ZigBee Device Service Specification

Draft

37 Pages

Abstract

This specification defines the Java API to discover, control and implement ZigBee devices on the OSGi platform and according to OSGi service design patterns. This API maps the representation of ZigBee entities defined by ZigBee Cluster Library 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 implementing this API.

0 Document Information

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.

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	*May, 16th, 2012*	*Andre Bottaro, Orange, andre.bottaro@orange.com*
1st Draft	*September, 20th, 2012*	*Bâle presentation*
	October, 16th, 2012	*API Summary Initialized*
	October, 18th, 2012	*ZigBeeClusterDescription and ZigBeeCommandDescription section initialized*
	December,14th, 2012	*Added details and references, cleared comments, fixed few mistakes.*

Revision	Date	Comments
v11-reg	*January, 15th*	*Andre Bottaro, Orange* *Jean-Pierre Poutcheu, Orange* *1. ZigBeeDeviceDescription and ZigBeeDeviceDescriptionSet classes are added and the registration of device descriptions is explained. The base driver and any bundle are now able to register the set of ZigBeeDeviceDescription objects which they have the knowledge. Those sets are registered with ZigBeeDeviceDescriptionSet interface.* *2. ZigBeeEvent.getCluster() added in order to be able to retrieve the ids of the devicenode, the endpoint and the cluster which attributes values are notified.* *3. Masaki 1st point: ZigBeeEndpoint now provides getDeviceNode() method in ZigBeeEndpoint class without any input argument.* *4. Masaki 2nd point: ZigBeeEndpoint class now provides a method to retrieve all available input ZigBeeCluster objects and a method to retrieve all output ones.* *5.Whenever getXXXid() can be changed into getId() without ambiguity, the change is made. The same change is applied to getXXXName, getXXXVersion(). For instance, ZigBeeCluster.getClusterId() is changed into ZigBeeCluster.getId().* *6. ZigBeeDataType.getJavaDataType() has now the same signature as UpnPDataType.getJavaDataType().* *7. ZigBeeHost.getPanId() is removed and the method is added to the parent class: ZigBeeDeviceNode.getPanId() is added.* *8. PAN_ID property was a property only specified for exported ZigBeeEndpoint services. It is now specified for all ZigBeeEndpoint services. Other properties are added to improve filtering features made on ZigBeeEndpoint services.* *9. An Endpoint was able to be registered once and exported on several networks by distinct hosts. This lead to an issue: which host to return in ZigBeeEndpoint.getDeviceNode() method? Thus, the spec has been changed: a distinct ZigBeeEndpoint object has now to be created and registered for every distinct targeted network (identified by a distinct PAN_ID)*

Revision	Date	Comments
v12-reg	*January, 29th*	*Andre Bottaro, Orange* *Jean-Pierre Poutcheu, Orange* 1. *Typed collections (Java 1.5) may remain in the javadoc. That's a bug and they are removed. (javadoc to be sent to the list later).* 2. *The link between ZigBeeDeviceDescription and ZigBeeClusterDescription was missing in the UML schema. It is now added.* 3. *Masaki's 4th point: Permit duration taken into account.* 4. *Standard properties are proposed for the ZigBeeDeviceDescriptionSet service. The right mapping with ZigBee standard names and the format of values is now applied.* 5. *The list of constant ZigBeeDataTypeDescription objects was missing. The developer needs to be able to retrieve those ZigBee constant objects. It is now specified in a new interface named "ZigBeeDataTypes".* 6. *Nicola's point on de/serialization of data types. isAnalog(), serialize/deserialise() method names taken into account.* 7. *Cardinality 0..1 is replaced by * when it involves a table or a vector of objects (attributedescs, clusterdescs, …).* 8. *Masaki 3rd point: a method « void ZigBeeEndPoint.notExported(ZigBeeException ze) » is added. Explanations are now in the Export section.* 9. *The ZigBeeDeviceDescriptionSet class was missing in the UML schema. It is now added. (and the 'ZigBee Cluster Descriptor' implementation (grey blox) is removed).*
v13-reg	*February, 5th*	*Andre Bottaro, Orange* *Jean-Pierre Poutcheu, Orange* *The ZigBee Extended PAN ID is now mentioned and used in the specification*

Revision	Date	Comments
v14-reg	*February, 25th*	*Andre Bottaro, Orange* *Jean-Pierre Poutcheu, Orange* *Thanks to Evgeni Grigorov's (Prosyst's) comments and Nicola Portinaro (Telecom Italia's) comments* 1. *Added a 'leave()' method in ZigBeeDeviceNode javadoc for removing nodes to request the device to leave the network:* *void leave(boolean rejoin, boolean request, boolean removeChildren, ZigBeeHandler handler)* 2. *Added a 'checkValue(Object obj)' in ZigBeeParameterDescription which returns true if the parameter value is valid according to his description and possible value ranges and other specific information.* 3. *Added new filters in listener, with names closer to ZCL documentation ones ZigBeeAttribute.REPORTABLE_CHANGE, ZigBeeAttribute.MIN_REPORT_INTERVAL, ZigBeeAttribute.MAX_REPORT_INTERVAL, ZigBeeAttribute.TIMEOUT_PERIOD* 4. *Added a 'public void setValue(Object value, ZigBeeHandler handler) throws ZigBeeException' method in ZigBeeAttribute* 5. *Added a description in 'Implementing a ZigBee Endpoint' about the use case where, an exportable endpoint corresponds to two more than 1 ZigBeeHost, at this time a ZigBeeException is thrown.* 6. *Added a paragraph to tell the reader that EndPoint 0 and 255 are not registered in the registry. And that EndPoint 241-255 should not be registered since these numbers are said "reserved for future use" in the ZB spec.*
v20-reg	*May, 6th*	*Jean-Pierre Poutcheu, Orange* *Arnaud Rinquin, Orange* 1. *ZigBeeAttributeHandler,notifyResponse(), use of Map instead of dictionary* 2. *Moved getAccesType() and isReportable() from ZigBeeAttribute to ZigBeeAttributeDescription* 3. *UNSIGNED_INTEGER_64 mapped with BigInteger Java class* 4. *ZigBeeCluster.readAttributeAsByte(...) has been removed* 5. *Added get and setChannel mask operations in ZigBeeHost*

Revision	Date	Comments
v21-reg	*May, 15th*	*Jean-Pierre Poutcheu, Orange* *Arnaud Rinquin, Orange* *1. Added a new Exception ZigBeeNoDescriptionAvailableException* *2. Added a new class ZigBeeAttributeRecord* *3. Modified ZigBeeCluster.writeAttributes(...) to* *- void writeAttributes(boolean undivided, int[] attributesIds, byte[] values, ZigBeeAttributesHandler handler) throws ZigBeeNoDescriptionAvailableException;* *- void writeAttributes(boolean undivided, ZigBeeAttributeRecord[] attributes, ZigBeeAttributesHandler handler)* *4. Change ZigBeeHandler.notifyResponse to notifyResponse(int Status, Map values)* *5. Nicola's Point : New package org.osgi.service.zigbee.descriptors for all the descriptors* *6. Nicola's Point: ZigBeeException use hex value are used for constants. (Thx to Nicola)* *7. Evgeni's Point: Removed ZigBeeCoordinator.getLinkKey() and ZigBeeCoordinator.getMasterKey() methods*
v22-reg	*May, 22th*	Jean-Pierre Poutcheu, Orange Arnaud Rinquin, Orange *1. Explain that 'no response' command are used when handler is null in writeAttributes commands* *2. Explain that map use attribute ids as key and objects as values*
v23-reg	*May, 29th*	Jean-Pierre Poutcheu, Orange Arnaud Rinquin, Orange *1. Added a new section about ZigBeeAttributeRecord class* *2. Added getInvalidNumber() method in ZigBeeDataTypeDescription*
v24-reg	*June, 5th*	Jean-Pierre Poutcheu, Orange Arnaud Rinquin, Orange *1. Moved getSimpleDescriptor() from ZigBee Node section to Endpoint section* *2. Changed getInputCluster()/getOutputCluster() by getServerCluster()/getClientCluster()*

Revision	Date	Comments
v25-reg	*June, 12th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Renamed ZigBeeDeviceNode interface by ZigBeeNode* 2. *Updated 'Operation Summary' section to take into account the registration of ZigBeeNode as an OSGi service by the base driver.* 3. *In ZigBeeEndpoint, getDeviceNode() replaced by getNodeAddress(), which returns the node IEEE Address* 4. *In ZigBeeNode interface, static field ID replaced by IEEE_ADDRESS* 5. *Updated figure 6.2: 'Device Node' → 'Node'*
v26-reg	*June, 19th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Updated figure 6.1: 'ZigBeeDeviceNode' → 'ZigBeeNode'* 2. *Moved refreshNetwork(ZigBeeHandler) from ZigBeeCoordinator to ZigBeeHost* 3. *Added start() in ZigBeeHost*
v27-reg	*June, 28th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Deleted all the mention about ZigBeeCoordinator* 2. *Updated figure 6.1: Removed ZigBeeCoordinator interface* 3. *More explanation in ZigBee Networking section about the role of ZigBeeHost*
v28-reg	*July, 3rd*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange *ZigBeeHost.setOperationalMode(short) replaced by ZigBeeHost.setLogicalType(short)*
v29-reg	*July, 17th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Deleted ZigBeeAttributesHandler interface* 2. *IEEE Address managed as a Long Java type*
v30-reg	*July, 25th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Added properties ZigBeeNode.HOST_PID and ZigBeeEndpoint.HOST_PID_TARGET* 2. *Updated endpoint export section to take into account HOST_PID_PARGET property when exporting an endpoint.*

Revision	Date	Comments
v31-reg	*August, 29th*	Jean-Pierre Poutcheu, Orange André Bottaro, Orange 1. *Added ZigBeeGlobalClusterDescription.getClusterFunctionalDomain ()* 2. *Added a table in ZigBeeHandler section describing Map parameter response for onSuccess(Map) and onFailure(Map)*
v32-reg	*September, 3th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *Fix typo, spelling, and grammar.* 2. *Remove several methods duplicated from Javadoc.* 3. *Enhance some sentences.*
v33-reg	*September, 9th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *Enhance ZigBee Handler part.* 2. *Fix references.* 3. *Merge import/export HOST_PID.* 4. *Add some open questions as comments.*
v34-reg	*September, 17th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *Delete invoke(Object[] values, ZigBeeDataTypeDescription[] inputTypes, ZigBeeDataTypeDescription[] outputTypes, ZigBeeHandler handler)* 2. *Add serialize, and deserialize methods to ZigBeeCommandDescription. These methods are designed to ease the use of ZigBeeCommand.invoke(byte[] bytes, ZigBeeHandler handler).* 3. *Add ZigBeeHost.stop() method.*

Revision	Date	Comments
v35-reg	*September, 30th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *ZigBeeEvent: change Dictionary getAttributesEvents() to Object getValue()* 2. *Remove getDescription() from ZigBeeCluster, ZigBeeCommand, and ZigBeeAttributes.* 3. *In ZigBeeCluster, remove: public void readAttributesAsBytes(int[] attributesIds, ZigBeeHandler handler); and public void writeAttributes(boolean undivided, int[] attributesIds, byte[] values, ZigBeeHandler handler) throws ZigBeeNoDescriptionAvailableException;* 4. *Introduce ZigBeeCommandHandler* 5. *Rename ZigBeeException.ATTRIBUTE_NOT_SUPPORTED to UNSUPPORTED_ATTRIBUTE* 6. *Add constructor in ZigBeeAttributeRecord, and remove setters.* 7. *Remove setters/getters methods with bytes parameters in ZigBeeAttribute.* 8. *Introduce ZigBeeAttributesHandler.* 9. *ZigBee*Handler.onFailure now takes a ZigBeeException as parameter.* 10. *Remove getDeviceDescription() from ZigBeeEndpoint.*
v36-reg	*October, 2th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *ZigBeeEventListener: remove filter.* 2. *Event API: Specify mandatory, and optional pseudo properties for event filtering. Add ZigBeeEndpoint.ENDPOINT.* 3. *Add ZCL document version* 4. *ZigBeeEventListener: Add public void onFailure(ZigBeeException e);* 5. *ZigBeeEndpoint: move ENDPOINT – (zigbee.device.endpoint) to ZigBeeEndpoint.ID – (zigbee.endpoint.id); move zigbee.device.clusters.input to zigbee.endpoint.clusters.input; move zigbee.device.clusters.output to zigbee.endpoint.clusters.output.* 6. *ZigeeCluster: move zigbee.listener.cluster.* to zigbee.cluster.** 7. *ZigBeeNode: move zigbee.listener.node.ieee.address to zigbee.node.ieee.address* 8. *ZigBeeAttribute: move zigbee.listener.attribute.* to zigbee.attribute.**

Revision	Date	Comments
v37-reg	*October, 7th* *October, 10th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *Remove no longer needed ZigBeeNoDescriptionAvailableException* 2. *In the ZigBeeHandler section, Map took int as key, and now take Integer (This is a Java requirement).* 3. *Update ZCL document version (from 075123r01ZB to 075123r04ZB)* 4. *Add isPartOfAScene() method to ZigBeeAttributeDescription* 5. *Add a short explanation regarding ZigBeeEvent.*
v38-reg	*October, 18th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *ZigBeeAttributeRecord is now a final java class.* 2. *Remove PAN_ID and EXTENDED_PAN_ID from ZigBeeEventListener interface; the ones from ZigBeeEndpoint interface must be used.* 3. *Move listener static fields that are listener properties into ZigBeeEventListener: (REPORTABLE_CHANGE, MIN/MAX_INTERVAL, TIME_OUT). Keep the zigbee.attribute. prefix.* 4. *Event API: Specify mandatory, and optional pseudo-properties for event filtering. Add ATTRIBUTE_DATA_TYPE = "zigbee.attribute.datatype"*
v39-reg	*November, 4th*	Antonin Chazalet, Orange André Bottaro, Orange 1. *Update some comments on ZigBeeEventListener's optional properties.* 2. *ZigBeeEndPoint: Add additional properties.*

1 Introduction

ZigBee [1]. is a standard wireless communication protocol designed for low-cost and low-power devices by ZigBee Alliance. ZigBee 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 ZigBee devices.

This specification defines how OSGi bundles can be developed to discover and control ZigBee devices on the one hand, and act as ZigBee devices and interoperate with ZigBee clients on the other hand. In particular, a Java mapping is provided for the standard hierarchical representation of ZigBee devices called ZigBee Cluster Library Erreur : source de la référence non trouvée. The specification also describes the external API of a ZigBee Base Driver according to Device Access specification, the example made by UPnP Device Service specification and spread OSGi practices on residential market [3].[4]..

2 Application Domain

System Architecture

When installing a new ZigBee network into a residential network with a home gateway, there are 2 options. One is to add ZigBee communication capability to your home gateway with an additional hardware such as a USB device called "dongle". The other one is to replace the current home gateway with one which has ZigBee communication capability. In both cases OSGi applications call the ZigBee driver API to communicate with the ZigBee network (and its ZigBee devices) as shown in 1.

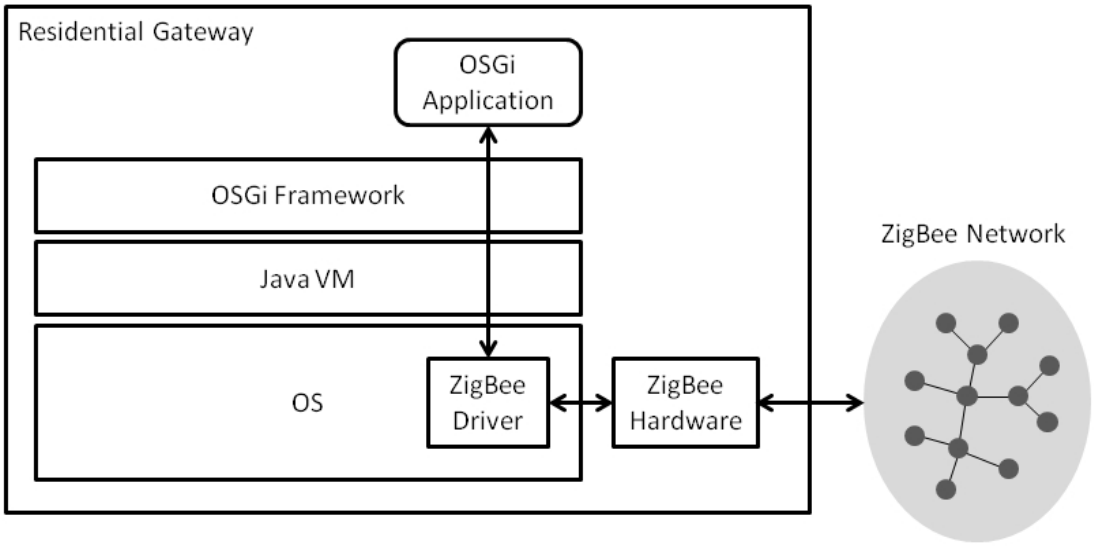


Figure 1: Communication with ZigBee devices through a ZigBee driver

The ZigBee specification defines three types of ZigBee devices: ZigBee Coordinator (ZC), ZigBee Router (ZR) and ZigBee End Device (ZED). In the above case the ZigBee hardware works as the ZigBee Coordinator and the other ZigBee devices are attached to the ZigBee network as ZigBee End Device or ZigBee Router.

– ZigBee Coordinator (ZC) is responsible for managing a ZigBee network and ZigBee devices on the network. There is one, and only one ZigBee Coordinator is in each ZigBee network.

– ZigBee Router (ZR) is capable of extending a ZigBee network by relaying messages from other ZigBee devices.

– ZigBee End Device (ZED) has functionality to communicate with either ZigBee Coordinator or ZigBee Router.

ZigBee Stack

The ZigBee stack is shown in 2. The two bottom layers, the PHY layer and the MAC layer, are defined by IEEE802.15.4 standard. The ZigBee standard defines network (NWK) layer, application (APL) layer and security layer on top of it. The NWK layer is responsible for managing the network formation and routing. The APL layer hosts application objects developed by manufacturers. The security service provider is responsible for encryption and authentication.

The application layer consists of three functional blocks: application support sub-layer, ZigBee Device Object (ZDO) and application framework. The application support sub-layer provides the transmission capability of data and management messages. The ZDO provides common functionality used by all applications. The application framework is the environment where application objects are hosted to control and manage the protocol layers.

There are two interfaces available to applications: APSDE-SAP and ZDO public interface. The APSDE-SAP provides data transmission functionality between ZigBee devices. The ZDO public interface provides applications with management functionality such as device discovery, service discovery and network management.

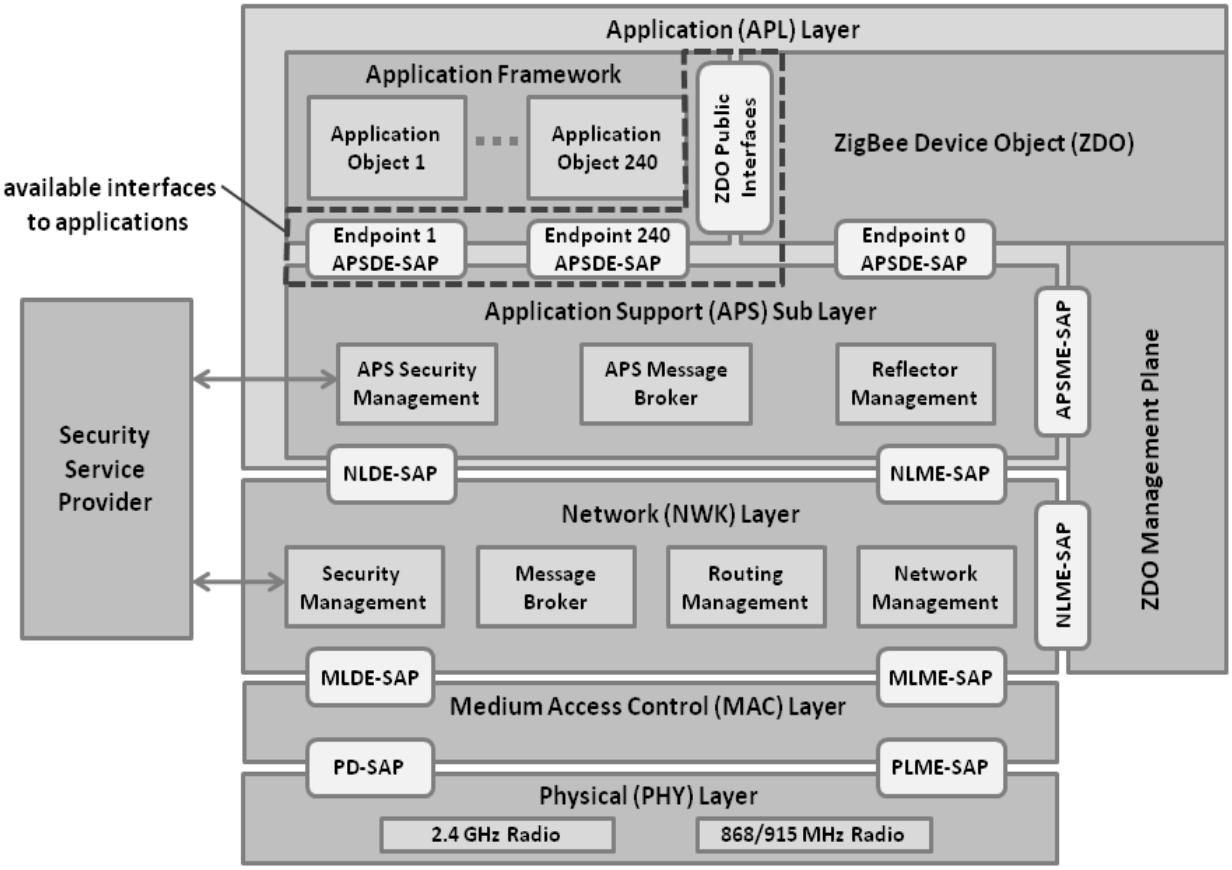


Figure 2: ZigBee Stack

Application Profiles and ZigBee Cluster Library (ZCL)

The application profiles allow interoperability between products developed by different vendors for a specific application. For example, in a light control scenario, switches developed by a vendor can turn on and turn off lights developed by another vendor if the both vendors take the same application profile. The ZigBee Alliance has defined nine public application profiles such as Home Automation (HA) and ZigBee Smart Energy (ZSE).

An application profile defines its application domain, a list of specific devices supported in the profile and a list of clusters supported by the devices. A cluster is a relevant collection of commands and attributes which together define an interface for a specific functionality. The clusters used in public application profiles are defined in the ZigBee Cluster Library (ZCL) specification. The ZCL specification defines a number of clusters and categories them into groups by their functionality.

3 Problem Description

As described in the section Erreur : source de la référence non trouvée, OSGi applications which communicate with ZigBee devices are supposed to call the API of the driver provided by the vendor. The API is proprietary and different vendor by vendor since it is not standardized in the ZigBee specification. This causes the following problems:

1) Application developers need to know which vendor's ZigBee hardware is used with the target residential gateway in advance before developing their applications.
2) An application which was developed for a certain environment may not work for other environments.

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

The standard ZigBee API demanded in this RFP would give developers a unified way of communicating with ZigBee devices. The developers will no longer need to care about the proprietary API of drivers but will simply use the standard one.

4 Requirements

R1: The solution MUST provide an API for data transmission supported by APSDE-SAP.

R2: The solution MUST provide a base driver interface as an OSGi service for management operations supported by ZDO: device and service discovery, security management, network management, binding management, node management and group management.

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

R4: The solution MUST provide API for switching the type of the local ZigBee device among ZC, ZR and ZED.

R5: The solution MUST provide a mechanism which notifies OSGi applications of events occurred in the ZigBee network and devices.

R6: The solution MUST provide an installation capability of cluster libraries within OSGi service-oriented architecture.

R7: The solution MUST register a Device Service object representing each found ZigBee device into Service Registry and unregister the Device Service object when the ZigBee device is unavailable.

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 interface used to control the ZigBee network. The solution MUST likewise work with network controllers based on ZigBee built-in chips, ZigBee USB dongles and high level protocols offered by ZigBee Gateway Devices compliant with the ZigBee Alliance specification.

5 Technical Solution

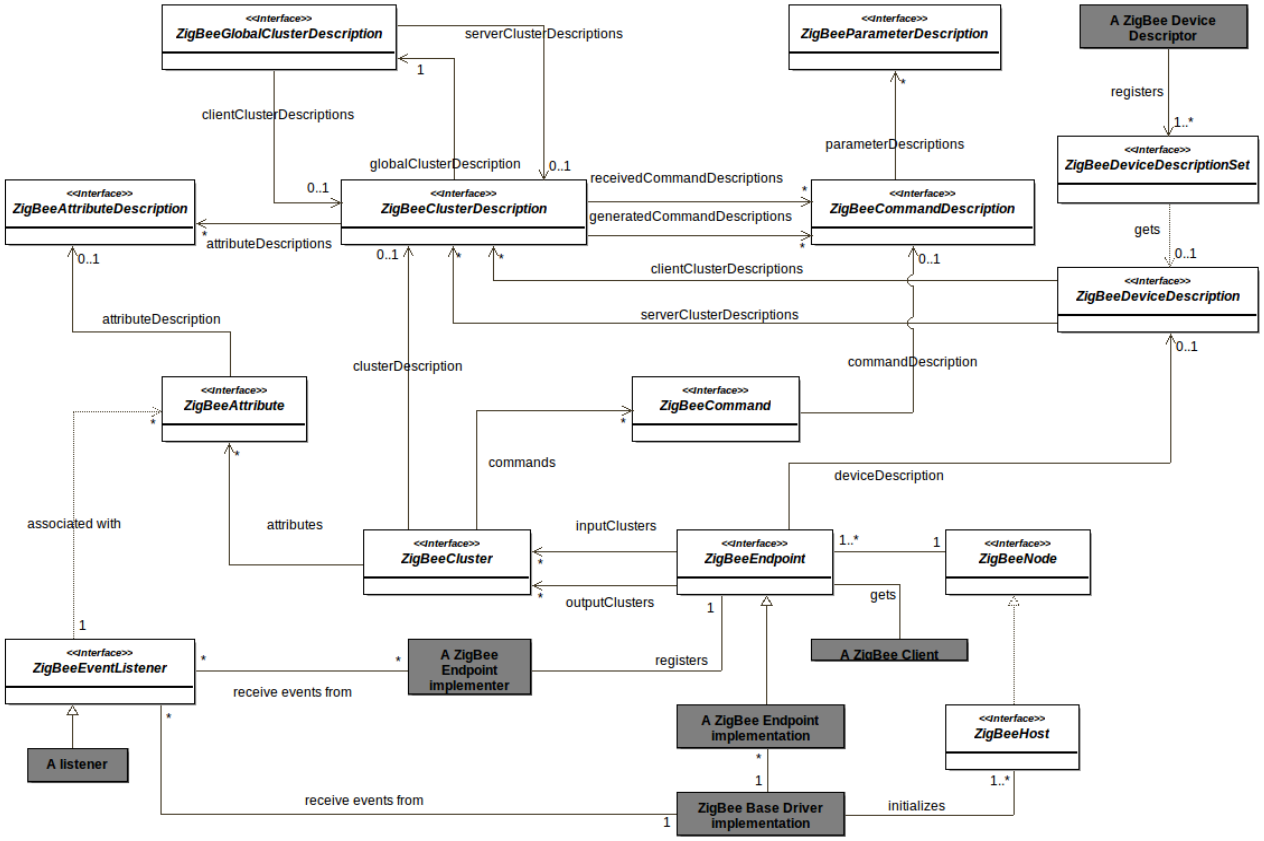
Essentials

- *Scope* – This specification is limited to general device discovery and control aspects of the ZigBee specifications. Aspects concerning the representation of specific ZigBee profiles is not addressed.
- *Transparency* – ZigBee 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 ZigBee device services on the OSGi platform is optional. Some base driver implementations may implement all the classes including ZigBee device description classes while implementations targeting constrained devices are able to implement only the part that is necessary for ZigBee device discovery and control.
- *Network Selection* – It must be possible to restrict the use of the ZigBee protocols to a selection of the connected networks.
- *Logical node type selection* – It is possible to make an OSGi-based device appearing as a ZigBee end device, a ZigBee router or a ZigBee coordinator.
- *Event handling* – Bundles are able to listen to ZigBee events.
- *Discover and Control ZigBee Endpoints as OSGi services* – Available ZigBee endpoints are dynamically reified as OSGi services in the service registry.
- *Export OSGi services as ZigBee Endpoints* – OSGi services implementing the API defined here and explicitly set to be *exported* should be made available to networks with ZigBee enabled endpoints in a transparent way.

Entities

- *ZigBee Base Driver* – The bundle that implements the bridge between OSGi and ZigBee networks.
- *ZigBee Node* – A physical ZigBee node. This entity is represented by a **ZigBeeNode** object. It is registered as an OSGi service by the Base Driver.
- *ZigBee Endpoint* – A logical device that defines a communication entity within a ZigBee node through which a specific application profile is carried. This concept is represented by a **ZigBeeEndpoint** object. Registered as an OSGi service, an endpoint can be local (implemented by the Framework) or external (implemented by another device on the network).
- *ZigBee Device Description* – Statically describes a ZigBee endpoint by providing its input/output clusters and specifies which of these clusters are mandatory or not. This entity is represented by a **ZigBeeDeviceDescription** object.
- *ZigBee Cluster* – Represents a cluster existing in the network. It holds its cluster description, the current value for each attributes and allows command invocation. This concept is represented by a **ZigBeeCluster** object.
- *ZigBee Cluster Description* – Cluster description provides details about available commands and attributes for a specific Cluster. A cluster description should be constant. A cluster description hold either Client or Server Cluster description and refers to a global cluster description.
- *ZigBee Cluster Description Set* – Cluster description Set provides to ZigBeeEndpoint a service for retrieving cluster descriptions. This concept is represented by a **ZigBeeClusterDescriptionSet** object and registered as an OSGi service.

- *ZigBee Global Cluster Description* – Global cluster description holds the server and client cluster description as well as common information such as cluster id, description and name. This concept is represented by a **ZigBeeGlobalClusterDescription** object. A global cluster description may be exposed as a service so the Base Driver can automatically associate it to newly or existing Clusters.
- *ZigBee Command* – It's a ready-to-invoke command related to an existing ZigBee cluster This entity is represented by a **ZigBeeCommand** object.
- *ZigBee Command Description* – Statically describes a specific cluster command by giving its name, id, parameters. This entity is represented by a **ZigBeeCommandDescription** object.
-
- *ZigBee Parameter Description* – A ZigBee parameter description has a name, a range and a data type. This entity description is represented by a **ZigBeeParameterDescription** object.
- *ZigBee Attribute* – Holds the current value of an existing cluster attribute, it allows easy (de)encoding. This concept is represented by a **ZigBeeAttribute** object.
- *ZigBee Attribute Description* – Statically describes a ZigBee Attributes (data type, name, default value). It does not hold any current value. This concept is represented by a **ZigBeeAttributeDescription** object.
- *ZigBee Event Listener Service* – A service that listens to events coming from ZigBee devices.
- *ZigBee Event* – An event generated by a ZigBee node. It contains a modified attribute value of a specific cluster. This concept is represented by a **ZigBeeEvent** object.
- *ZigBee Handler* – A ZigBee handler is a helper that manages asynchronous communication with the base driver. This entity is represented, e.g. by **ZigBeeHandler**.
- *ZigBee Host* – The machine that hosts the code to run a ZigBee device or client. It contains information related to the Host. If the host is in the coordinator logical node type, it enables networking configuration. It is registered as an OSGi service. This concept is represented by **ZigBeeHost**.
- *ZigBee Client* – An application that is intended to control ZigBee devices services.
- *ZigBee Exception* – An exception that delivers errors that occurred in the ZigBee stack.



– *Figure 3: ZigBee Service Specification class Diagram org.osgi.service.zigbee package*

Operation Summary

To make a ZigBee device available (as an OSGi service) to ZigBee clients on a network, an OSGi service object must be registered under the **ZigBeeNode** interface with the OSGi Framework. The ZigBee base driver must detect these ZigBee Node services and must make them available to the network as ZigBee nodes using the ZigBee protocol on the ZigBee host (e.g., a ZigBee chip) associated to the base driver.

ZigBee nodes available on the local network must be detected and automatically registered under the **ZigBeeNode** interface. Available ZigBee endpoints must also be registered as services under **ZigBeeEndpoint** interface with the Framework by the ZigBee driver implementation bundle. Bundles should not be able to distinguish resident or remote ZigBee Endpoint services.

A bundle that wants to control ZigBee devices, for example to implement a ZigBee client, should track ZigBee Endpoint services in the OSGi service registry and control them appropriately.

ZigBee Base Driver

The functionality described in the operation summary is implemented in a ZigBee *base driver*. A ZigBee base driver is a bundle that implements the ZigBee protocols and handles the interaction with bundles that use the ZigBee devices. It must discover ZigBee devices on the ZigBee network and map each discovered device into OSGi registered ZigBee Node services. It must also expose, on the ZigBee Network, ZigBee Node services (programmatically registered as OSGi services).

ZigBee Node

A ZigBee node represents a physical ZigBee device and should adhere to a specific application profile that can be either public or private. Profiles define the environment of the application, the type of devices and the clusters used for them to communicate.

A physical device is reified and registered as a **ZigBeeNode** service in the Framework. A ZigBee node holds several ZigBee endpoints that are registered as **ZigBeeEndpoint** objects. **ZigBeeNode** provides getEndPoints() method which returns its associated endpoints.

ZigBee nodes properties are defined in the ZigBee Specification. These properties must be registered in the OSGi Framework services registry so they are searchable. **ZigBeeNode** must be registered with the following properties:

- NODE_TYPE – (`zigbee.node.description.type`) specifies the device type of the ZigBee node. The ZigBee specification defines three types of nodes: ZigBee coordinator, ZigBee router and ZigBee end device.
- MANUFACTURER_CODE – (`zigbee.node.description.manufacturer.code`) specifies a manufacturer code that is allocated by the ZigBee Alliance, relating to the device manufacturer.
- MANUFACTURER_NAME – (`zigbee.node.description.manufacturer.name`) The manufacturer name contains a string representing the name of the manufacturer of the device. This property is optional.
- MODEL_NAME – (`zigbee.node.description.model.name`) contains a string representing the name of the manufacturers model of the device. This property is optional.
- PAN_ID – (`zigbee.node.pan.id`) (Personal Area Network Identifier) is a 16-bit value that identifies a ZigBee network. Every **ZigBeeNode** object is associated to a PAN ID, which can be retrieved through the ZigBeeNode.getPanId() method.
- EXTENDED_PAN_ID – (`zigbee.node.pan.extended.id`) Extended PAN ID is a 64-bit numbers that uniquely identify a PAN. It is intended to enhance selection of a PAN and enable recognition of network after PAN ID change (due to previous conflict). ZigBeeNode.getExtendedPanId() returns the network extended PAN ID if specified.

  PAN_ID and EXTENDED_PAN_ID are optional, but at least one of these properties MUST be specified.

ZigBee nodes describes themselves using descriptor data structures. There are five descriptors (i.e. four are related to the Node level, and one is related to the Endpoint level):

- getNodeDescriptor() – Returns a **ZigBeeNodeDescriptor** object representing Node Descriptor which contains information about the node capabilities.
- getPowerDescriptor() – Returns a **ZigBeePowerDescriptor** object representing Node Power Descriptor which gives a dynamic indication of the node power status.
- getComplexDescriptor() – Returns a **ZigBeeComplexDescriptor** object representing Complex Descriptor which contains extended information for each device descriptions contained in this node. Returns null if no Complex Descriptor is provided.
- getUserDescriptor() – Returns a **ZigBeeUserDescriptor** object representing User Descriptor which contains information that allows the user to identify the device using user-friendly character string. Returns null if no User Descriptor is provided.

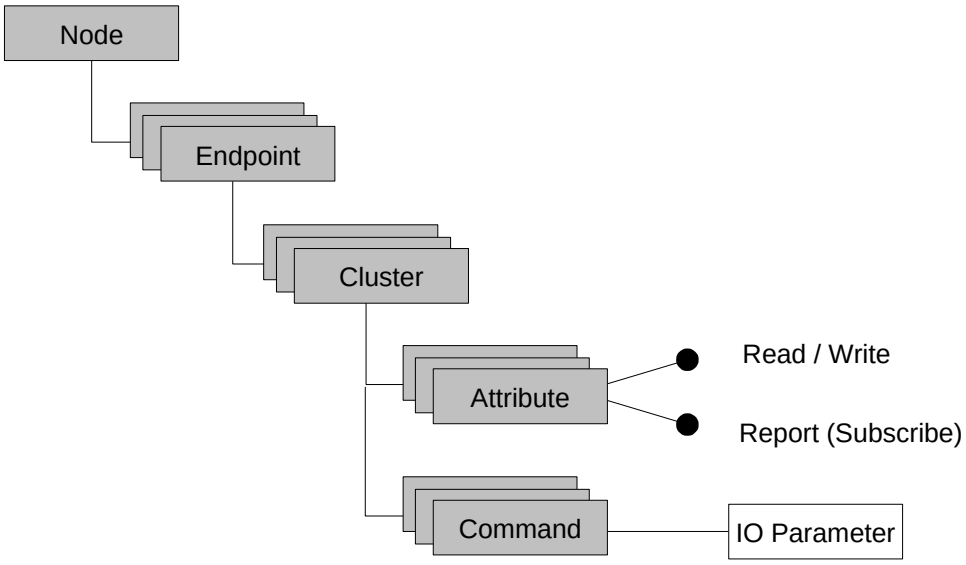


Figure 4: ZigBee hierarchy model

All interfaces corresponding to the hierarchy model must be implemented in order to discover and control asynchronously (events) ZigBee devices. Classes related to the description of these entities (named with suffix "*Description" can optionally be implemented. This rule follows the fact that ZigBee device descriptions are not downloadable on the device itself and are often given to developers in an out-of-band manner.

ZigBee Endpoint

Communication between devices is done through an addressable component called ZigBee endpoint which holds a number of ZigBee clusters. A ZigBee cluster represents a functional unit in a device.

An endpoint defines a communication entity within a device through which a specific application is carried. So, it represents a logical device object used for communication.

For example, a remote control light might allocate Endpoint 7 for the control of lights in the master bedroom, Endpoint 9 to manage the heating and air conditioning system, and Endpoint 14 for controlling the security system.

The ZigBee specification defines that a maximum of 240 Endpoints is allowed per **ZigBeeNode**. Endpoint 0 is reserved to describe the generic device capabilities, Endpoint 255 is reserved for broadcasting to all endpoints, Endpoints 241-254 are reserved for future use.

Endpoint 0 and Endpoint 255 capabilities are not exposed, only Endpoints 1-240 should be registered as services. Endpoints are registered under the **ZigBeeEndpoint** interface with the following properties:

- ID – (`zigbee.endpoint.id/String`) specifies the endpoint address within the node. Applications shall only use endpoints 1-240.
- PROFILE_ID – (`zigbee.device.profile.id/String`) identifies the profile that is supported by this endpoint. The ZigBee specification defines several profile identifiers, and some others are vendor specific.
- DEVICE_ID – (`zigbee.device.id/String`) identifies the device description supported by this endpoint. Like the profiles identifiers, the ZigBee specification defines several device identifiers, and some others are vendor specific.
- DEVICE_VERSION – (`zigbee.device.version/String`) specifies the device description version

supported by this endpoint.

- INPUT_CLUSTERS – (`zigbee.endpoint.clusters.input/String[]`) specifies the list of input cluster ids supported by this endpoint.
- OUTPUT_CLUSTERS – (`zigbee.endpoint.clusters.output/String[]`) specifies the list of output cluster ids supported by this endpoint.
- org.osgi.service.device.Constants.DEVICE_CATEGORY (see OSGI Compendium: 103 Device Access Specification) – ("DEVICE_CATEGORY") defines the ZigBee device category. Its value is "ZigBee" (org.osgi.service.zigbee.ZigBeeEndpoint.DEVICE_CATEGORY).

Additional properties (defined in Device Access – 103.2.1) may be set: DEVICE_CATEGORY, DEVICE_DESCRIPTION, DEVICE_SERIAL, and service.pid.

A **ZigBeeEndpoint** may contain a number of input or output clusters. **ZigBeeEndpoint** provides getServerCluster(int clusterId) and getClientCluster(int clusterId) to return a specific server input or client output cluster.

Each endpoint must provide a simple descriptor. getSimpleDescriptor() returns a **ZigBeeSimpleDescriptor** object which contains general information about the endpoint.

ZigBee Device Description

A ZigBee endpoint may have a description used to define and instantiate his input and output clusters, but also describes which of these clusters are mandatory or not.

ZigBeeDeviceDescription provides general information about an endpoint, and describes its inputs and outputs clusters.

ZigBee Device Description Set

ZigBeeDeviceDescriptionSet objects are registered as OSGi services by the Base Driver. A ZigBeeDeviceDescriptionSet provides getDeviceSpecification(int deviceId, short version) which returns the device description if provided, or null otherwise. ZigBeeDeviceDescriptionSet service should be registered with the following properties:

- VERSION – (`zigbee.profile.version`) The application profile version.
- PROFILE_ID – (`zigbee.profile.id`) The profile identifier.
- PROFILE_NAME – (`zigbee.profile.name`) The profile name.
- MANUFACTURER_CODE – (`zigbee.profile.manufacturer.code`) The manufacturer profile code.
- MANUFACTURER_NAME – (`zigbee.profile.manufacturer.name`) The manufacturer profile name.
- DEVICES – (`zigbee.profile.devices`) A comma separated list of devices identifiers supported by the profile.

ZigBee Cluster

Devices communicate with each other by means of clusters, which may be inputs to or outputs from the device. For example, in the ZigBee HA (Home Automation) profile there is a cluster dedicated to the control of lighting subsystems. Clusters are represented under **ZigBeeCluster** interface.

ZigBeeCluster objects combine one or more ZigBeeCommand and ZigBeeAttribute.

ZigBeeCluster provides some methods for reading and writing attributes values:

- readAttributes(int[] attributeIds, ZigBeeAttributesHandler handler) – The read attributes command is generated when a device wishes to determine the value of one or more attributes located on another device.
- writeAttributes(boolean undivided, ZigBeeAttributeRecord[] attributes, ZigBeeAttributesHandler handler) – The write attributes command is generated when a device wishes to change the values of one or more attributes located on another device. If handler is set to null, the base driver should use a 'no response' ZigBee general command (see Chapter 2.4 General Commands in ZCL specification). The boolean undivided parameter specifies that if any attribute cannot be written (e.g. If an attribute is not implemented on the device, or a value to be written is outside the valid range), no attribute values are changed.

ZigBee Cluster Description

ZigBeeClusterDescription describes the server or client part of a **ZigBeeCluster**. It lists the available commands and attributes for this client or server cluster.

Each cluster client and server may have attributes (see ZCL specifications chapter 2.2.1), received and generated commands. ZigBeeClusterDescription provides methods to describe commands, attributes and retrieve general cluster information.

ZigBee Global Cluster Description

ZigBeeGlobalClusterDescription describes a cluster general information: id, name, description. It provides the **ZigBeeClusterDescription** for both client and server part of this cluster.

ZigBee Command

A ZigBee cluster is associated with a number of commands. There are two types of commands: generated commands and received commands. Theses commands could be sent either by a client or server cluster. ZigBee Commands are used to manipulate ZigBee Attributes values and ZigBee Device state in general.

Every command is represented by a **ZigBeeCommand** interface.

ZigBeeCommandDescription contains **ZigBeeParameterDescription** objects which describe the command parameters. **ZigBeeParameterDescription** has, for instance, a checkValue(Object value) method which returns true if the parameter value is valid according to its description.

ZigBeeCommand provides one operation for its invocation using a handler based response; operations parameters are attributes that will be manipulated:

- invoke(byte[] bytes, ZigBeeCommandHandler handler) – bytes is a sequence of byte and represents the command frame.

A handler is provided to manage the command response asynchronously.

ZigBee Command Description

ZigBeeCommandDescription describes a ZigBeeCommand object.

All clusters (server and client) shall support generation, reception and execution of the Default response command.

Each cluster (server or client) that implements attributes shall support reception of, execution of, and response to all commands to discover, read, write, report, configure reporting of, and read reporting configuration of these attributes. Generation of these commands is application dependent.

ZigBeeCommandDescription also provides two methods for serializing (Java Values to bytes), and deserializing (bytes to Java Values). These bytes are, respectively, the parameters, and the returned value of the **ZigBeeCommand**.invoke method.

ZigBee Attribute

A ZigBee cluster is associated with a set of attributes. Each attribute is represented by a **ZigBeeAttribute** interface. ZigBeeAttribute provides getValue(ZigBeeAttributesHandler handler) and setValue(Object value, ZigBeeAttributesHandler handler) to retrieve and set current attribute value.

ZigBee Attribute Description

ZigBeeAttributeDescription describes information about a specific **ZigBeeAttribute.**

ZigBee Data Type Description

ZigBeeAttributeDescription and **ZigBeeParameterDescription** provides a getDataType() which returns a **ZigBeeDataTypeDescription** object. The interface provides, amongst others, the following methods:

- serialize(Object param) – Serializes a Java Object corresponding to the Java data type given by getJavaDataType() into a byte array according to ZigBee Cluster Library.
- deserialize(byte[] data) – Deserializes a byte array into a Java Object of the Java data type given by getJavaDataType().

ZigBee Attribute Record

A **ZigBeeAttributeRecord** object holds some useful information about a given **ZigBee attribute**. It's mainly used by ZigBeeCluster.writeAttributes(...) for writing operations. **ZigBeeAttributeRecord** provides the following methods:

- getId() – Returns the attribute identifier.
- getDataType() – Returns the attribute data type.
- getValue() – Returns the attribute object value.

ZigBee Handler

The ZigBee Handlers (i.e. **ZigBeeHandler**, **ZigBeeCommandHandler**, and **ZigBeeAttributesHandler**) help to manage asynchronous communication with the base driver. The defined interfaces are used when requesting the base driver and provide onSuccess(...) and onFailure(...) methods for managing responses.

This table shows the methods that uses a **Handler** and the following description of the map parameter:

Methods	Map parameter description
ZigBeeCommand.invoke(byte[] bytes, ZigBeeCommandHandler handler)	The onSuccess method takes the response of **byte[]** type as parameter. In case of a failure, onFailure takes a ZigBeeException.
ZigBeeCluster.readAttributes(int[] attritesIds, ZigBeeAttributesHandler handler)	For each Map entry, the key is the attribute identifier of **Integer** type and the value is the associated attribute value of **Object** type (or null if an UNSUPPORTED_ATTRIBUTE occurred).
ZigBeeCluster.writeAttributes(boolean undivided, ZigBeeAttributeRecord[] attributes, ZigBeeAttributesHandler handler)	For each Map entry, the key is the attribute identifier of **Integer** type and the value is the associated attribute status, i.e. SUCCESS, INVALID_VALUE, etc. In case **undivided** equals **false**, **onSuccess()** is always called to notify the response. In case **undivided** equals **true** and an error has occurred, **onFailure** is

	called with a ZigBeeException.
ZigBeeAttribute.getValue(ZigBeeAttributesHandler handler)	Only one Map entry, the key is the attribute identifier of **Integer** type and the value is the associated attribute value of **byte[]** type. In case of a failure, **onFailure** is called with a ZigBeeException.
ZigBeeAttribute.setValue(Object value, ZigBeeAttributesHandler handler)	Only one Map entry, the key is the attribute identifier of **Integer** type and the value is **true** if the attribute value has been written or **false** otherwise.

ZigBee Data Types

ZigBeeDataTypes provides all standard ZigBee type descriptions as **ZigBeeDataTypeDescription** objects assigned to public final static fields (constants).

Here is the table of encoding relations between ZigBee types and Java types:

ZigBeeDataType constant	ZigBee type	Java type
BITMAP_8 BITMAP_16 BITMAP_24 BITMAP_32 GENERAL_DATA_8 GENERAL_DATA_16 GENERAL_DATA_24 GENERAL_DATA_32	8-bit bitmap 16-bit bitmap 24-bit bitmap 32-bit bitmap 8-bit data 16-bit data 24-bit data 32-bit data	Integer
BITMAP_40 BITMAP_48 BITMAP_56 BITMAP_64 GENERAL_DATA_40 GENERAL_DATA_48 GENERAL_DATA_56 GENERAL_DATA_64	40-bit bitmap 48-bit bitmap 56-bit bitmap 64-bit bitmap 40-bit data 48-bit data 56-bit data 64-bit data	Long
OCTET_STRING LONG_OCTET_STRING	Octet string Long octet string	ByteBuffer
CHARACTER_STRING LONG_CHARACTER_STRING SECURITY_KEY	Character string Long character string 128-bit Security Key	String
BOOLEAN	Logical	Boolean
SIGNED_INTEGER_8	Signed 8-bit integer	Byte
UNSIGNED_INTEGER_8 SIGNED_INTEGER_16	Unsigned 8-bit integer Signed 16-bit integer	Short
UNSIGNED_INTEGER_16 UNSIGNED_INTEGER_24	Unsigned 16-bit integer Unsigned 24-bit integer	

SIGNED_INTEGER_24 SIGNED_INTEGER_32 ENUMERATION_8 ENUMERATION_16 CLUSTER_ID ATTRIBUTE_ID	Signed 24-bit integer Signed 32-bit integer 8-bit enumeration 16-bit enumeration Unsigned 16-bit integer Unsigned 16-bit integer	Integer
UNSIGNED_INTEGER_32 UNSIGNED_INTEGER_40 UNSIGNED_INTEGER_48 UNSIGNED_INTEGER_56 UNSIGNED_INTEGER_64 SIGNED_INTEGER_40 SIGNED_INTEGER_48 SIGNED_INTEGER_56 SIGNED_INTEGER_64 UTC_TIME BACNET_OID IEEE_ADDRESS	Unsigned 32-bit integer Unsigned 40-bit integer Unsigned 48-bit integer Unsigned 56-bit integer Unsigned 64-bit integer Signed 40-bit integer Signed 48-bit integer Signed 56-bit integer Signed 64-bit integer UTCTime BACnet OID*(Unsigned 32-bit integer) IEEE address (MAC-48,EUI-48/64)	Long
TIME_OF_DAY DATE	Time of day Date	Date
FLOATING_SEMI FLOATING_SINGLE	Semi-precision Single precision	Float
FLOATING_DOUBLE	Double precision	Double
ARRAY STRUCTURE	Array Structure	[ ] List
SET BAG	Set Bag	Set
UNKNOWN	Unknown	

* BACnet OID (Object identifier) data type is included to allow interworking with BACnet (see [5].). The format is described in the referenced standard.

Working With a ZigBee Endpoint

All discovered ZigBee endpoints in the local networks are registered under **ZigBeeEndpoint** interface with the OSGi Framework. Every time a ZigBee endpoint appears or quits the network, the associated OSGi service is registered or unregistered in the OSGi service registry. Thanks to the ZigBee Base Driver, the OSGi service availability in the registry mirrors ZigBee device availability on ZigBee networks Erreur : source de la référence non trouvée.

Using a remote ZigBee endpoint thus involves tracking ZigBee Endpoint services in the OSGi service registry. The following code illustrates how this can be done. The sample Controller class extends the ServiceTracker class so that it can track all ZigBee Endpoint services and add them to a user interface, such as a remote controller application.

```
class Controller extends ServiceTracker {
   UI ui;
   Controller( BundleContext context ) {
   super( context, ZigBeeEndpoint.class, null );
   }
```

```
public Object addingService( ServiceReference ref ) {
       ZigBeeEndpoint endpoint = (ZigBeeEndpoint)super.addingService(ref);
       ui.addEndpoint( endpoint );
       return endpoint;
}
public void removedService( ServiceReference ref, Object endpoint ) {
       ui.removeEndpoint( (ZigBeeEndpoint) endpoint );
}
...
  }
```

Implementing a ZigBee Endpoint

OSGi services can also be exported as ZigBee endpoints to the local networks, in a way that is transparent to typical ZigBee devices endpoints. This allows developers to bridge legacy devices to ZigBee networks. A bundle should register a **ZigBeeEndpoint** with the following properties to export an OSGi service as a ZigBee endpoint:

- ZIGBEE_EXPORT – To indicate that the endpoint is an exportable endpoint.
- HOST_PID (`zigbee.device.target.host.pid`) – The ZigBee local host identifier is intended to uniquely identify the ZigBee local host, since there could be many hosts on the same platform. This property is not mandatory. All the nodes that belong to a specific network MUST specify the value of the associated host number.

An OSGi platform can host multiple ZigBee networks. HOST_PID, PAN_ID and EXTENDED_PAN_ID are used to select the appropriate network. At least one of these properties MUST be specified. If provided, HOST_PID have priority on PAN_ID and EXTENDED_PAN_ID to identify the host that is targeted for export.

In addition, the **ZigBeeEndpoint** service MUST declare the same properties as an imported endpoint. The bundle registering endpoint services must make sure these properties are set accordingly or that none of these properties are set. In case a ZigBee Host is not initialized yet or the base driver is not active on the OSGi framework, an endpoint implementation could wait for a **ZigBeeHost** to appear in the OSGi service registry before setting these properties.

The Base Driver will export the endpoint on the ZigBee network associated to the ZigBee HOST PID, ZigBee PAN ID or Extended PAN ID. The associated **ZigBeeNode** object MUST be one of the available **ZigBeeHost** objects. Every time an Endpoint is registered or unregistered with both ZIGBEE_EXPORT and PAN IDs properties set, the associated ZigBeeHost service is modified accordingly (getEndPoints() returns a different Enumeration object).

If an error occurs when exporting a ZigBee endpoint, then the base driver calls ZigBeeEndpoint.notExported() method with a relevant **ZigBeeException** object as the input argument.

The reader must note that a same ZigBeeEndpoint object can not be registered several times with distinct PAN IDs since ZigBeeEndpoint.NodeAddress() method can only return one ZigBee Node address. If the situation happens then the base driver calls ZigBeeEndpoint.notExported() with the error code ZigBeeException.DUPLICATE_EXISTS.

If the PAN ID corresponds to more than one ZigBeeHost, the ZigBeeEndpoint MUST define the Extended PAN ID property which uniquely identifies a ZigBee network. The base driver will call ZigBeeEndpoint.notExported() with the error code ZigBeeException.DUPLICATE_EXISTS if the Extended PAN ID property is not properly defined in this specific situation.

Moreover, if the HOST PID corresponds to more than one ZigBeeHost, the base driver will also call ZigBeeEndpoint.notExported() with the error code ZigBeeException.DUPLICATE_EXISTS.

Event API

There are two distinct event directions for the ZigBee Service specification.

- External events from the network must be dispatched to listeners inside the OSGi Service Platform. The ZigBee Base driver is responsible for mapping the network events to internal listener events.
- Implementations of ZigBee endpoints must send out events to local listeners. The ZigBee Base driver dispatches events from its own listeners.

ZigBee events are sent using the whiteboard pattern [6]., in which a bundle interested in receiving the ZigBee events registers an object implementing the **ZigBeeEventListener** interface. The service MUST be registered with PAN_ID_TARGET (zigbee.device.target.pan.id) and/or EXTENDED_PAN_ID_TARGET (zigbee.device.target.extended.pan.id) properties. These properties indicates the network targeted by the listener since an OSGi platform can host multiple ZigBee networks.

A filter can be set to limit the events for which a bundle is notified. The ZigBee Base driver must register a ZigBee Event Listener without filter that receives all events.

The filter refers to any valid combination of the following pseudo properties for event filtering. The mandatory properties are:

- ZigBeeNode.IEEE_ADDRESS – (zigbee.node.ieee.address) Only events generated by endpoints matching the specific node are delivered.
- ZigBeeEndpoint.ID – (zigbee.endpoint.id) Only events matching a specific endpoint are delivered.
- ZigBeeCluster.ID – (zigbee.cluster.id) Only events generated by endpoints matching a specific cluster are delivered.
- ZigBeeAttribute.ID – (zigbee.attribute.id) Only events generated by endpoints matching a specific attribute are delivered.

The optional properties are:

- ZigBeeEventListener.ATTRIBUTE_DATA_TYPE – (zigbee.attribute.datatype) The Attribute data type field contains the data type of the attribute that is to be reported (see ZCL – 2.4.7.1.4 Attribute Data Type Field).
- ZigBeeEventListener.MIN_REPORT_INTERVAL – (zigbee.attribute.min.report.interval) The minimum interval, in seconds, between issuing reports of the specified attribute.
- ZigBeeEventListener.MAX_REPORT_INTERVAL – (zigbee.attribute.max.report.interval) The maximum interval, in seconds, between issuing reports of the specified attribute.
- ZigBeeEventListener.REPORTABLE_CHANGE – (zigbee.attribute.reportable.change) The minimum change to the attribute that will result in a report being issued.
- ZigBeeEventListener.TIMEOUT_PERIOD – (zigbee.attribute.timeout.period) The maximum expected time, in seconds, between received reports for the attribute (see ZCL – 2.4.7.1.8 section).

All the 4 following properties must be set into a configure reporting request: ATTRIBUTE_DATA_TYPE, MIN_REPORT_INTERVAL, MAX_REPORT_INTERVAL, and REPORTABLE_CHANGE.

If TIMEOUT_PERIOD (see [2]. 2.4.7 Configure Reporting Command) is specified, it will be handled with a distinct configure reporting request.

A ZigBee event is represented by a **ZigBeeEvent** object.

If an event is generated by either the local endpoint or via the base driver for an external device, the notifyZigBeeEvent(ZigBeeEvent event) method is called on all registered **ZigBeeEventListener** services for which the optional filter matches for that event. If no filter is specified, all events must be delivered. If the filter does not match, the ZigBee Driver must not call the ZigBee Event Listener service. The way events must be delivered is the same as described in Delivering Events in Life Cycle Layer chapter of the Core specification.

The ZigBee base driver SHOULD group subscriptions into one configure reporting requests) to the targeted ZigBee device. It SHOULD also notify every listener with respect to their specific expectations.

ZigBee Exception

The **ZigBeeException** can be thrown and holds information about the different ZigBee layers. The following errors are defined:

- FAILURE – (1) Operation was not successful.
- MALFORMED_COMMAND – (128) Wrong or missing field command.
- CLUSTER_COMMAND_NOT_SUPPORTED – (129) Cluster command not supported.
- GENERAL_COMMAND_NOT_SUPPORTED – (130) General command not supported.
- MANUF_GENERAL_COMMAND_NOT_SUPPORTED – (131) Manufacturer general command not supported.
- MANUF_CLUSTER_COMMAND_NOT_SUPPORTED – (132) Manufacturer cluster command not supported.
- INVALID_FIELD – (133) Invalid field.
- ATTRIBUTE_NOT_SUPPORTED – (134) Attribute not supported.
- INVALID_VALUE – (135) Invalid attribute value.
- READ_ONLY – (136) Read only attribute.
- INSUFFICIENT_SPACE – (137) Insufficient amount of free space.
- DUPLICATE_EXISTS – (138) Entry already exists in the table.
- NOT_FOUND – (139) Requested information can not be found.
- UNREPORTABLE_TYPE – (140) Attribute periodic reports cannot be issued.
- INVALID_DATA_TYPE – (141) Incorrect attribute data type.
- HARDWARE_FAILURE – (192) Operation unsuccessful due to a hardware failure.
- SOFTWARE_FAILURE – (193) Operation unsuccessful due to a software failure.
- CALIBRATION_ERROR – (194) An error occurred during calibration.

ZigBee Networking

Logical node type

The ZigBee specification defines three types of ZigBee nodes on the network:

- ZigBee Coordinator (ZC) – The most capable device, the coordinator forms the root of the network. There is exactly one ZigBee coordinator in every network. It is able to store information about the network, to act as the Trust Center and repository for security keys. Constant value **ZigBeeNode.**COORDINATOR represents the ZigBee coordinator.

- ZigBee Router (ZR) – A router is capable of extending a ZigBee network by routing data from other ZigBee devices. Constant value **ZigBeeNode.**ROUTER represents a ZigBee router.

- ZigBee End Device (ZED) – An end device contains just enough functionality to talk to the parent node (either the coordinator or a router); it cannot relay data from other devices. Constant value **ZigBeeNode.**END_DEVICE represents a ZigBee end device.

Each discovered **ZigBeeNode** on the network must have a logical node type returned by **ZigBeeNode.getNodeDescriptor().getLogicalType()**.

Network selection

The base driver provides for each available ZigBee local host – one and only one – ZigBee Host object. A ZigBee local host can represent a ZigBee chip on a USB dongle, a ZigBee built-in chip or a ZigBee Gateway Device (see [7].) This object must be registered under **ZigBeeHost**. ZigBee Host object stores networking configuration information and provides the following methods:

- start() – Starts the host.

- stop() – Stops the host.
- setLogicalType(short logicalNodeType) – Sets the node host logical type. This logical type will then be available on the host when it will restart.
- permitJoin(short duration) – Indicates if a new ZigBee device can join the network, only ZigBee Coordinator or Router can allow a new device to join the network. If the **ZigBeeHost** is a ZigBee End Device, this method must returns null.
- getChannel() – Returns the network channel. ZigBee uses direct-sequence spread spectrum and operates on a fixed channel. The 802.15.4 PHY defines 16 operating channels in the 2.4 GHz frequency band numbered from 11 to 26.
- getSecurityLevel() – Returns the network security level, 0 if security is disabled.

In ZigBee networks, the coordinator must select a PAN identifier and a channel to start a network. After that, it behaves essentially like a router. The coordinator and routers can allow other devices to join the network and route data.

After an end device joins a router or coordinator, it is able to transmit or receive data through that router or coordinator. The router or coordinator that allowed an end device to join becomes the parent of the end device. Since the end device can sleep, the parent must be able to buffer or retain incoming data packets destined for the end device until the end device is able to wake up and receive the data.

Network coordination

In case **ZigBeeHost** is configured as the network coordinator, ZigBeeHost.**getNodeDescriptor().getLogicalType()** MUST return ZigBeeNode.COORDINATOR constant value. **ZigBeeHost** object will then be able to use the following operations for managing the network:

- setChannel(ZigBeeHandler handler, byte channel) – Sets the network channel.
- refreshNetwork(ZigBeeHandler handler) – Requests the base driver to launch new discovery requests and refresh devices service registration according to current devices availability. This method is made mandatory since ZigBee specification allows devices not to notify the network when they leave it.

Networking considerations

The Network Address is a 16 bits address that is assigned by the coordinator when a node has joined a network and that must be unique for a given network in order for the node to be identified uniquely. **ZigBeeNode** provides getNetworkAddress() and getIEEEAddress() which returns device network address and device IEEE MAC address.

Security

Security management

ZigBee security is based on a 128-bit algorithm built on the security model provided by IEEE 802.15.4. ZigBee specification defines the Trust Center.

The Trust Center is the device trusted by devices within a network to distribute keys for the purpose of network and end-to-end application configuration management. All members of the network shall recognize exactly one Trust Center, and there shall be exactly one Trust Center in each secure network.

Security amongst a network of ZigBee devices is based on link keys and a network key. Unicast communication between entities is secured by means of a 128-bit link key shared by two devices, one of those is normally the Trust Center. Broadcast communications are secured by means of a 128-bit network key shared amongst all devices in the network. The master key is only use as initial shared secret between two devices when they perform the Key Establishment to generate Link Keys.

Security configuration is provided by getSecurityLevel() of **ZigBeeHost** object returning whether the security mode is activated or not on the ZigBee network.

A **ZigBeeHost** with a coordinator logical node type will acts as a the Trust Center according to the ZigBee specification, it can also be any other device of the network. The Trust Center stores all the shared network keys. **ZigBeeHost**.getMasterKey() operation returns the network master key.

Conditional permission

When a bundle registers a ZigBee Endpoint OSGi service, then the basedriver exposes this Endpoint on the outside ZigBee network and this Endpoint has the ability to communicate with the others network devices. The basedriver also provides an equivalent behavior when discovering a ZigBee Endpoint from the outside network and exposing it as an OSGi Service in the OSGi Framework service registry. It is therefore recommended that `ServicePermission[ZigBeeHost|ZigBeeEndpoint|ZigBeeEventListener, REGISTER|GET]` be used sparingly and only for trusted bundles.

---

org.osgi.service.zigbee

ZigBee 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.zigbee; version="[1.0, 2.0)"

Summary

- *ZigBeeAttribute* – Represents a ZigBee attribute.
- *ZigBeeAttributeDescription* – Represents a ZigBee attribute description.
- *ZigBeeAttributeRecord* – Represents a ZigBee attribute with some useful information for writing operation.
- *ZigBeeClient* – Represents ZigBee device that is intended to control devices over the network.
- *ZigBeeCluster* – Represents a ZigBee cluster.
- *ZigBeeClusterDescription* – Represents a ZigBee cluster description.
- *ZigBeeClusterDescriptionSet* – Represents cluster descriptions service provider.
- *ZigBeeCommand* – A ZigBee command.
- *ZigBeeCommandDescription* – A ZigBee command description.
- *ZigBeeDataTypeDescription* – Represents a ZigBee data type description.
- *ZigBeeDataTypes* – All the data types defined in the ZigBee specification.
- *ZigBeeNode* – Represents a ZigBee node.
- *ZigBeeEndpoint* – An addressable component (or logical device) that defines a communication entity within a ZigBee node.
- *ZigBeeDeviceDescription* – Represents a ZigBee endpoint description.
- *ZigBeeEvent* – An event generated by a ZigBee node.
- *ZigBeeEventListener* – ZigBee Events are mapped and delivered to applications according to the OSGi whiteboard pattern.
- *ZigBeeException* – There are several defined error situations describing ZigBee problems while invoking commands on a ZigBeeNode.
- *ZigBeeHandler* – A ZigBee Handler is a helper that manages asynchronous communication with the base driver.
- *ZigBeeHost* – Represents the machine that hosts the code to run a ZigBee device or client.
- *ZigBeeNodeDescriptor* – Represents ZigBee Node descriptor.
- *ZigBeeParameterDescription* – A ZigBee command parameter description.
- *ZigBeePowerDescriptor* – Represents ZigBee Power descriptor.
- *ZigBeeSimpleDescriptor* – Represents ZigBee Simple descriptor.
- *ZigBeeUserDescriptor* – Represents ZigBee User descriptor.

6 Considered Alternatives

- A **ZigBeeAttribute** object can also implement the **ZigBeeLocalAttribute** interface if the device is implemented locally. That is, the device is not imported from the network. The **ZigBeeLocalAttribute** interface provides a getCurrentValue() method that provides direct access to the actual value of the attribute.
- In Java, primitives types are not objects and the generic function decode(byte[]) returns an Object type. That's why Java objects types instead of primitives are used to represents ZigBee types.
- Is it possible to change the logical node type, e.g., an end device becoming a coordinator with a setLogicalType? Those changes are not described in ZigBee specifications and sound to be complex. So there is not setter for the operational mode in this specification.

Which entity has to be registered in the service registry? The ZigBeeEndpoint object and/or the ZigBeeNode object?

First, a decision has been taken (to be re-thought?) during Basel meeting (September 2012) on the number of objects to be registered: In order to avoid a burst events from 2 entities that are hierarchically related, it is decided only to register one object or the other.
Before arguing between the registration of ZigBeeEndpoint objects or the registration of ZigBeeNode objects, let's describe the two main use cases:

- 1st use case is associated to a special application like a light switch client: The client will search for light switch servers (standardized ZigBee endpoints) in the service registry before interacting with them. The bundle associated to the application will search for light switches and only for this type of services in the registry.
- 2nd use case is associated to a ZigBee network administrator (e.g., the user) who wants to explore the network and all the ZigBee devices and embedded services. The application or HMI will dynamically represent to the administrator all the devices that are available on the ZigBee network. So the application looks for ZigBee nodes in the service registry before exploring the endpoints, clusters, commands and attributes that are hierarchically hosted by these nodes.

Arguments in favor of the registration of ZigBeeEndpoint objects:

- The Endpoint brings more metadata and the information on the real functions brought by ZigBee devices. They are the first entity whose instances are standardized in terms of device profiles (e.g., ZigBee Home Automation profile standardizes light switch endpoints whereas nodes are not standardized). So the registration of this entity makes applications benefit from full OSGi service filter features to search for the right ZigBee services (Endpoints). The first use case is then easier in this case. The second use case will be slightly less easy since the application will have to ask for the node id of any endpoint and filter the list of the available unique nodes.
- Declarative Services lazy mode will be possible and very efficient for the first use case. The application will declare a service dependency towards endpoints that are light switch servers. Declarative Services lazy mode will build the service component only when light switches are available and will save hardware resources (cpu, memory) in when light switches are not available on the ZigBee network (and the OSGi service registry).

Arguments in favor of the registration of ZigBeeNode objects:

- The ZigBeeNode is the root object of the object graph of a ZigBee device. The registration of the ZigBeeNode object is thus enough to represent ZigBee network dynamicity and would avoid the multiplicity of events coming from the registration of all ZigBeeEndpoint objects. The discovery phase of the second case will be immediate to implement. However, in the first use case, the application will have to ask any node whether it hosts a light switch server. Declarative Services lazy mode will not be usable in that case.

Why having startNetwork() and permitJoin(short duration)? (And not rely on bundle API)

Every ZigBee chip/network has to be started in an independent way while the Base Driver maintains the bindings with available ZigBeeEndpoints to be exported (and that could be exported on a chip that is already started and on a chip that is not started). Relying on bundle start and stop would not make this distinction. This is why startNetwork() and permitJoin() methods are needed in the ZigBeeHost class.

Configure reporting and the White Board Pattern

ConfigureReporting command is a general command. Like every general command, it is implemented through a specific object design pattern. (e.g., Read/Write attribute are implemented with Attribute.get/SetValue() method calls)

Here, the Configure Reporting command enables an application (a client) to subscribe to application-specific events notified by a ZigBee device. In Java, you have 3 patterns available to implement eventing: Observer, WhiteBoard Pattern, Publish Subscribe (from the less to the most loosely coupling pattern). In OSGi, the Observer is not an option. Event Admin is the recommended one when it is relevant. The use of Event Admin, because it totally uncouples Publishers and Subscribers, is not possible for ZigBee eventing. That is why the use of Event Admin is not specified. Actually, ZigBee devices adapt their notification to client needs in attributes, frequency and considered range values. For ZigBee devices need to detect client needs, the Whiteboard pattern is the relevant model. We then have applied the WhiteBoard pattern like it was applied first in UPnP Device Service specification.

In brief:

- Applications interested in attribute reporting (ZigBeeEvents) register ZigBeeEventListener objects into the registry. The Attribute IDs, the frequency, attribute relevant value ranges are configurable into service properties.
- The Base Driver (for imported Endpoints) and locally implemented Endpoints request relevant listeners (relevance through service filtering) and read subscription information into service properties. Then, whenever an event matches a subscription, they call notifyEvent() method on every relevant registered listener.

Thus, registering a ZigBeeEventListener triggers ‘Configure Reporting’ commands sent by the base driver on networked devices. See ‘Event API’ section in ZigBee RFC and the javadoc for the detailed API specification.

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 Document Support

References

[1]. ZigBee Alliance, ZigBee 2007 specification, 2007

[2]. ZigBee Alliance, ZigBee Cluster Library, document 075123r04ZB, 2007

[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 2002

[5]. ASHRAE 135-2004 standard, Data Communication Protocol for Building Automation and Control Networks

[6]. Peter Kriens, BJ Hargrave for the OSGi Alliance, "Listeners considered harmful: The whiteboard pattern", Technical Whitepaper, August 2004

[7]. ZigBee Alliance, ZigBee Gateway, 2011

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

[9]. (was 2) Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

Author's Address

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

Name	Arnaud Rinquin
Company	Orange
Address	28 Chemin du Vieux Chêne, Meylan, France
Voice	+33 4 76 76 45 59
e-mail	arnaud.rinquin@orange.com

Name	Jean-Pierre Poutcheu
Company	Orange
Address	28 Chemin du Vieux Chêne, Meylan, France
Voice	+33 4 76 76 41 03
e-mail	jeanpierre.poutcheu@orange.com

Name	Fabrice Blache
Company	Orange
Address	28 Chemin du Vieux Chêne, Meylan, France
Voice	+33 4 76 76 41 03
e-mail	fabrice.blache@orange.com

Name	Christophe Demottie
Company	Orange
Address	28 Chemin du Vieux Chêne, Meylan, France
Voice	+33 4 76 76 41 03
e-mail	christophe.demottie@orange.com

Name	Antonin Chazalet
Company	Orange
Address	28 Chemin du Vieux chêne, 38240 Meylan, France
Voice	+33 4 76 76 41 03
e-mail	antonin.chazalet@orange.com

Name	Evgeni Grigorov
Company	ProSyst Software
Address	222, 50935 Cologne, Germany
Voice	+49 221 6604 501
e-mail	e.grigorov@prosyst.com

Name	Nicola Portinaro
Company	Telecom Italia
Address	Via G. Reiss Romoli, 274 – 10148 Turin, Italy
Voice	+39 011 228 5635
e-mail	nicola.portinaro@telecomitalia.it

Acronyms and Abbreviations

End of Document