EnOcean Device Service Specification

Final

32 Pages

Abstract
This specification defines the Java API to discover and control EnOcean devices on the OSGi platform and according to OSGi service design patterns. This API maps the representation of EnOcean entities defined by EnOcean Equipment Profiles standard into Java classes. OSGi service design patterns are used on the one hand for dynamic discovery, control and eventing of local and networked devices and on the other hand for dynamic network advertising and control of local OSGi services.

0 Document Information

License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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.

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.

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/.

Table of Contents

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 .

Source code is shown in this typeface.

Revision History

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

Revision	Date	Comments
Initial	*February, 26th, 2013*	*Maïlys Robin, France Telecom Orange, mrobin.ext@orange.com* *Victor Perron, France Telecom Orange, victor.perron@orange.fr*
First draft	*April 4th, 2013*	*A.Bottaro, France Telecom Orange,* *M.Robin, France Telecom Orange,* *V.Perron, France Telecom Orange*
Revision 1	*April 18th, 2013*	*V.Perron, France Telecom Orange* • *Rename EO* concepts into EnOcean** • *Use a .learnedDevices property instead of getLearnedDevices()* • *Add a link between "Client Bundle" and "EnOceanDevice"*
Revision 2	*May 14th, 2013*	*V.Perron, France Telecom Orange* *A.Bottaro, France Telecom Orange* • *Addition of the EnOceanMessage, EnOceanChannel, EnOceanScaledChannel, EnOceanEnumChannel, EnOceanEnumChannelRange interfaces* • *Rewrite of the EnOceanProfile, EnOceanRPC, EnOceanRMCC specification* • *Revision of the main and EnOceanDevice diagrams* • *Addition of the known EnOcean Exceptions* • *Addition of the EnOcean Event API section* • *EnOcean networking explanations*
Revision 3	*May 20th, 2013*	*V. Perron, France Telecom Orange* *A.Bottaro, France Telecom Orange* *N. Portinaro, Telecom Italia* *A. Kraft, Deutsche Telekom* • *Take N. Portinaro's and A. Kraft's remarks about send() standardization and level of detail* • *Remove EnOceanProfile notion in profit of EnOceanMessage.* • *Merged together RPC and RMCC notions.* • *The heavy changes to EnOceanMessage and EnOceanChannel types Introduced the EnOceanChannelDescription type that follows a more common design with UpnP and Zigbee device services.*

Revision	Date	Comments
Revision 4	*May 27th, 2013*	*V. Perron, France Telecom Orange* *A.Bottaro, France Telecom Orange* • *Add support for Security* • *Challenge Generic Profiles support* • *Convergence towards EnOcean Link notions of Channels* • *Improve EnOceanHost notion into EnOceanGatewayChip* • *EnOceanDevice EXPORT situation should work.*
Revision 5	*June 2nd, 2013*	*V. Perron, France Telecom Orange* *A.Bottaro, France Telecom Orange* *N. Portinaro, Telecom Italia* • *Discussion is ongoing within OSGi members about the use of EnOceanHost as a low-level notion or bundled inside of the base driver; for now, only chip configuration is available, not send methods.* • *Use protected getters and setters instead of plain properties for security objects.* • *Add the repeater notion to EnOcean device.* • *Move any non-filtering property to a method form.* • *Discussed setChannels() and getChannels() methods that would allow for a generic implementation of a Message, finally not integrated.*
Revision 6	*June 8th, 2013*	*V. Perron, France Telecom Orange* *A.Bottaro, France Telecom Orange* • *Remove the Repeater notion from EnOcean Device and keep it only at the EnOceanHost level.* • *Add the sendSecureTeachIn() method* • *Some overall cleanup; question to reintegrate SmartAck.*

Revision	Date	Comments
Revision 7	*June 19th, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* *N. Portinaro, Telecom Italia* • *Overall cleeanups: the RFC's style has been rewritten in order to have less inclusions of Java-like text and be more descriptive. The Java specification, generated from the Javadocs, has been move to the end of the document, as what has been done with other service specifications.* • *EnOceanDevice: sendTeachIn, sendSecureTeachIn are removed in favor of a send(TeachInMessage). Provide setters for the dynamic, implementation-independant properties, like the senderId, security features, etc.* • *EnOceanMessage: the STATUS field is no more a filtering property, it carries not enough information and changes too often to be used as such. A getSubMessageCount() method has been added to help serializers in the case of multiple-frame messages. Those should be supported by the implementation transparently.* • *EnOceanChannel: Add the rawValue property that stores the value of the channel in bytes. Add setRawValue() and setValue() methods to enable for dynamic rewrite of the values.* • *EnOceanEnumChannelDescription / EnOceanScaledChannelDescription: define them as subinterfaces of the EnOceanChannelDescription interface. Make the serialization operations generic to the top-level EnOceanChannelDescription interface. Use doubles instead of floats in scaled channels.* • *Remove references to SmartAck and make it clearer that it will not be included for this iteration of the specification.* • *Still keep using only INTERFACES in this specification, but add methods to add/set properties. A bundle that would like to implement a "generic" Device/Message/etc class could then use those methods to do so.*

<table>
<tr><th>Revision</th><th>Date</th><th>Comments</th></tr>
<tr><td>Revision 8</td><td>July 9th, 2013</td><td>V. Perron, France Telecom Orange
A. Bottaro, France Telecom Orange
N. Portinaro, Telecom Italia
A. Kraft, Deutsche Telekom
E. Grigorov, Prosyst
K. Hackbarth, Deutsche Telekom

Rename EnOceanTelegram into EnOceanMessage. Fits better to EnOcean idea of a high-level, multipart message.
Add dBm and redundancy information to EnOceanMessage object. Every EnOceanMessage is sent by burst of three; knowing how many have been actually received, and at which average power level, can help giving an idea of the link quality.
Narrow EnOceanHost's calpabilities to "what should be awaited from a Gateway device" more than "what can ESP do"; we should, as it's done with Zigbee and the ZCL, not stick to ESP for Gateways, since some hardware vendors would not follow it anyway.
Datafields have been renamed to Channels, to stick better to EnOcean notions. Enumerated channels have been split into Enumerated as before, and a Flag type that describes boolean channels.
</td></tr>
<tr><td>Revision 9</td><td>July 31st, 2013</td><td>V. Perron, France Telecom Orange
A. Bottaro, France Telecom Orange
M. Robin, France Telecom Orange

EXPORT scenario: BD chooses the appropriate dongle, associate service PID and sender ID propose an optional API to retrieve the sender ID or deassociate it.
EnOceanHost: remove ability to send messages (role of the BD) but add an API to retrieve the sender ID associated to a service PID, if allocated within that chip's ID pool.
Requirements: EnOceanDevice properties such as profile info, security info... MUST be persisted to survive a framework reset; those properties can only be retrieved during an (often manual) teach-in procedure.
EnOceanDevice: for imported devices, there is a CHIP_ID property that is set by the BD. For exported devices, there is no such property, but an ENOCEAN_EXPORT property is there. In both cases, a SERVICE_PID property is present and unique.
</td></tr>
</table>

Revision	Date	Comments
Revision 10	*Aug. 17th, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* • *Every reference to RMCC has been removed, in favour of a united RPC notion.* • *Corrected main class diagram to make Set interfaces appear better, remove faulty <<Set>> notion.* • *Add the notion of EnOceanChannelDescriptionSet, EnOceanRPCSet, EnOceanMessageSet in Entities* • *Clarify the Operations Summary section: full update.* • *Clarify and rewrote EnOceanDevice section; move 'Export' section there and merge its information.* • *Rewrite EnOceanMessage section, cleanup artifacts from previous versions.* • *Reword the EnOceanChannel section; remove the notion of Shortcut and Friendly Name, those are not standard nor used;* • *Corrected the EnOceanChannelDescription part deeply; now more precise about description sets, new class diagram, introduced the Flag channel better, cleaned up outdated examples, introduce an unique identifier that is to be set for every Description class.* • *Rewrote the EnOcean Remote Management part; is clearer and standardized EnOceanRPCHandler into an EnOceanResponseHandler, with a notifyResponse() handler.* • *Confirm that there is no generic deserialization of the EnOceanRPC byte[] payload as of this specification, since EnOcean remote management is still extremely rare and there is no actual specification of it yet.* • *Moved EnOcean Networking and Security sections to a new "Annex" section at the end of the document.*

Revision	Date	Comments
Revision 11	*Aug. 24th, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* • *Remove the getValue() / setValue() interfaces from the EnOceanChannel object, will rely on EnOceanChannelDescription only. Keep get/setRawValue().* • *List of paired devices is an ID-based list.* • *List of available RPC is an ID-based list.* • *Introduce the 'EXTRA' subtype of an EnOcean message, which in some rare cases is required to further uniquely identify the message type in a Set.* • *The securityLevelFormat property of a device cannot be stored as a service property since it is discovered later on with a dedicated teach-in message. Same goes for non-essential properties such as the Name and ProfileName of the device.* • *Every registered Set object must provide a PROVIDER_ID and VERSION identifiers to not be in conflict with others. No more constraints are specified.* • *An EnOceanHost object is registered as a Host, but not as a device: it bears no profile information.* • *Add details about the format and constraints of EnOceanChannelDescription unique Ids.*
Revision 12	*Aug 14th, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* • *Teach-in standard procedure may include Manufacturer ID, it is now a registered property of EnOceanDevice.* • *Emphasize the 'silent dropping' behaviour of the BaseDriver for any message coming from an unknown peer.* • *Filters and device properties confirmed as String objects. No embedded objects in Event Admin.*

Revision	Date	Comments
Revision 13	*Oct 10, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* *N. Portinaro, Telecom Italia* *E. Grigorov, Prosyst* • *Introduce EnOceanMessageDescriptionSet interface.* • *EnOceanMessageSet : PROVIDER_ID and VERSION registration properties MAY be specified.* • *The EnOceanChannel.getDescription() has been pulled off, and we'll rely on the service registry to get them.* • *Add more details concerning Event Admin topics and properties.specially, how Generic Messages may be sent over without any message description registered.* • *EnOceanMessage.setStatus() method removed, since the status of a message is an instance-constant.* • *EnOceanMessage.deserialize() actually uses data bytes from the EnOcean Serial packet in order to include signal strength information, repeating status, and so on.* • *EnOceanMessages may need an “extra” identifier from time to time, apart from the RORG-FUNC-TYPE. Sometimes it's gonna be the “direction” parameter, or something else.*
Revision 14	*Nov 12, 2013*	*V. Perron, France Telecom Orange* *A. Bottaro, France Telecom Orange* *N. Portinaro, Telecom Italia* *E. Grigorov, Prosyst* *M. Hönsch, EnOcean* • *Remove the PROVIDER_ID/VERSION identifiers from all Sets.* • *EnOceanDataChannelDescription : input DOMAIN and output RANGES, mathematic standard denominations.* • *EnOceanDevice: remove NAME & PROFILE_NAME service properties.* • *EXPORT situation: documentation update and clarifications.* • *Updates to EnOceanDevice's send method, renamed to invoke and used solely for EnOceanRPC.* • *Introduction of* enocean.device.export *property on Event Admin to carry exported device's messages onto EnOcean network.* • *Rename EnOceanResponseHandler to EnOceanHandler, keep only the RPC return method.* • *EnOceanDevice supports external modification of profile for manual input, necessary for some devices.*

1 Introduction

EnOcean is a standard wireless communication protocol designed for low-cost and low-power devices by EnOcean GmBh.

EnOcean is widely supported by various types of devices such as smart meters, lights and many kinds of sensors in the residential area. OSGi applications need to communicate with those EnOcean devices. This specification defines how OSGi bundles can be developed to discover and control EnOcean devices on the one hand, and act as EnOcean devices and interoperate with EnOcean clients on the other hand. In particular, a Java mapping is provided for the standard representation of EnOcean devices called EnOcean Equipment Profile.

The specification also describes the external API of an EnOcean Base Driver according to Device Access specification, the example made by ZigBee Device Service specification and spread OSGi practices on residential market.

2 Application Domain

System Architecture

When installing a new EnOcean network into a residential network with an OSGi home gateway, there are 2 options:

- Add EnOcean communication capability to your home gateway, with an additional hardware such as a USB device called "dongle" and then add the necessary software (bundles) to interpret the EnOcean messages.
- Replace the current home gateway with one featuring EnOcean communication.

In both cases OSGi applications call the EnOcean driver API to communicate with the EnOcean devices as shown in Figure 1.

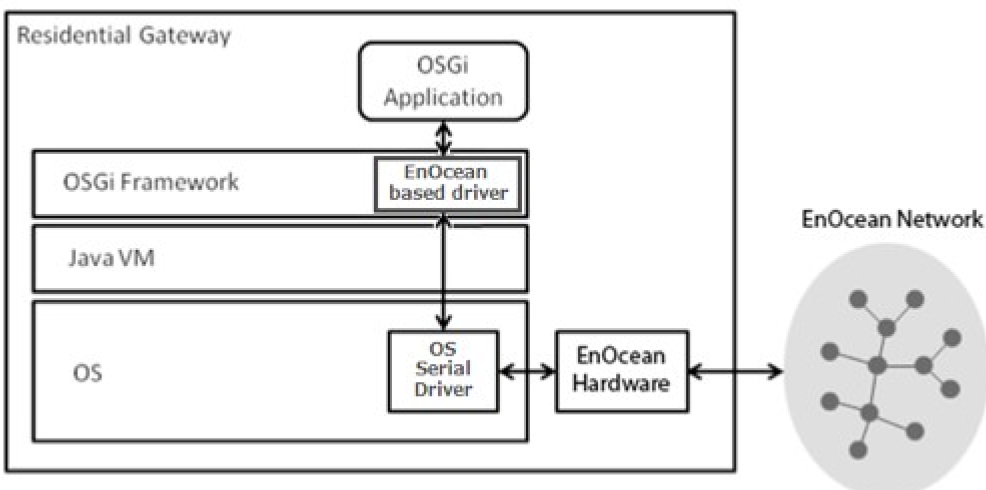


Figure 1: Communication with EnOcean devices through an EnOcean driver

The EnOcean specification defines two main types of devices: the transmitters and the receivers. Some receivers can be used as repeaters, and therefore are using bidirectional communication. The transmitters are using unidirectional communication only.

The very recent '*Smart Ack*' specification now enables transmitters to stay active for a few milliseconds after a transmission in order to receive messages from a remote device. For this to be possible, "mailboxes" have to be enabled on line-powered devices.

The EnOcean network is mainly composed of those transmitters paired to receivers through a "teach-in" procedure. It is a many-to-many model with no particular hierarchy, the opposite of a star network like Zigbee where every device relies on a single coordinator.

In this respect, the EnOcean gateway's hardware is no more and no less than an universal EnOcean transceiver, for which the “teach-in” and control procedures have to be software-defined.

EnOcean Stack

The EnOcean stack is shown in Figure 2. The three bottom layers, the **PHYSICAL** layer (not shown in the figure), the **DATALINK** layer and the **NETWORK** layer are defined by the ISO/IEC14543-3-10 standard, which is a new standard for the wireless application with ultra-low power consumption.

The EnOcean standard defines the **Application** and **Security** layers; it also defines:

1. The EnOcean Serial Protocol (ESP) for serial communication between a host and capable EnOcean modules;
2. The EnOcean Radio Protocol (ERP) defines packeted radio communication between EnOcean nodes;
3. Smart-Ack describes the use of “Mailboxes” on line-powered devices to send messages to energy-harvesting transmitters;
4. The EnOcean Equipment Profiles (EEP), described in detail in the next section, defines standard device profiles to be used by EnOcean devices.

The ISO standard enabled the physical, data link and network layers to be available for all, while the EnOcean application layers are available by joining the EnOcean Alliance.

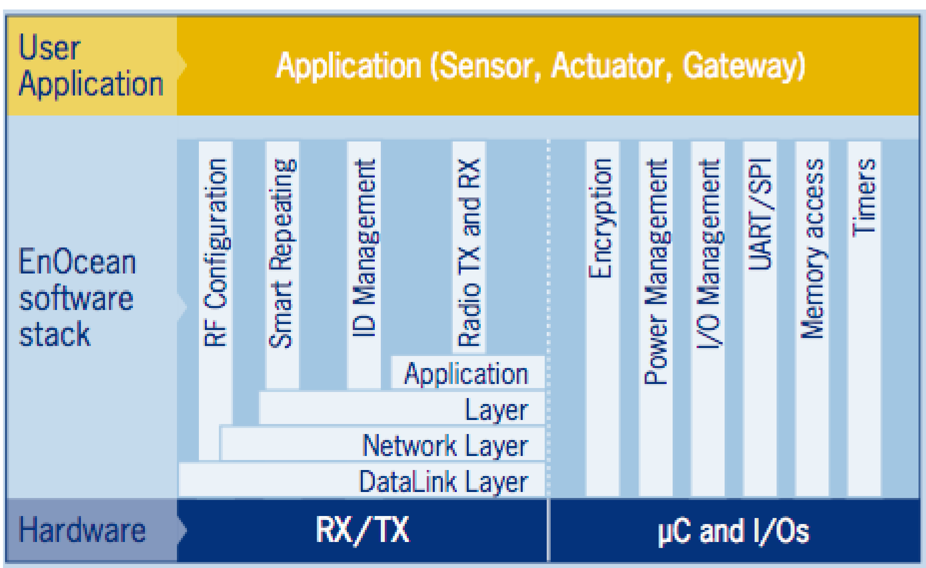


Figure 2: EnOcean Stack (source: EnOcean Website)

EnOcean Equipment Profiles (EEP)

The EnOcean Equipment Profiles enables interoperability between products developed by different vendors. For example, in a light control scenario, switches developed by a vendor can turn on and turn off lights developed by another vendor if both vendors are aware of each other's EEP Profiles. The EnOcean Alliance draws up the specifications for the applications based on the standard.

A device's EEP profile is fully defined by three combined elements:

- Its EnOcean Radio Protocol radio message type (**RORG, 8 bits**)
- Basic functionality of the data content (**FUNC, 6 bits**)
- Type of device; it refines the main functionality given by FUNC (**TYPE, 7 bits**)
- Extra information: in a few rare cases, messages are not entirely defined by those three identifiers, and some extra information is needed, such as a 'DIRECTION' or 'CATEGORY' information.

There are currently around 100 profiles defined.

When the existing profiles are not adequate, it is possible to create a new profile. Once developed, it should be submitted to the technical working group of the EnOcean Alliance.

3 Problem Description

With the increasing number of EnOcean vendors, the number of manufacturer-specific APIs is also raising, causing the following problems:

- Application developers cannot rely on standard EnOcean hardware interoperability within the target residential gateway's environment.
- An application that was developed for a given environment may not work in other environments without significant changes.

Those problems make it difficult for third parties to develop portable OSGi applications communicating with EnOcean devices.

The standard Java API requested in this RFC for the access of EnOcean devices would give developers a unified way of communicating with EnOcean devices, allowing developers to rely on a single, vendor-agnostic API.

4 Requirements

R1: The solution **MUST** provide an API for controlling EnOcean devices.

R2: The solution **MUST** provide a base driver interface as an OSGi service for the following operations: device and service discovery, network management, binding management, device management.

R3: The solution **SHOULD** enable applications to trigger a re-scan of the network to refresh the registry with actual EnOcean device services.

R4: The solution **MUST** provide a mechanism which notifies OSGi applications of events occurred in the EnOcean network and devices.

R5: The solution **MUST** register a Device Service object representing each found EnOcean device into Service Registry and unregister the Device Service object when the EnOcean device is unavailable or has not sent updates since a very long time.

R6: The solution **MUST** associate an EEP profile for each found EnOcean device and update the EEP if it is changing.

R7: The solution **MUST** be able to add new profiles to the existing ones (in the case of a new profile is created by a member of the EnOcean Alliance).

R8: The solution **MAY** define the driver provisioning process in accordance with the OSGi Device Access specification.

R9: The solution **MUST** be independent from the physical interface used to control the EnOcean network. The solution **MUST** likewise work with network controllers based on EnOcean built-in chips, EnOcean USB dongles and high level protocols offered by EnOcean Gateway Devices compliant with the EnOcean Alliance specification.

R10: The solution **MUST** include device access control based on user and application permissions compliant with the OSGi security model.

5 Technical Solution

6 Initial Spec Chapter

Essentials

- *Scope* – This specification is limited to general device discovery and control aspects of the standard EnOcean specifications. Aspects concerning the representation of specific or proprietary EnOcean profiles is not addressed.
- *Transparency* - EnOcean devices discovered on the network and devices locally implemented on the platform are represented in the OSGi service registry with the same API.
- *Lightweight implementation option* – The full description of EnOcean device services on the OSGi platform is optional. Some base driver implementations may implement all the classes including EnOcean device description classes while Implementations targeting constrained devices are able to implement only the part that is necessary for EnOcean device discovery and control.
- *Network Selection* – It must be possible to restrict the use of the EnOcean protocols to a selection of the connected devices.
- *Event handling* – Bundles are able to listen to EnOcean events.
- *Discover and control EnOcean devices as OSGi services* – Available learnt (via an EnOcean teach-in procedure) EnOcean external endpoints are dynamically reified as OSGi services on the service registry upon discovery.
- *OSGi services as exported EnOcean devices* – OSGi services implementing the API defined here and explicitly set to be exported should be made available to networks with EnOcean-enabled endpoints in a transparent way.

Entities

- *EnOcean Base Driver* – The bundle that implements the bridge between OSGi and EnOcean networks. It is responsible for accessing the various EnOcean gateway chips on the execution machine, and ensures reception and translation of EnOcean messages into proper objects. It is also used to send messages on the EnOcean network, using whatever chip it deems most appropriate.
- *EnOcean Host* – The EnOceanHost object is a link between the software and the EnOcean network. It represents the chip configuration (gateway capabilities) described in ESP3[9]. It is registered as an OSGi service.
- *EnOcean Device* – An EnOcean device. This entity is represented by a EnOceanDevice interface and registered as a service within the framework. It carries the unique chip ID of the device, and may represent either an imported or exported device, which may be a pure transmitter or a transceiver.
- *EnOcean Messages* – Every EnOcean reporting equipment is supposed to follow a "profile", which is essentially the way the emitted data is encoded. In order to reflect this standard as it is defined in the EEP[5]., manufacturers are able to register the description of "Messages", the essence of a profile, along with their associated payload (as Channels). See "EnOcean Channels" below for more information.
- *EnOcean Channel* – EnOcean channels are available as an array inside EnOceanMessage objects. They are a useful way to define any kind of payload that would be put inside of an EnOcean Message.

  EnOcean Messages and their associated Channels can be described with EnOceanMessageDescription and EnOceanChannelDescription interfaces. Description providers aggregate these descriptions in sets that they register with EnOceanMessageDescriptionSet and EnOceanChannelDescriptionSet interfaces within the framework.

  The mechanism allows in particular a lightweight implementation of the EnOcean device service platform, by leaving the possibility not to implement the unnecessary message or channel descriptions.
- *EnOcean RPC* – An interface that enables the invocation of vendor-specific Remote Procedure Calls and Remote Management Commands. These are particular types of Messages and are not linked to any EnOcean Profile, so that their descriptions are defined and registered in another way. The RPCs are documented via the EnOceanRPCSet interface.
- *EnOcean Response Handler* – Enables clients to asynchronously get answers to their Messages and RPCs.
- *EnOcean Client* – An application that is intended to control EnOcean device services.
- *EnOcean Exception* – Delivers errors during EnOceanMessage serialization/deserialization or during execution outside transmission.

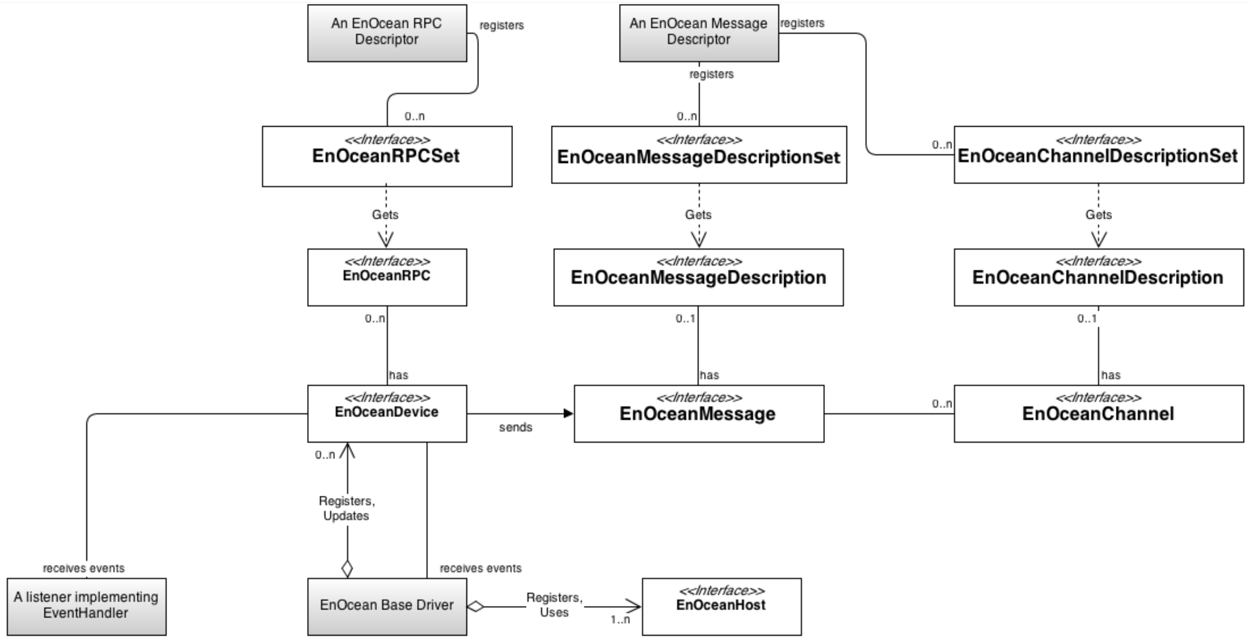


Figure 3: EnOcean Service Specification, Class Diagram

Operation Summary

To make an EnOcean device service available to EnOcean clients on the OSGi platform, it must be registered under the **EnOceanDevice** interface within the OSGi framework.

The EnOcean Base Driver is responsible for mapping external devices into **EnOceanDevice** objects, through the use of an EnOcean gateway. The latter is represented on OSGi framework as an object implementing **EnOceanHost** interface. EnOcean “teach-in” messages will trigger this behaviour, this is called a **device import** situation.

Client bundles may also expose framework-internal (local) EnOceanDevice instances, registered within the framework. The Base Driver then should emulate those objects as EnOcean devices on the EnOcean network. This is a **device export** situation, made possible by the use of the 127 virtual base IDs available on an EnOcean gateway. For more information about this process, please report to the “Exporting an EnOcean device” section below.

Updates concerning Messages emitted and received by the EnOcean devices are accessible through the proper Event Admin subscription, described below.

EnOcean Base Driver

Most of the functionality described in the operation summary is implemented in an EnOcean *base driver*. This bundle implements the EnOcean protocol and handles the interaction with bundles that use the EnOcean devices. An EnOcean base driver is able to discover EnOcean devices on the network and map each discovered device into an OSGi registered EnOceanDevice service.

It also is the receptor, through **EventAdmin** service and OSGi service registry, of all the events related to local devices and clients. It enables bidirectional communication for RPC and Channel updates.

EnOcean Host

The EnOcean host represents an EnOcean gateway chip. Any EnOcean device service implementation should rely on at least one Gateway Chip in order to send and receive messages on the external EnOcean network.

This interface enables standard control over an EnOcean compatible chip.

Every EnOceanHost object should at least be identified by its unique chip ID.

The EnOceanHost interface enables OSGi applications to:

- Get or set gateway metadata (version, name, etc);
- Reset the gateway chip device;
- Retrieve a Sender ID (derived from EnOcean's BASE_ID) for the given Service PID of a device.
- Broadcast a message, or plain raw bytes, on the EnOcean network.
- Send an addressed message on the EnOcean network.

EnOcean Device

Generics

A physical EnOcean device is reified as an **EnOceanDevice** object within the framework. Any **EnOceanHost** is also an EnOceanDevice, but the to concepts are not linked by any inheritance.

An EnOcean device holds most of the natural properties for an EnOcean object: its unique ID, the profile, a friendly name, its security information, and its available RPCs – along with the associated getters (and setters when applicable). All those properties MUST be persistent across restart so that teach-in procedures are made only once.

It also holds methods that reflect the natural actions a user application may physically trigger on such a device: send a message to the device, send a teach-in message to the device, or switch the device to learning mode.

Every EnOcean Device keeps a service PID property that is assigned either by the base driver or by any service-exporting bundle. The property value format is free and the value must be unique on the framework.

The properties on which EnOceanDevice services can be filtered on are: the device's service PID and chip ID, and its profile identifiers (RORG / FUNC / TYPE integers).

The EnOceanDevice also keeps security features as defined in the EnOcean Security Draft[9]., which allow for a security level format (integer mask), an encryption key and/or a rolling authentication code. See the Security Of EnOcean networks section below.

The EnOceanDevice service MUST also be registered with org.osgi.service.device.Constants.DEVICE_CATEGORY property (see OSGI Compendium: 103 Device Access Specification) that describes a table (String[]) of categories to which the device belongs. One value MUST be “EnOcean” (org.osgi.service.enocean.EnOceanDevice.DEVICE_CATEGORY).

The additional properties (defined in Device Access – 103.2.1) : DEVICE_DESCRIPTION, DEVICE_SERIAL are not set in the EnOcean context as no description nor application-level serial number are provided in the protocol.

Import Situation

In **import** situations, the device's chip ID is uniquely set by the Base Driver, according to the one present in the teach-in message that originated the Device's creation. The service PID (cf. Core Specification R4 v4.3, section 5.2.5) should also be generated and deterministically derived from the chip ID to allow reconstruction of a device without a new teach-in process after a framework restart.

Export Situation

In **export** situations:

1. The registering Client bundle sets the service PID of the **EnOceanDevice** object by itself, in a unique manner, and registers that object.
2. The **chip ID** (this device's EnOcean source ID when it issues messages) will be allocated by the Base Driver. The latter keeps a dictionary of the currently allocated chip IDs. The Client bundle must also set an ENOCEAN_EXPORT property in the registered device's Property Map.

The standard way to programmatically retrieve an exported chip ID from a given service PID is by using **EnOceanHost's** dedicated interface for this use.

The Base Driver **MUST** ensure the persistency of the CHIP_ID:SERVICE_PID mapping.

Optionally, the Base Driver could provide an API to retrieve the chip ID associated to a particular service PID, or be instructed to erase such an association.

As an application developer, please refer to the documentation of your Base Driver to know its policies concerning exported chip ID updating, deletion and exhaustion.

Interface

The **EnOceanDevice** interface enables client bundles to:

- Get or set the security features of the Device in a protected way;
- Retrieve the currently paired devices in the case of a receiver, as a collection of device IDs;
- Get the ID-based list of currently available RPCs for the device, as a Map of {manufacturerID: [commandId1, commandId2, ...] };
- Invoke RPCs onto the device, through the invoke*()* call.

EnOcean Messages

Introduction

EnOcean Messages are at the core of the EnOcean application layer as a whole and the EnOcean Equipment Profile specification[5]. in particular. Every exchange of information within EnOcean networks is done with a dedicated message. In this specification we will be especially interested in a particular portion of an **EnOcean Serial Protocol Type 1 (RADIO)** message:

Group	Offset	Size	Field	Value Hex
DataPayload	6	x	…	...
OptionalData	6+x	1	SubTelNum	0xnn
	7+x	4	destinationID	0xnnnnnnnn
	11+x	1	dBm	0xnn
	12+x	1	SecurityLevel	0x0n

The Data Payload but also Number of Subtelegrams, Destination ID, Signal Strength and Security Information are made available to OSGi applications.

This model enables reading both the EnOcean radio telegram data and the associated metadata that may be attached to it in a single object, **EnOceanMessage**.

In case the 'Optional Data' section gets missing at the lowest level (the radio access layer not following ESP protocol for instance) it is the responsibility of the Base Driver to mock the missing field's (dBm, destinationID, …) values.

Mode of operation

Any EnOceanMessage object creation will be mirrored to Event Admin.

Details about the available topics, filters and properties can be found in the Event API section below.

EnOceanMessage objects will be created only if the originating device already has been registered in the OSGi Service Registry, along with profile information.

Identification

The RORG of a message defines its shape and generic type; all the RORGs are defined in the EnOcean Radio Specification.

An addressed message will be encapsulated into an Adressed Telegram (ADT) by the base driver transparently; this means that from the application level, it will be represented under its original RORG, but with a valid destinationID.

A particular EnOcean Equipment Profile message is identified by three numbers: its RORG, and its FUNC, TYPE and EXTRA subtypes. In EnOcean, a (RORG, FUNC, TYPE) triplet is enough to identify a profile; though an EXTRA identifier is sometimes needed to identify a particular message layout for that profile.

Those identifiers allow for retrieving **EnOceanMessageDescription** objects within a registered **EnOceanMessageDescriptionSet**, which give the application more information to parse the message.

Interface

The methods available in the **EnOceanMessage** interface are:

- Identification methods, retrieving the message's profile, sender ID, optional destination ID, status;
- A method to get the raw bytes of payload data in the message. This data can then be passed to the deserializer of the **EnOceanMessageDescription** object to be converted to EnOceanChannels, which may -again- be documented (through **EnOceanChannelDescription** objects) or not.
- Link quality information read-only methods that mirror some of the 'Optional Data' header information.
-

EnOcean Message Description

EnOceanMessageDescription objects exposes only two methods:

- *deserialize()*: makes the user able to deserialize the payload bytes of a raw EnOceanMessage object, into a collection of **EnOceanChannel** objects.
- *serialize()*: serializes the input **EnOceanChannel** objects into a collection of bytes.

EnOcean Channel

The EnOceanChannel interface is the first step of an abstraction to generate or interpret EnOceanMessage channels with plain Java types.

The simple **EnOceanChannel** interface provides a way to separate the different fields in a message payload, knowing their offset and size in the byte array that constitutes the full message's payload.

At the EnOceanChannel level, the only way to get/set the information contained in the channel is through a pair of *getRawValue()* and *setRawValue()* methods, which act on plain bytes.

Those bytes are meant right-aligned, and the number of those bytes is the size of the datafield, floored up to the next multiple of 8. For instance, a 3-bit long channel would be encoded on one byte, all the necessary information starting from bit 0.

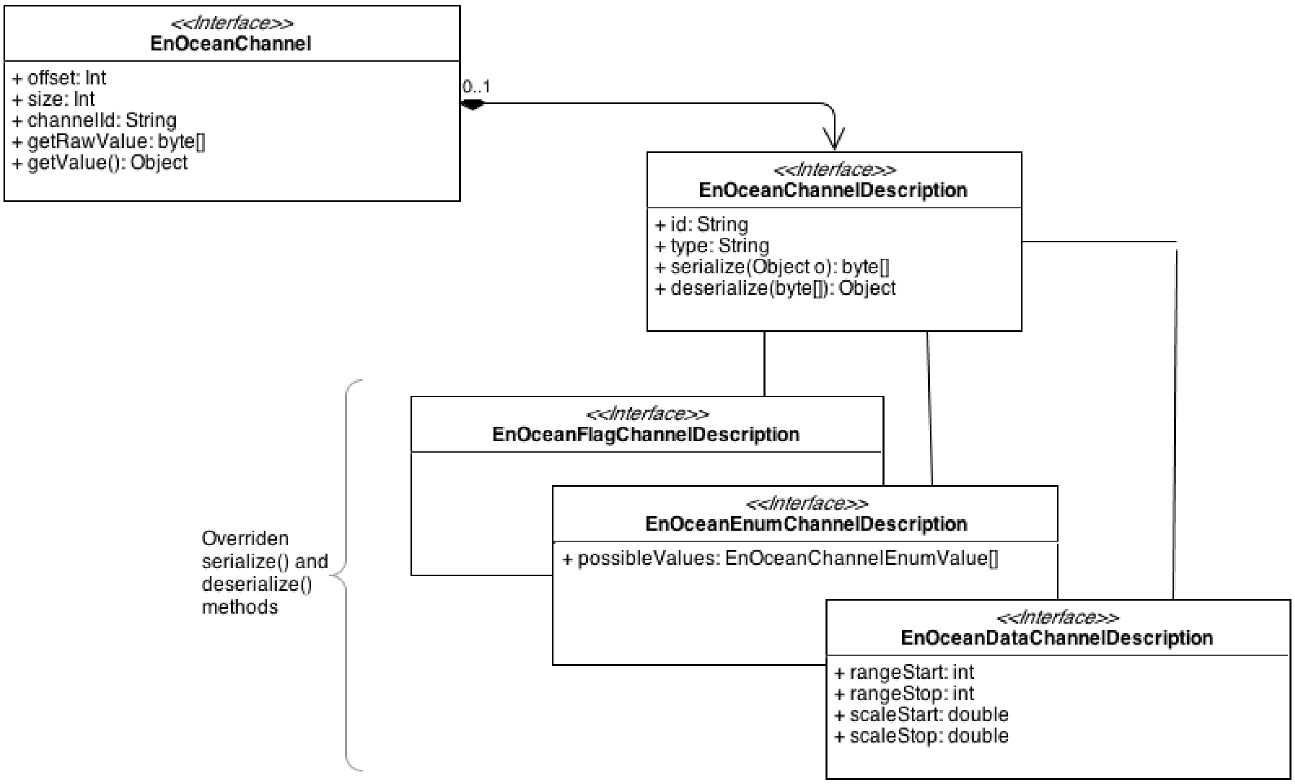
Every EnOceanMessage as described in the EEP Specification contains a various amount of channels, each of them being identified by their unique ID.

This ID, or **channelID**, is constituted of the “Shortcut” field of this channel from the EEP 2.5 Specification [5]. and a number fixed by the order of appearance of such a “Shortcut” in the specification.

This unique identifier links a Channel to an **EnOceanChannelDescription** object that provides more information to encode and decode that channel's information; see below for more details. This enables for loose coupling of the raw Channel itself and a richer, 3rd-party provided, information.

As an example, if the platform being developed is an electronic display that waits for Messages from a well-known temperature sensor, the Client bundle on the platform may interpret the Temperature Channels in every Temperature Message without needing an appropriate TemperatureChannelDescription object; it may directly cast and convert the **Byte[]** array of every received message to a properly valued **Double** and display that.

Otherwise, it could as well use the **channelID** to get a TemperatureChannelDescription object that would properly handle the deserialization process from the raw bytes to a proper, physical unit-augmented, result.



EnOcean Channel Description

The **EnOceanChannelDescription** interface enables the description of all the various channels as specified in the EnOcean specification, as well as the description of channels issued by 3rd party actors.

Those description objects are retrieved from the registered **EnOceanChannelDescriptionSet** interface using an unique ID known as the **channelID.**

Here are the Channel types defined in this specification:

- *CHANNEL_TYPE_RAW*: A collection of bytes. This type is used when the description is not provided, and is thus the default. For this type, the EnOceanChannelDescription's *deserialize()* call actually returns a *byte[]* collection. The encryption key or a device ID on 4 bytes are examples of such raw types.
- *CHANNEL_TYPE_DATA*: A scaled physical value. Used when the data can be mapped to a physical value; for instance, the 'WND – Wind Speed' channel is a raw binary value, in a range from 0 to 255, that will be mapped as a wind speed between 0 and 70 m/s. For this type, the EnOceanChannelDescription's *deserialize()* call actually returns a **Double** value.
- *CHANNEL_TYPE_FLAG*: A boolean value. Used when the Channel value can be either 1 or 0. The "*Teach-In*" Channel is a well-known example; this 1-bit field may either be 0 or 1, depending whether

the Message is a teach-in one or not. For this type, the EnOceanChannelDescription's *deserialize()* call actually returns a **Boolean** value.

- *CHANNEL_TYPE_ENUM*: An enumeration of possible values. Used when the Channel can only take a discrete number of values. More complicated than the Flag type, Enumerated types may have thresholds: for instance, the A5-30 "*Digital Input- Input State (IPS)*" channel is a 8-bit value which means "Contact closed" between 0 and 195, and "Contact open" from 196 to 255. For this type, the EnOceanChannelDescription's *deserialize()* call actually returns an **EnOceanChannelEnum** object.

According to the channel type, the actual description object should implement one of the following specialized interfaces. This will ease the use of casting to the specialized interfaces on documented channels.

EnOcean Data Channel Description

The EnOceanDataChannelDescription interface inherits from EnOceanChannelDescription interface.

Two more methods give access to the integer input domain of the data channel (such as 0-255) and to the floating-point output range of it (such as -30,0°C – 24,5°C). A method is also present to retrieve the physical unit of the channel. The serialize() and deserialize() methods are implemented to easily convert from the raw byte[] collection to a Double, and vice versa.

Here a a few samples of such Channels:

Short	Description	Possible implemented name	Domain	Range	Unit
TMP	Temperature (linear)	TemperatureScaledChannel_X	0..255	-10°..+30°	°C
HUM	Humidity (linear)	HumidityScaledChannel_X	0..250	0..100	%

EnOcean Flag Channel Description

The EnOceanFlagChannelDescription interface inherits from the EnOceanChannelDescription interface.

Those channels, typically are used for On/Off reporting values (like a switch) ; they have no additional methods, though the deserialize() method converts the input bit into a proper Boolean object.

EnOcean Enumerated Channel Description

The EnOceanEnumeratedChannelDescription interface inherits from the EnOceanChannelDescription interface.

The additional method provided to this interface is *getPossibleValues(),* which returns an array of the available **EnOceanChannelEnumValue** objects accessible to this channel. Every EnumValue object contains its integer input range and a String identifier that defines its meaning.

The serialize() and deserialize() methods of an EnOceanEnumeratedChannelDescription object thus convert an integer input value (say, 156) to an EnOceanChannelEnumValue, and vice versa.

Here is an example that shows the input range and the associated EnOceanChannelEnumValues.

Device profile	EnOceanChannelEnumValue	Start	Stop	Meaning
Fan speed stage switch	FanStageSwitch_Stage3	0	144	Fan speed: Stage 3
	FanStageSwitch_Stage2	145	164	Fan speed: Stage 2
	FanStageSwitch_Stage1	165	189	Fan speed: Stage 1
	FanStageSwitch_Stage0	190	209	Fan speed: Stage 0

EnOcean Remote Management.

Remote Management is a feature which allows EnOcean devices to be configured and maintained over the air using radio messages.

The Remote Procedure Calls, or RPCs -as defined by the EnOcean Remote Management specification[6].- are not related to any EnOcean Equipment Profile.

EnOcean RPC

An EnOceanRPC object enables client bundles to remotely manage EnOcean devices using already defined behaviour.

RPCs are mandatorily defined by a **MANUFACTURER_ID** (11 bits, 0x7FF for the EnOcean alliance) and a unique **FUNCTION_ID** code on 12 bits.

EnOceanRPC can be retrieved from an **EnOceanRPCSet** that had been previously registered within the OSGi framework by a documenting bundle, thanks to the unique MANUFACTURER_ID and FUNCTION_ID codes.

RPCs are called directly onto an **EnOceanDevice** object via the device.invoke() method, which accepts also a non-mandatory EnOceanHandler object as a parameter to retrieve the asynchronous answer.

Broadcasted RPCs can be addressed directly to the Base Driver using the relevant Event Admin topic; see the Event API section below.

EnOcean Handler

Responses to RPCs are processed by the driver and sent back to a handler using **EnOceanHandler**'s *notifyResponse()* method when an EnOceanHandler is passed to the base driver.

Working With an EnOcean Device

Service Tracking

All discovered EnOcean devices in the local networks are registered under **EnOceanDevice** interface within the OSGi framework. Every time an EnOcean device appears or quits the network, the associated OSGi service is registered or unregistered in the OSGi service registry. Thanks to the EnOcean Base Driver, the OSGi service availability in the registry mirrors EnOcean device availability on EnOcean network.

Thanks to service events, a bundle is able to track the addition, modification and removal of an EnOceanDevice service.

Below stands an example showing how the tracking can be implemented. The sample Controller class extends the ServiceTracker class so that it can track all EnOceanDevice services.

```
public Class Controller extends ServiceTracker {
	public Object addingService(ServiceReference arg0) {
		Object service = context.getService(arg0);
		if (service != null && service instanceof EnOceanDevice) {
			eoDevice = (EnOceanDevice) service;
		}
		Logger.debug(service.getClass().getName() + " service found.", null);
		return service;
	}

	public void modifiedService(ServiceReference arg0, Object arg1) {
		/* Unimplemented */
	}

	public void removedService(ServiceReference arg0, Object service) {
		if (service instanceof EnOceanDevice) {
```

```
                        eoDevice = null;
                        Logger.debug("EnOceanDevice service was removed.", null);
                    }
                }
            }
```

Event API

EnOcean events must be delivered to the **EventAdmin** service by the EnOcean implementation, if present. EnOcean event topic follow the following form:

org/osgi/service/enocean/EnOceanEvent/SUBTOPIC

Here are the available subtopics:

MESSAGE_RECEIVED

Properties: (every event may dispatch some or all of the following properties)

- EnOceanDevice.CHIP_ID – (enocean.device.chip_id/String)

  The CHIP_ID of the sending device.

- Constants.SERVICE_PID – (service.pid/String)

  The service PID of the exported device.

- EnOceanDevice.RORG – (enocean.device.profile.rorg/String)

  The RORG (Radio Telegram Type) of the sending device.

- EnOceanDevice.FUNC – (enocean.device.profile.func/String)

  The FUNC profile identifier of the sending device.

- EnOceanDevice.TYPE – (enocean.device.profile.type/String)
- The TYPE profile identifier of the sending device.
- EnOceanEvent.PROPERTY_MESSAGE – (enocean.message/EnOceanMessage)

  The EnOceanMessage object associated with this event.

- EnOceanEvent.PROPERTY_EXPORTED – (enocean.message.is_exported/Object)

  The presence of this property means that this message actually has been exported from a locally implemented EnOcean Device.

RPC_BROADCAST

This event is used whenever an RPC is broadcasted on EnOcean networks, in IMPORT or EXPORT situations.

Properties: (every event may dispatch some or all of the following properties)

- EnOceanRPC.MANUFACTURER_ID – (enocean.rpc.manufacturer_id/String)

  The RPC's manufacturer ID.

- EnOceanRPC.FUNCTION_ID – (enocean.rpc.function_id/String)

  The RPC's function ID.

- EnOceanEvent.PROPERTY_EXPORTED – (enocean.message.is_exported/Object)

  The presence of this property means that this RPC actually has been exported from a locally implemented EnOcean Device.

- EnOceanEvent.PROPERTY_RPC – (enocean.rpc/EnOceanRPC)

  The EnOceanRPC object associated with this event.

EnOcean Exceptions

The EnOceanException can be thrown and holds information about the different EnOcean layers. Here below, ESP stands for "EnOcean Serial Protocol". The following errors are defined:

- FAILURE – (0x01) Operation was not successful.
- ESP_RET_NOT_SUPPORTED – (0x02) The ESP command was not supported by the driver.
- ESP_RET_WRONG_PARAM – (0x03) The ESP command was supplied wrong parameters.
- ESP_RET_OPERATION_DENIED – (0x04) The ESP command was denied authorization.
- INVALID_TELEGRAM – (0xF0) The message was invalid.

Java Interface Specification: org.osgi.service.enocean

EnOcean Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest.

For example: Import-Package: org.osgi.service.enocean; version="[1.0, 2.0)"

Summary

- Export the types discussed above.

Considered Alternatives

June 19, 2013:

The RFC's style has been changed to be more lightly descriptive on the actual Java types. The implementation efforts in parallel, on the opposite, are driving the actual interface specification that is now put and update in the "Java Interface Specification" paragraph above.

About the dynamic implementation of Messages, as proposed by N. Portinaro, it is decided to keep using interfaces only in the specification, and not define classes. Nevertheless, a sample of such a dynamic implementation using anonymous classes implementing those interfaces on-the-fly has been proposed as an example.

June 4, 2013:

It has been discussed whether or not to add a setChannels() or appendChannels() method to the EnOceanMessage interface. Unfortunately, trying to do so resulted in a cluttered and difficult to read interface for no clear benefits, so it has been decided not to add it yet.

The possibilty to send messages using the EnOceanHost interface has also been declined yet; this interface should be used for the configuration of the gateway chip only.

After an evaluation of the issues and rewards that would bring an implementation of the scarcely-used SmartAck protocol, it has been decided to set it aside for this iteration.

May 27, 2013:

The 'export' feature is not anymore set aside and should be challenged for consideration. The 'SmartAck' feature status is still under evaluation. The Security features of EnOcean have been integrated. Remote Management is integrated in a minimal fashion. Since there is no clear specification on the equivalent of the 'datatypes' for Remote Management, it has been decided yet (V.Perron) to set aside the development of abstractions for them and let the programmer implement extra methods above the specification when deemed useful.

April 03, 2013:

It has been decided (A.Bottaro, M.Robin, V.Perron) to set aside the 'export' feature of EnOcean device service for further reflexion, as well as 'SmartAck' feature from EnOcean, which will require extra effort. For this latter topic, one of the tracks worth exploring was the setup of Mailboxes at the EnOceanHost level (which sticks to the reality of the EnOcean gateway chips) and recommanding a dedicated 'real-time' channel to be implemented, so that SmartAck message frames could be carried synchronously.

7 Security Considerations

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

8 Annex

EnOcean Networking

EnOcean networking is a quite particular wireless network in the sense that there is no actual “topology”. Every device emits messages on the same frequency band, which depends on the world region and local regulations.

In Europe, the 868 MHz frequency band is used; in Asia, the 315 MHz is adopted. The 902 MHz band is in the process of being used for North America. There is no notion of a “network identificator” in EnOcean.

The transmitting devices usually broadcasts all of their messages on this frequency, and most of the time do not wait for an answer. The transmitting devices being mostly energy-harvesting devices, they cannot easily wait for an answer.

The receiver modules listen to every message sent on the frequency band. They filter the messages of interest based on the Sender ID that is embedded within every message. They are supposed to listen only to Sender ID that have previously been "taught" to them, and discard the others.

The teach-in procedure is specific to EnOcean. The receiver module has to be manually (or remotely, but that is still very rare) switched to a "learning mode". It will wait for a special kind of EnOcean messages, called "teach-in" messages. Those "teach-in" messages have to be sent by the emitting device that is targeted to be learnt by the receiver.

Once the receiver module has received this "teach-in" message, it should keep in non-volatile memory the sender ID of that message and such, be "paired" with it.

Because of this process, EnOcean networks are N-to-N: you may pair N emitters to 1 receiver, 1 emitter with N receivers, or even N emitters to M receivers.

In this respect, the EnOcean gateway is somewhat special; it is a device able to both send and receive messages, is line-powered, and listens to every message in the frequency band.

EnOcean Network Security

The security in EnOcean exists in a point-to-point fashion. The emitting device will be responsible of transmitting the optional Key and/or Rolling Authentication Code (RLC) to the receiver device during a dedicated "Security-Teach-In" phase.

The security configuration, Key and RLC are transmitted using a special message. The receiver device then associates the given key and RLC to that device internally, and uses them to decode any further message coming from it.

As a result, the Key and RLC parameters, as well as the current security configuration, are properties tied to a sending EnOceanDevice object; furthermore, an arbitrary number of security configurations, keys and RLCs may coexist within the same EnOcean network.

It will be the responsibility of the receiver object to fetch the current security properties of the sending object and use them to decode further messages.

9 Document Support

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]. André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive Service Composition in the Home Network", 21st IEEE International Conference on Advanced Information Networking and Applications (AINA-07), Niagara Falls, Canada, May 2007

[4]. Pavlin Dobrev, David Famolari, Christian Kurzke, Brent A. Miller, "Device and Service Discovery in Home

Networks with OSGi", IEEE Communications magazine, Volume 40, Issue 8, pp. 86-92, August 2002ASHRAE 135-2004 standard, Data Communication Protocol for Building Automation and Control Networks

[5]. EnOcean Equipment Profiles v2.5, EnOcean Alliance, March 04, 2013

[6]. EnOcean System Specification – Remote Management v1.7, EnOcean Alliance, December 16, 2010

[7]. EnOcean System Specification – Smart Acknowledgment v1.4, EnOcean Alliance, September 15, 2010

[8]. EnOcean System Specification – EnOcean Serial Protocol v1.17, EnOcean Alliance, August 2, 2011

[9]. EnOcean System Specification – Security of EnOcean Radio Networks v1.3, EnOcean Alliance, July 31, 2012

Author's Address

Name	André Bottaro
Company	France Telecom Orange
Address	28 Chemin du Vieux Chêne, Meylan, France
Voice	+33 4 76 76 41 03
e-mail	andre.bottaro@orange.com

Name	Maïlys Robin
Company	Orange Labs Tokyo
Address	Keio Shinjuku Oiwake Bldg. 9F, 3-1-13 Shinjuku, Shinjuku-ku, Tokyo 160-0022
e-mail	mrobin.ext@orange.com

Name	Victor Perron
Company	Orange Labs Tokyo
Address	230 rue La Fayette, 75010 PARIS
e-mail	victor.perron@orange.fr

Acronyms and Abbreviations

End of Document