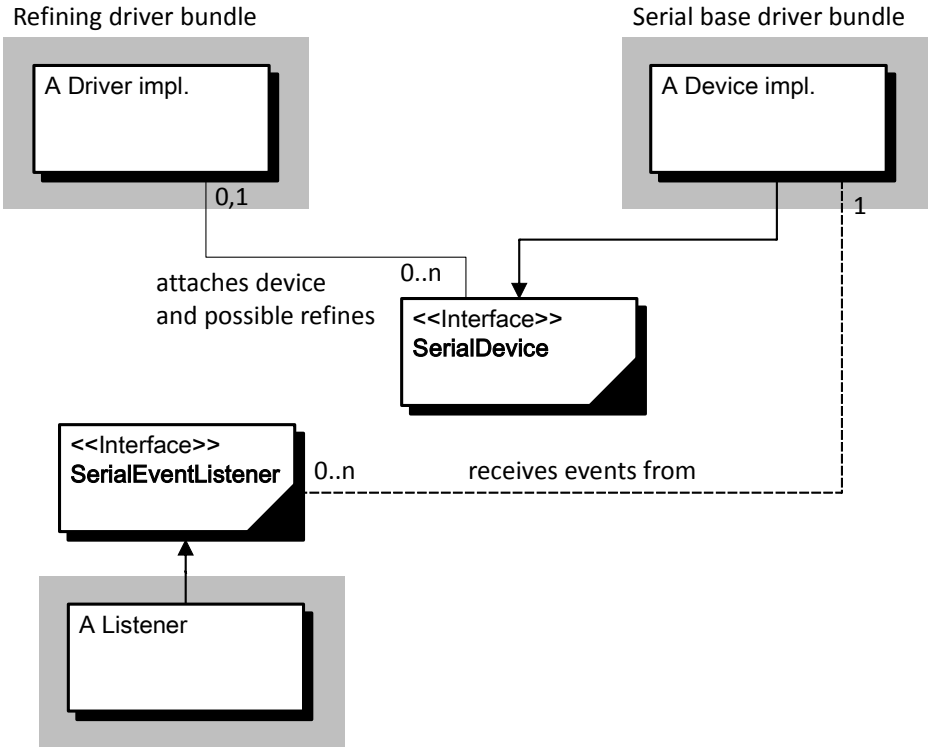

Fig 2: Serial Device Service class diagram

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.

If the device is USB-Serial device, then it is recommended that the base driver implements org.osgi.service.usbinfo.USBInfo and SerialDevice concurrently, and registers the service under USBInfo and SerialDevice interfaces (call registerService(String[] clazzes, service, props)).

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

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. 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"

5.6 SerialEventListener Service

Serial events are sent using the whiteboard model, in which a bundle interested in receiving the Serial events registers an object implementing the SerialEventListener interface. A COM port name can be set to limit the events for which a bundle is notified.

6 Data Transfer Objects

This RFC does not provide Data Transfer Objects.

7 Javadoc

OSGi Javadoc

12/15/14 5:27 PM

Package Summary		Page
org.osgi.service.serial	Serial Device Service Specification Package Version 1.0.	13

Package org.osgi.service.serial

Serial Device Service Specification Package Version 1.0.

See:

[Description](#)

Interface Summary		Page
SerialDevice	SerialDevice is an interface to express a device performing serial communication.	22
SerialEventListener	Serial events are sent using the whiteboard model, in which a bundle interested in receiving the Serial events registers an object implementing the SerialEventListener interface.	28

Class Summary		Page
BaudRate		14
DataBits		16
FlowControl		18
Parity		20
SerialEvent		27
SerialPortDTO	An object represents the Serial port configuration	29
StopBits		31

Exception Summary		Page
SerialDeviceException		26

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 BaudRate

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.BaudRate
```

```
public class BaudRate
extends Object
```

Field Summary		Page
static int	BAUD_115200	15
static int	BAUD_14400	14
static int	BAUD_19200	14
static int	BAUD_38400	14
static int	BAUD_57600	15
static int	BAUD_9600	14
static int	BAUD_AUTO	14

Constructor Summary	Page
BaudRate()	15

Field Detail

BAUD_AUTO

```
public static final int BAUD_AUTO = -1
```

BAUD_9600

```
public static final int BAUD_9600 = 9600
```

BAUD_14400

```
public static final int BAUD_14400 = 14400
```

BAUD_19200

```
public static final int BAUD_19200 = 19200
```

BAUD_38400

```
public static final int BAUD_38400 = 38400
```

BAUD_57600

```
public static final int BAUD_57600 = 57600
```

BAUD_115200

```
public static final int BAUD_115200 = 115200
```

Constructor Detail

BaudRate

```
public BaudRate()
```

Class DataBits

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.DataBits
```

```
public class DataBits
extends Object
```

Field Summary		Page
static int	**EIGHT** Data bits: 8.	16
static int	**FIVE** Data bits: 5.	16
static int	**SEVEN** Data bits: 7.	16
static int	**SIX** Data bits: 6.	16

Constructor Summary	Page
DataBits()	17

Field Detail

FIVE

```
public static final int FIVE = 5
```

Data bits: 5.

SIX

```
public static final int SIX = 6
```

Data bits: 6.

SEVEN

```
public static final int SEVEN = 7
```

Data bits: 7.

EIGHT

```
public static final int EIGHT = 8
```

Data bits: 8.

Constructor Detail

DataBits

```
public DataBits()
```

Class FlowControl

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.FlowControl
```

```
public class FlowControl
extends Object
```

Field Summary		Page
static int	**NONE** Flow control: None.	18
static int	**RTSCTS_IN** Flow control: RTS/CTS on input.	18
static int	**RTSCTS_OUT** Flow control: RTS/CTS on output.	18
static int	**XONXOFF_IN** Flow control: XON/XOFF on input.	19
static int	**XONXOFF_OUT** Flow control: XON/XOFF on output.	19

Constructor Summary	Page
FlowControl()	19

Field Detail

NONE

```
public static final int NONE = 0
```

Flow control: None.

RTSCTS_IN

```
public static final int RTSCTS_IN = 1
```

Flow control: RTS/CTS on input.

RTSCTS_OUT

```
public static final int RTSCTS_OUT = 2
```

Flow control: RTS/CTS on output.

XONXOFF_IN

```
public static final int XONXOFF_IN = 4
```

Flow control: XON/XOFF on input.

XONXOFF_OUT

```
public static final int XONXOFF_OUT = 8
```

Flow control: XON/XOFF on output.

Constructor Detail

FlowControl

```
public FlowControl()
```

Class Parity

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.Parity
```

```
public class Parity
extends Object
```

Field Summary		Page
static int	**EVEN** Parity: Even.	20
static int	**MARK** Parity: Mark.	21
static int	**NONE** Parity: None.	20
static int	**ODD** Parity: Odd.	20
static int	**SPACE** Parity: Space.	21

Constructor Summary	Page
Parity()	21

Field Detail

NONE

```
public static final int NONE = 0
```

Parity: None.

ODD

```
public static final int ODD = 1
```

Parity: Odd.

EVEN

```
public static final int EVEN = 2
```

Parity: Even.

MARK

```
public static final int MARK = 3
```

Parity: Mark.

SPACE

```
public static final int SPACE = 4
```

Parity: Space.

Constructor Detail

Parity

```
public Parity()
```

Interface SerialDevice

org.osgi.service.serial

public interface **SerialDevice**

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

Field Summary		Page
String	**DEVICE_CATEGORY** Constant for the value of the service property DEVICE_CATEGORY used for all Serial devices.	23
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"	23

Method Summary		Page
SerialPortDTO	**getConfiguration**() Gets the Serial port configuration.	23
InputStream	**getInputStream**() Returns an input stream.	23
OutputStream	**getOutputStream**() Returns an output stream.	23
boolean	**isCTS**() Gets the CTS state.	25
boolean	**isDSR**() Gets the DSR state.	24
boolean	**isDTR**() Gets the DTR state.	24
boolean	**isRTS**() Gets the DTS state.	24
void	**setConfiguration**(SerialPortDTO dto) Sets the Serial port configuration.	24
void	**setDTR**(boolean dtr) Sets the DTR state.	25
void	**setRTS**(boolean rts) Sets the RTS state.	25

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"

Method Detail

getInputStream

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

Returns an input stream.

Returns:
an input stream
Throws:
`IOException` - if an I/O error occurred

---

getOutputStream

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

Returns an output stream.

Returns:
an output stream
Throws:
`IOException` - if an I/O error occurred

---

getConfiguration

```
SerialPortDTO getConfiguration()
```

Gets the Serial port configuration.

Returns:
SerialPortDTO object represents the configuration

---

setConfiguration

```
void setConfiguration(SerialPortDTO dto)
               throws SerialDeviceException
```

Sets the Serial port configuration.

Parameters:
`dto` - SerialPortDTO object represents the configuration
Throws:
`SerialDeviceException` - if the parameter is specified incorrectly or the parameter is not supported.

---

isDTR

```
boolean isDTR()
```

Gets the DTR state.

Returns:
the DTR state

---

isRTS

```
boolean isRTS()
```

Gets the DTS state.

Returns:
the DTS state

---

isDSR

```
boolean isDSR()
```

Gets the DSR state.

Returns:
the DSR state

---

isCTS

```
boolean isCTS()
```

Gets the CTS state.

Returns:
the CTS state

---

setDTR

```
void setDTR(boolean dtr)
     throws SerialDeviceException
```

Sets the DTR state.

Parameters:
`dtr` -

- `true` on DTR
- `false` off DTR

Throws:
`SerialDeviceException` - if the parameter is specified incorrectly or the parameter is not supported.

---

setRTS

```
void setRTS(boolean rts)
     throws SerialDeviceException
```

Sets the RTS state.

Parameters:
`rts` -

- `true` on RTS
- `false` off RTS

Throws:
`SerialDeviceException` - if the parameter is specified incorrectly or the parameter is not supported.

Class SerialDeviceException

org.osgi.service.serial

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

All Implemented Interfaces:

Serializable

```
public class SerialDeviceException
extends Exception
```

Constructor Summary	Page
SerialDeviceException(String message)	26

Constructor Detail

SerialDeviceException

```
public SerialDeviceException(String message)
```

Class SerialEvent

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.SerialEvent
```

```
public class SerialEvent
extends Object
```

Field Summary		Page
static int	DATA_AVAILABLE	27
int	dataType	27

Constructor Summary	Page
SerialEvent()	27

Field Detail

DATA_AVAILABLE

```
public static final int DATA_AVAILABLE = 1
```

---

dataType

```
public int dataType
```

Constructor Detail

SerialEvent

```
public SerialEvent()
```

Interface SerialEventListener

org.osgi.service.serial

```
public interface SerialEventListener
```

Serial events are sent using the whiteboard model, in which a bundle interested in receiving the Serial events registers an object implementing the SerialEventListener interface. A COM port name can be set to limit the events for which a bundle is notified.

Field Summary		Page
String	**SERIAL_COMPORT** Key for a service property that is used to limit received events.	28

Method Summary		Page
void	**notifyEvent**(SerialEvent event)	28

Field Detail

SERIAL_COMPORT

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

Key for a service property that is used to limit received events.

Method Detail

notifyEvent

```
void notifyEvent(SerialEvent event)
```

Class SerialPortDTO

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.SerialPortDTO
```

```
public class SerialPortDTO
extends Object
```

An object represents the Serial port configuration

Field Summary		Page
int	baudRate	29
int	dataBits	29
int	flowControl	29
int	parity	29
int	stopBits	29

Constructor Summary	Page
SerialPortDTO()	30

Field Detail

baudRate

```
public int baudRate
```

---

dataBits

```
public int dataBits
```

---

flowControl

```
public int flowControl
```

---

parity

```
public int parity
```

---

stopBits

```
public int stopBits
```

Constructor Detail

SerialPortDTO

```
public SerialPortDTO()
```

Class StopBits

org.osgi.service.serial

```
java.lang.Object
  └ org.osgi.service.serial.StopBits
```

```
public class StopBits
extends Object
```

Field Summary		Page
static int	ONE Stop bits: 1.	31
static int	ONE_AND_HALF Stop bits: 1.5.	31
static int	TWO Stop bits: 2.	31

Constructor Summary	Page
StopBits()	31

Field Detail

ONE

```
public static final int ONE = 1
```

Stop bits: 1.

TWO

```
public static final int TWO = 2
```

Stop bits: 2.

ONE_AND_HALF

```
public static final int ONE_AND_HALF = 3
```

Stop bits: 1.5.

Constructor Detail

StopBits

```
public StopBits()
```

8 Considered Alternatives

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