|  |
| --- |
| University Of Applied Sciences, Stuttgart |
| Concepts and Implementation of the AllJoyn Bridge for the EnOcean Protocol |
|  |
| |  |  |  | | --- | --- | --- | | 1st Examiner  Prof. Dr. Stefan Knauth,  Hochschule für Technik Stuttgart |  | 2nd Examiner  Mr. Oliver Fischer,  Digital Concepts GmbH, Stuttgart | |  |  |  | |  |  |  | |
| By: Zahoor Ahmed Baloch  14/06/2016  Matriculation Nr. 753437 |
|  |
| |  |  | | --- | --- | |  | Smarte Lösungen | |

# Declaration

I hereby declare that the Master Thesis has been written only by the undersigned and without any assistance from third parties. Furthermore, I declare that no sources have been used in the preparation of this document, other than those indicated in the document itself.

Stuttgart, Germany Baloch Zahoor Ahmed

# Acknowledgement

I would like to thank my company supervisor and CEO of the company, Mr. Oliver Fischer, for his unconditional support and guidance throughout this thesis.

Many thanks to Prof. Dr. Stefan Knauth, my academic supervisor, for giving valuable guidance during the course of this thesis.

Special thanks to Mr. Dominic Böhler, for giving me valuable insights on the existing solutions and helping me with the research and design during the thesis work.

Table Of Contents

[Abstract 5](#_Toc453556980)

[List of figures 6](#_Toc453556981)

[Abbreviations 8](#_Toc453556982)

[1 Introduction 9](#_Toc453556983)

[1.1 About Company 9](#_Toc453556984)

[1.2 EnOcean Technology 9](#_Toc453556985)

[1.3 Smart EnOcean Gateway (DC-GW/EO-IP) 10](#_Toc453556986)

[1.4 AllJoyn 10](#_Toc453556987)

[1.5 AllJoyn Device System Bridge 11](#_Toc453556988)

[1.6 Motivation 11](#_Toc453556989)

[1.7 Project Plan 13](#_Toc453556990)

[2 AllJoyn 14](#_Toc453556991)

[2.1 Network Architecture 15](#_Toc453556992)

[2.2 Software Architecture 16](#_Toc453556993)

[2.2.1 AllJoyn Application Layer 16](#_Toc453556994)

[2.2.2 AllJoyn Service Framework 17](#_Toc453556995)

[2.2.3 AllJoyn Core 18](#_Toc453556996)

[3 Smart EnOcean Gateway (DC-GW/EO-IP) 22](#_Toc453556997)

[3.1 Hardware 22](#_Toc453556998)

[3.1.1 Raspberry Pi 2 model b 23](#_Toc453556999)

[3.1.2 EnOcean Pi 23](#_Toc453557000)

[3.2 Software 24](#_Toc453557001)

[3.2.1 Gateway API 24](#_Toc453557002)

[3.2.2 Learn-In 30](#_Toc453557003)

[4 AllJoyn Device System Bridge 32](#_Toc453557004)

[4.1 Device System Bridge Architecture 33](#_Toc453557005)

[4.2 Special Handlers 34](#_Toc453557006)

[4.3 Lighting Service Framework 34](#_Toc453557007)

[4.3.1 LSF Architecture 35](#_Toc453557008)

[4.4 Adapter Classes 36](#_Toc453557009)

[4.5 Mapping Bridge Interface Objects to AllJoyn 37](#_Toc453557010)

[5 Tools and Configuration 43](#_Toc453557016)

[5.1 Operating System 43](#_Toc453557017)

[5.2 Visual Studio 43](#_Toc453557018)

[5.3 AllJoyn Device System Bridge Template 44](#_Toc453557019)

[5.4 AllJoyn Explorer 45](#_Toc453557020)

[5.5 Higgns 46](#_Toc453557021)

[5.6 JSON Framework 46](#_Toc453557022)

[6 Design and Implementation of the EnOcean DSB 47](#_Toc453557023)

[6.1 Possibilities 47](#_Toc453557024)

[6.1.1 Java Bridge 48](#_Toc453557025)

[6.1.2 EnOcean AllJoyn DSB 48](#_Toc453557026)

[6.2 EnOcean AllJoyn DSB design 49](#_Toc453557027)

[6.2.1 Communication Architecture 49](#_Toc453557028)

[6.2.2 Use Case Diagram 50](#_Toc453557029)

[6.2.3 Class Diagram 51](#_Toc453557030)

[6.2.4 EnOcean Profile to DSB mapping 52](#_Toc453557031)

[6.3 EnOcean DSB implementation 52](#_Toc453557032)

[6.3.1 AllJoyn Client and EnOcean Gateway Communication 55](#_Toc453557033)

[6.3.2 Processing of JSON objects between the Gateway and the DSB 59](#_Toc453557034)

[6.3.3 Learn-in Process 62](#_Toc453557035)

[7 Conclusion 65](#_Toc453557036)

[7.1 Limitations 65](#_Toc453557037)

[7.2 Future work 66](#_Toc453557038)

[Appendix 67](#_Toc453557039)

[Bibliography 70](#_Toc453557040)

# Abstract

The Internet of Things (IoT) is the concept of connecting physical devices through the internet, but it does not define the standard process to connect them together. Therefore, different companies have defined own protocols/systems to create IoT products. These standards allow to connect devices which have the same standard, but it is also important to connect these standards together, especially with the legacy IoT protocols. For this purpose, every modern IoT standard provides a middleware connector to connect with other standards. In this Master thesis, the EnOcean protocol is integrated with the modern IoT system called AllJoyn.

The thesis goal is the design and implementation of a plugin prototype which connects the AllJoyn system with the EnOcean protocol. This goal is achievable with the help of the AllJoyn Device System Bridge (DSB) and the Smart EnOcean Gateway. First of all, an introduction of the related technologies and the purpose behind the thesis topic is presented. Then, concepts and architecture of AllJoyn, the AllJoyn DSB, and the EnOcean Gateway are explained separately in details. Finally, the communication architecture is defined between the AllJoyn DSB and the EnOcean gateway, and the implementation of chosen architecture is shown.

Concluding part evaluates the AllJoyn DSB implementation for the EnOcean protocol and identifies limitations and future work.

# List of figures

[Figure 1: IoT standards integration with EnOcean standard 7 12](#_Toc453557270)

[Figure 2: Project Plan 13](#_Toc453557271)

[Figure 3: AllJoyn Network Architecture9 15](#_Toc453557272)

[Figure 4: AllJoyn Software Architecture 16](#_Toc453557273)

[Figure 5: AllJoyn Session Types13 19](#_Toc453557274)

[Figure 6: AllJoyn Bus Attachment13 21](#_Toc453557275)

[Figure 7: Smart EnOcean Gateway (DC-GW/EO-IP)14 22](#_Toc453557276)

[Figure 8: Smart EnOcean Gateway (DC-GW/EO-IP) internal hardware15 23](#_Toc453557277)

[Figure 9: EnOcean Pi (TCM 310 Transceiver)16 24](#_Toc453557278)

[Figure 10: JSON API vs Simple API20 27](#_Toc453557279)

[Figure 11: synchronous communication with Streaming API21 28](#_Toc453557280)

[Figure 12: Synchronous Communication22 29](#_Toc453557281)

[Figure 13: Asynchronous Communication23 30](#_Toc453557282)

[Figure 14: Device System Bridge working design24 32](#_Toc453557283)

[Figure 15: Device System Bridge Architecture24 33](#_Toc453557284)

[Figure 16: DSB special Handler24 34](#_Toc453557285)

[Figure 17: LSF Architecture25 36](#_Toc453557286)

[Figure 18: DSB Adapter Classes24 37](#_Toc453557287)

[Figure 19: AllJoyn DSB template solution 45](#_Toc453557288)

[Figure 20: AllJoyn representation of EO device in different Bridges 48](#_Toc453557289)

[Figure 21: Communication Architecture of the AllJoyn System and EnOcean Protocol 49](#_Toc453557290)

[Figure 22: EnOcean DSB Use Case Diagram 50](#_Toc453557291)

[Figure 23: EnOcean DSB Class Diagram 51](#_Toc453557292)

[Figure 24: EnOcean Profile to EnOCean DSB Mapping 52](#_Toc453557293)

[Figure 25: Start Bridge Method in HeadedAdapterApp 53](#_Toc453557294)

[Figure 26: Adapter Initialize Method 53](#_Toc453557295)

[Figure 27: Getting Devices from the EO Gateway 54](#_Toc453557296)

[Figure 28: Start EnOCean DSB Sequence Diagram 55](#_Toc453557297)

[Figure 29: Handling commands by the EO DSB 56](#_Toc453557298)

[Figure 30: Reading Stream from EO Gateway 57](#_Toc453557299)

[Figure 31: Process of Handling the Action from AllJoyn Client 58](#_Toc453557300)

[Figure 32: Process of handling the telegram from the EO device 59](#_Toc453557301)

[Figure 33: JSON Object to set Device State 60](#_Toc453557302)

[Figure 34: Sending Request to the EO Gateway 60](#_Toc453557303)

[Figure 35: JSON Object Creation 61](#_Toc453557304)

[Figure 36: Handling of JSON object form the EO Gateway 61](#_Toc453557305)

[Figure 37: Creating AdapterDevice for Learn-in Process 62](#_Toc453557306)

[Figure 38: Learn-in EO Device Interface 62](#_Toc453557307)

[Figure 39: User Action Handler in EO DSB 63](#_Toc453557308)

[Figure 40: Create and Send Back Learn-in Telegram 63](#_Toc453557309)

[Figure 41: Learn-in Process 64](#_Toc453557310)

[Figure 42: Creation of Adapter Device for EO Device 67](#_Toc453557311)

[Figure 43: Update All EO Devices Abstractions in the EO DSB 68](#_Toc453557312)

[Figure 44: Update Abstraction of an EO Device 69](#_Toc453557313)

# Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| DC | Digital Concepts |
| DSB | Device System Bridge |
| GW | Gateway |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| EEP | EnOcean Equipment Profile |
| EO | EnOcean |
| ERP | EnOcean Radio Protocol |
| IoT | Internet of Things |
| IP | Internet Protocol |
| LSF | Lighting Service Framework |
| REST | Representation State Transfer |
| TCP | Transmission Control Protocol |

# Introduction

The Internet of Things (IoT) is the network of electronic devices that are capable of connecting and communicating over the internet. IoT devices are being used everywhere like in home, building and even in the city to make them smarter.

Nowadays a lot of companies are providing IoT solutions. "Digital Concepts" is one of them which specializes in smart home and smart business solution based on the EnOcean Technology.

This chapter provides the introduction to "Digital Concepts", then the technology stack used in the thesis, and finally the motivation behind the master thesis.

## About Company

Digital Concepts GmbH (DC) has the slogan “Smart Living”, and it founded1 in 2012. DC is a member of the Smart Home Deutschland, the EnOcean Alliance and the Allseen Alliance. DC received Germany’s Startup Company of the year award in 2014 by Smart Home Initiative. DC provides software solutions for the Smart Homes and Business based on the EnOcean technology. These solutions bridge EO technology with other technologies/standards so that the EnOcean protocol forms an integrated network with various technologies and standards, and provide flexibility to use different technologies together with the EnOcean. DC also provide customized solutions based on EnOcean Technology according to needs of a customer.

## EnOcean Technology

The EnOcean technology2 is the energy harvesting technology, also known as power energy or energy scavenging, is the technology through which energy is generated by external sources like wind, solar, thermal or kinetic energy. EnOcean devices are based on the EnOcean technology. These devices3 require very less amount of energy to function. For example, a battery-free radio switch requires approximately 50μWs (50μJ) of energy to generate a full radio telegram, and it gets the energy when the switch is pressed.

The EnOcean Technology works on EnOcean Radio Protocol (ERP), it is a wireless protocol, designed to transmit information efficiently with maximum reliability by using minimum power consumption while keeping other products built with EnOcean technology compatible with each other.

EnOcean Radio Protocol determines the structure of the EnOcean telegram, and the EnOcean Equipment Profile (EEP) defines the content of a telegram. ERP is also the protocol for communication between EnOcean devices and the Smart EnOcean Gateway.

## Smart EnOcean Gateway (DC-GW/EO-IP)

The Smart EnOcean Gateway (DC-GW/EO-IP) is the product of Digital Concepts (DC), created to represent EnOcean devices as IoT devices. The Smart EnOcean Gateway4 is a TCP/IP gateway, designed to integrate the EnOcean standard with emerging IoT technologies. It offers an API as simple string and JSON, make it easy for other system creators to integrate their systems easily and efficiently with the EnOcean technology.

The design of the EnOcean gateway software is easy to understand, and it is built on REpresentation State Transfer (REST) API architecture. The system integrators have the option to choose from, simple string representation or more powerful and latest JavaScript Object Notation (JSON) implementation. In thesis topic, the JSON representation is used.

The software of the EnOcean Gateway runs on a small, durable and less energy-consuming hardware, which has an EnOcean 310 chipset integrated into it. With the help of the EnOcean TCM 310 chipset, the EnOcean gateway is capable of communicating with EnOcean devices, and software of the EnOcean gateway makes those telegrams meaningful for the IP world by transforming them into the string or JSON representation.

## AllJoyn5

AllJoyn is an open-source framework, created mainly for development of applications for Internet of Things (IoT) devices. Its development is led by QUALCOMM, but it is supported by a group of companies known as the AllSeen Alliance, and the member of these alliance are Microsoft, LG, Philips, Sony and more big and small companies.

The AllJoyn Framework5 makes it easy to create applications for embedded and mobile devices through which they can easily discover, connect and communicate with each other. With the help of the AllJoyn Framework applications work seamlessly with different transport layers, manufacturers and even without the need for internet access. The AllJoyn framework software is capable of running on Linux, Linux-based mobile device, Windows devices and other lightweight real-time operating system for embedded devices.

## AllJoyn Device System Bridge

Out of the box, the AllJoyn Framework provides compatibility among only AllJoyn devices, but there are a lot of legacy IoT devices, which are not AllJoyn compatible. Examples of these types of devices are the devices from Z-Wave, ZigBee and EnOcean protocol. EnOcean devices are not even IP devices and for now, the AllJoyn system only supports IP layer. For legacy devices, Microsoft has created a middleware to bridge them with the AllJoyn ecosystem, known as Microsoft Device System Bridge.

The AllJoyn DSB creates virtual AllJoyn representations of non-AllJoyn devices like EnOcean devices so that they are treated as AllJoyn device in the AllJoyn ecosystem.

## Motivation

The number of the Internet of Things (IoT) devices are increasing rapidly, and it is estimated6 that by 2015, there were be 4.9 billion IoT devices and by 2016 they will increase by 30 percentage. IoT devices will grow in the number faster in future, and it is expected that by 2020, there will be more than 20 billion devices (approximately three times the population of the world now) connected to each other. With this number of connected devices, there will always be compatibility issues between devices. To solve compatibility issue among devices, manufacturers of IoT devices and Softwares have come together and created different alliances as shown in figure 1.

Digital Concepts as a solution provider for smart home and business is well aware of this compatibility issue and has decided to make devices based on the EnOcean Protocol compatible with IoT standards created by different Alliances/Companies.

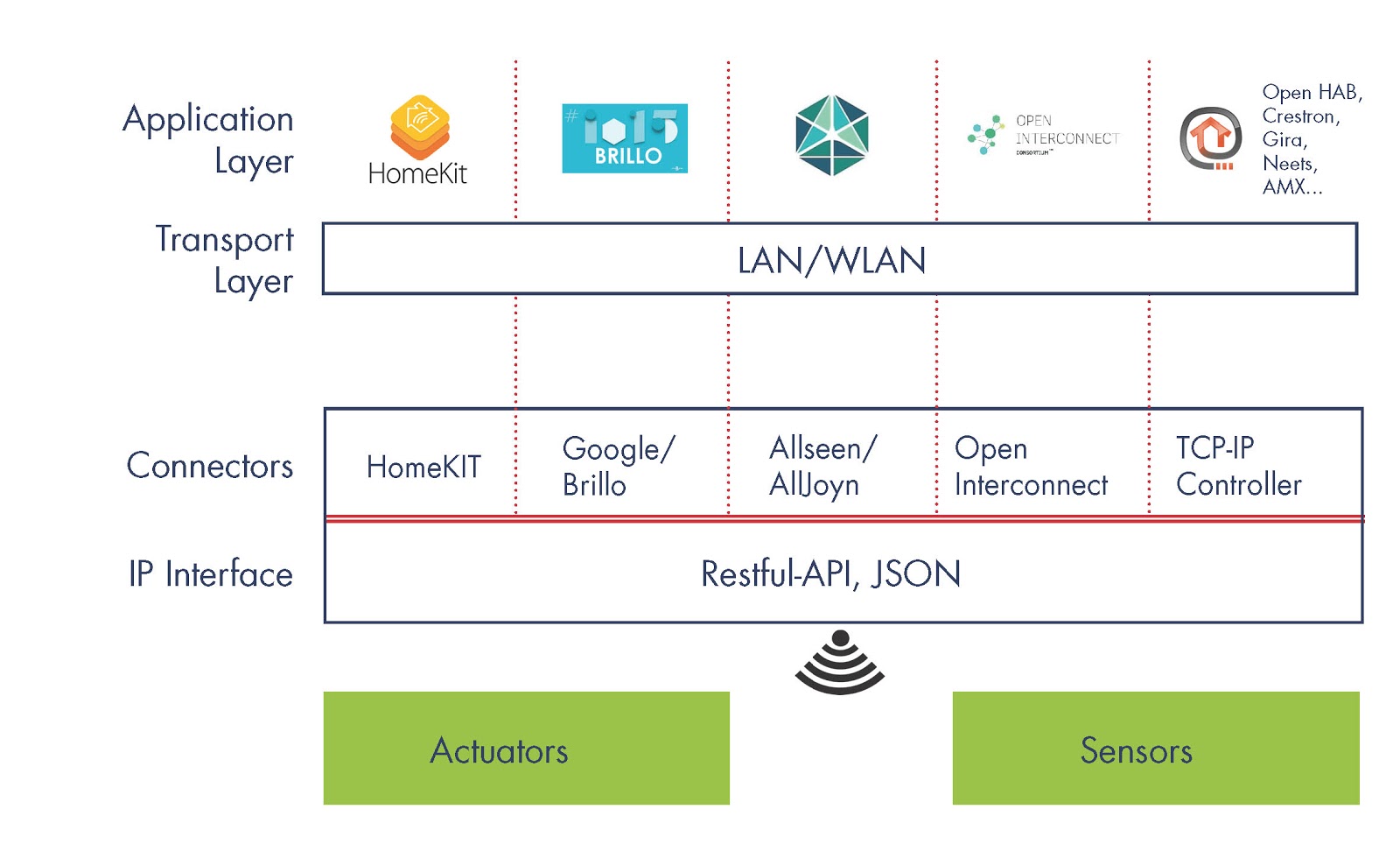
For each standard, a separate connector needs to be created which bridge the EnOcean technology with applications/devices of the corresponding standard. DC wants to build connectors for standards mentioned in below figure. One of them is the AllJoyn Software Framework, a standard framework for developing AllJoyn applications for secure and efficient communication between them and AllJoyn enabled IoT devices. In this thesis topic, the EnOcean protocol is integrated with the AllJoyn system.

Figure 1: IoT Standards Integration with EnOcean Standard 7

## Project Plan

This is overall project plan for the task of the thesis topic with their estimated time duration.

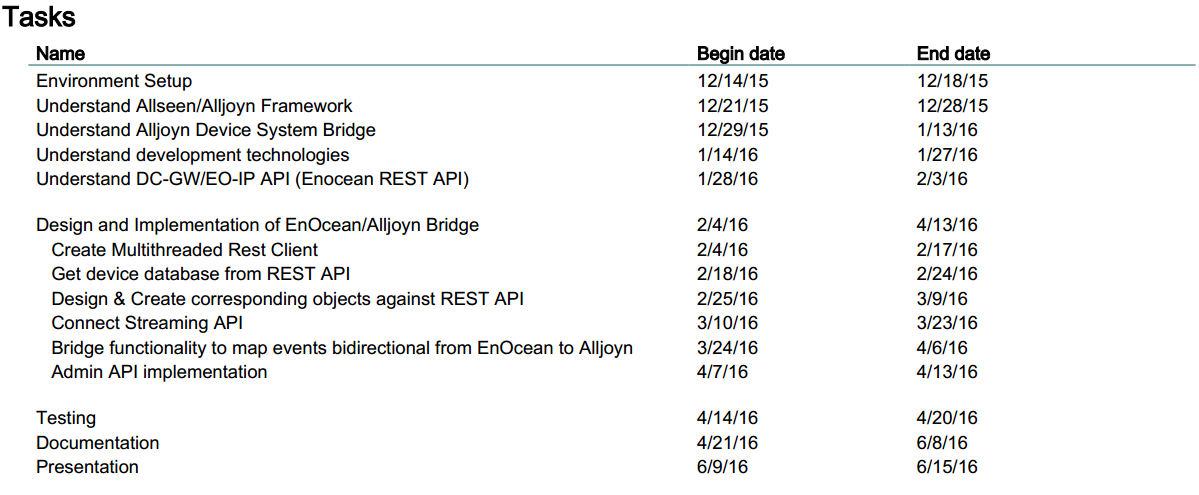


Figure 2: Project Plan

# AllJoyn8

The AllJoyn system was developed in 2011 by Qualcomm, and it was submitted to the Linux Foundation. The Linux Foundation created an association of companies called the AllSeen Alliance to make AllJoyn a standard for Internet of Things. The AllSeen Alliance is a non-profit alliance of more than 200 members, and it is growing. The purpose of this alliance is to make AllJoyn a standard for IoT products, and the members of Alliance are contributing by creating AllJoyn enabled IoT products and by developing solutions to make non-AllJoyn devices AllJoyn capable.

AllJoyn is an open source universal development framework, and it allows to create applications, services or devices which can find, connect and communicate with each other without the need for cloud or any other middleware. AllJoyn also provides cloud support through the AllJoyn gateway. AllJoyn enabled applications ranges from photo sharing application to AllJoyn services running on the air conditioner.

AllJoyn makes it possible to form ad hoc, peer to peer, the proximal and agnostic network of devices, services and applications having different transportation layers, platforms or manufacturers using D-Bus messages. For example, an AllJoyn enabled Wi-Fi coffee machine of one manufacturer can send a notification to AllJoyn enabled TV of other manufacturer assuming they are connected to the same network. In theory, AllJoyn is transport layer agnostic, however, for now, it does not have the implementation for all transportation layers.

The AllJoyn system is based on client-server technology. In the AllJoyn world server is called the producer and client as the consumer. The producer is programmed to expose its data and functionalities in the form of API, and client discovers, makes a session and consumes producer’s interface.

AllJoyn provides a broad range of options to developers for developing applications for any transport layer using their favorite programming language and running on any operating system. AllJoyn is an open source, so these features will grow in future when support for new language bindings and transport layers is created.

The architecture of the AllJoyn Framework is composed of two types, Network and Software Architectures.

## Network Architecture9

The AllJoyn Framework enables devices and applications to announce and discover each other running on the same network without any middleware. The AllJoyn system is composed of the AllJoyn Router and the AllJoyn Application.

AllJoyn Apps and Routers can interact with each other directly, but message between AllJoyn Apps are routed only through Router. The AllJoyn Router is also responsible for advertisement and discovery of services published by AllJoyn Apps.

AllJoyn is flexible to have Apps and Routers together in a single device or separately. Following three topologies are common for combining Apps and Routers.

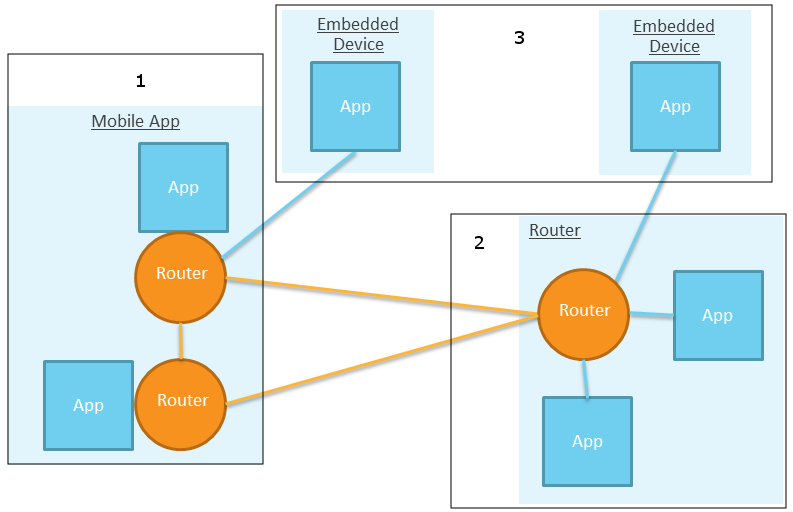
1. Each App has a Router bundled with it, and therefore the Router in this topologies is known as Bundled Router. AllJoyn Apps running on Mobile OSes and desktop OSes like Mac and Windows OS belong to this category.
2. A single Router is being used by Multiple Apps running on the same device. The Router in this category has no dependent App, so it is called the Standalone Router. The Standalone Router runs as a background service like a daemon process on the Linux system and AllJoyn Apps running on the same system connect to this Router.
3. In this category, the App running on a device uses the Router running on another device. Apps running on embedded devices fall into this category.

Figure 3: AllJoyn Network Architecture9

## Software Architecture9

The AllJoyn Apps and the AllJoyn Router forms an AllJoyn system. The architecture of an AllJoyn Application is formed by AllJoyn App Code, AllJoyn Service Framework and AllJoyn Core Library. The AllJoyn Router itself is a complete application which helps AllJoyn Apps to interact.

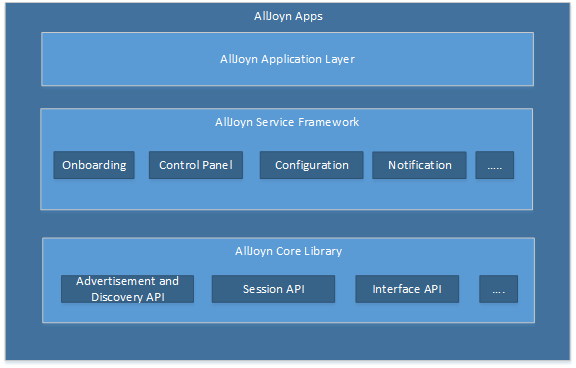


Figure 4: AllJoyn Software Architecture

### AllJoyn Application Layer10

The AllJoyn Application is a collection of services. These services are created using AllJoyn Core Library or already existing services in AllJoyn Service Framework Library. AllJoyn Service Framework Library provide a common set of services needed by IoT devices, and AllJoyn Core Libraries provide the set of API for creating AllJoyn core services for AllJoyn applications.

The AllJoyn application development is recommended using the AllJoyn Service Framework Library because only Apps created through these services will guarantee to be compatible with Apps and devices created by the Allseen working group. Developers can also create their services using AllJoyn Core Library. It is recommended to use Actions and Events mechanism of AllJoyn Core Lib while creating user-defined services.

There are two types of AllJoyn Apps, Standard and Thin client. Standard client Apps are built for standard Operating System (OS) and mobile OS, while thin clients are for embedded Operating Systems. AllJoyn Core and Service Framework have separate Libraries for Standard and Thin clients.

### AllJoyn Service Framework11

The AllJoyn service framework provides a set of common services, which are needed by many IoT devices to communicate and work with each other. Although all services are not used in the thesis, however, it is useful to have the idea about the basic services to work with AllJoyn enabled devices which are using these services. Following are the most common services known as Base Services available in the AllJoyn Service Framework.

#### Onboarding 12

Onboarding, as its name suggest, is the service that provides functionality to onboard new device to the proximal network. This service is particularly suitable for devices that do not have any user interface to configure them, like SmartPlug or LiFX bulb.

Onboarding service is currently supported for Wi-Fi only, but it can be developed for more hardware like Bluetooth Low Energy (BTLE) when they become commonly used in IoT devices.

#### Configuration

AllJoyn provides configuration service for devices which need their attributes to be configured by the user of a device. For example, setting a password or friendly name for the device.

#### Notification

The AllJoyn Notification Service Framework provides a service for devices to send and receive the text message from other devices which have Notification service implemented.

#### Control Panel

The AllJoyn Control Panel Service enables the application to provide control panel so that remote application renders UI widget corresponding to control panel. The Control panel is a collection of UI widgets.

### AllJoyn Core13

The AllJoyn Core Library consists of a low-level set of APIs for communicating with the AllJoyn network. The AllJoyn Service Framework is created from these set of APIs and developers can create their services using these APIs. AllJoyn Core has two versions of Libraries, Standard and Thin Core Library.

The AllJoyn Standard Core Library is created to run on devices which do not have any constraints, like general-purpose computers, while Thin Core Library provides support for devices which have memory and network constraints like embedded devices. AllJoyn Core API provides following main features.

#### Advertisement and Discovery

The AllJoyn Application uses “About Announcement” or “Well-known name” method to announce its services and discover services of other AllJoyn application on the network.

Well-known method is a low-level method for advertisement and discovery of services, it is needed for special scenarios otherwise use of the "About Announcement" method is suggested, underneath About Announcement also uses well-known mechanism.

The “About Announcement” is the common way to advertise meta data by the AllJoyn application for other remote application. The advertised data contain application properties like name, vendor, version, description and other similar properties.

#### Session Creation

The AllJoyn Core Framework provides complete session management API for AllJoyn applications. An AllJoyn App announces its services typically by “About Announcement” and other AllJoyn app discovers it, finds its UniqueName and makes a session.

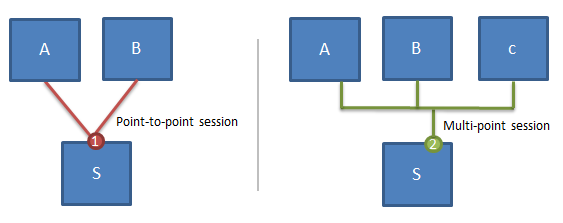
AllJoyn apps can have point to point or multipoint sessions. Point to point session is a direct session between two apps, and the multipoint session is a shared session between groups of application. In the following figure, left side shows a point-to-point session S on port 1 and on the right side a multi-point connection S on port 2.

Figure 5: AllJoyn Session Types13

#### Interface definition of methods, signals and properties

The Interface definition of an AllJoyn application contains implemented services and members (actions, signals and properties) and the interface is defined in the form introspectable XML file. AllJoyn Client App, introspect XML to know about object path and services offered by App. The Introspection functionality is built in feature of AllJoyn core library.

Implementation of the interface is available as BusObject on the AllJoyn bus, and client application calls methods of the BusObject using remote procedure call mechanism. Actions and events are the most used members in AllJoyn application. But applications/devices can also be created using the combination of events, actions and properties or only properties like LIFX bulb.

#### Events and Actions

The Events and actions mechanism offered by the AllJoyn Core allows applications to send and receive signals and in the same way Apps can provide actions for other Apps to call. The events and actions functionality is useful to create the dynamic interaction between devices. If This Then That (IFTTT) rules can be created using events and actions, for example: when a door open or close event triggers, then lamp turn on or off action is called.

The event is represented as a signal in AllJoyn interface with a description of the event and in the same way action as a method with description.

* **Property**

The AllJoyn property is another useful member of AllJoyn interface, which is an easy way to represent data of device attribute, for example showing the temperature of a thermostat device. But the property has more to offer than only showing values, it can also be used to change attribute value when the property is writeable, and property can also be configured to emit a signal on value change.

#### Object creation and handling8

The AllJoyn core provides the API for creating AllJoyn objects, connecting and handling communication between them. The AllJoyn system is a distributed virtual bus, which is responsible for connecting AllJoyn Daemons (Router) and linking AllJoyn bus attachments. AllJoyn BusAttachment object is the representation of AllJoyn application on AllJoyn Bus, and service offered by the application are registered to Bus Attachment as BusObject, so they are available on the AllJoyn bus and accessible by remote ProxyBusObject.

Each Bus Attachment is assigned auto-generated temporary unique connection name when it is connected to the AllJoyn Bus. The Bus Attachment requests Router to assign the well-known name to BusObject, if the well-known name is not being used by any other BusObject, it is assigned to BusObject and BusObject is registered with the Router by Bus Attachment. The remote BusObject on the AllJoyn bus can discover those bus objects by their well-known name, and consume exposed services.

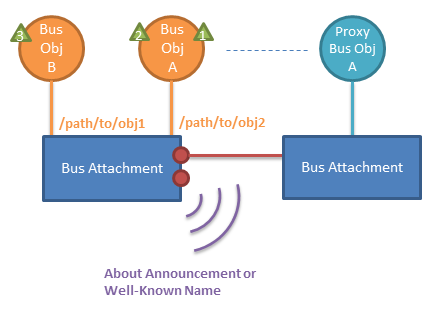


Figure : AllJoyn Bus Attachment13

# Smart EnOcean Gateway (DC-GW/EO-IP)

The purpose of the Smart EnOcean Gateway (DC-GW/EO-IP) is to resolve connectivity problem of EnOcean technology with TCP/IP world. EnOcean Equipment Profile (EEP) defines the content of telegram of EnOcean device and gateway provide a simple IP interface for complex EEP of devices. With the help of EnOcean receiver, EnOcean IP Gateway can understand and translate the messages of EEP-profile that were transmitted by EnOcean devices, and software of gateway translate these message into a simple and meaningful JSON representation so that other IoT devices can understand EO messages.

In this chapter, the EnOcean Gateway is explained in detailed. Firstly, the hardware part of EO Gateway is discussed, and then the software is explained.

## Hardware14

The EnOcean Gateway software runs on an energy-efficient hardware, fitted with an EnOcean TCM 310 chipset, and an optional antenna, which strengthen telegram receiving capability of the gateway, shown in below figure.



Figure 7: Smart EnOcean Gateway (DC-GW/EO-IP)14

### Raspberry Pi 2 model b14

The inside of the EnOcean Gateway hardware is Raspberry Pi 2 model b, shown in the figure. Below is the hardware specification of the EnOcean Gateway.

|  |  |
| --- | --- |
| Device name | DC-GW/EO-IP |
| Voltage Supply | 5 V DC , 600 mA |
| Dimensions (HxWxD) | 41 x 72 x 109 mm |
| Weight | 285 g (without Power Supply) |
| Operating temperature | 5 to 35 °C |
| Electric consumption (max) | 3,0 W |
| Connections | 1x LAN(RJ45), 1x Antenna connector, 4x USB |
| Range | Without obstacles up to 30m |

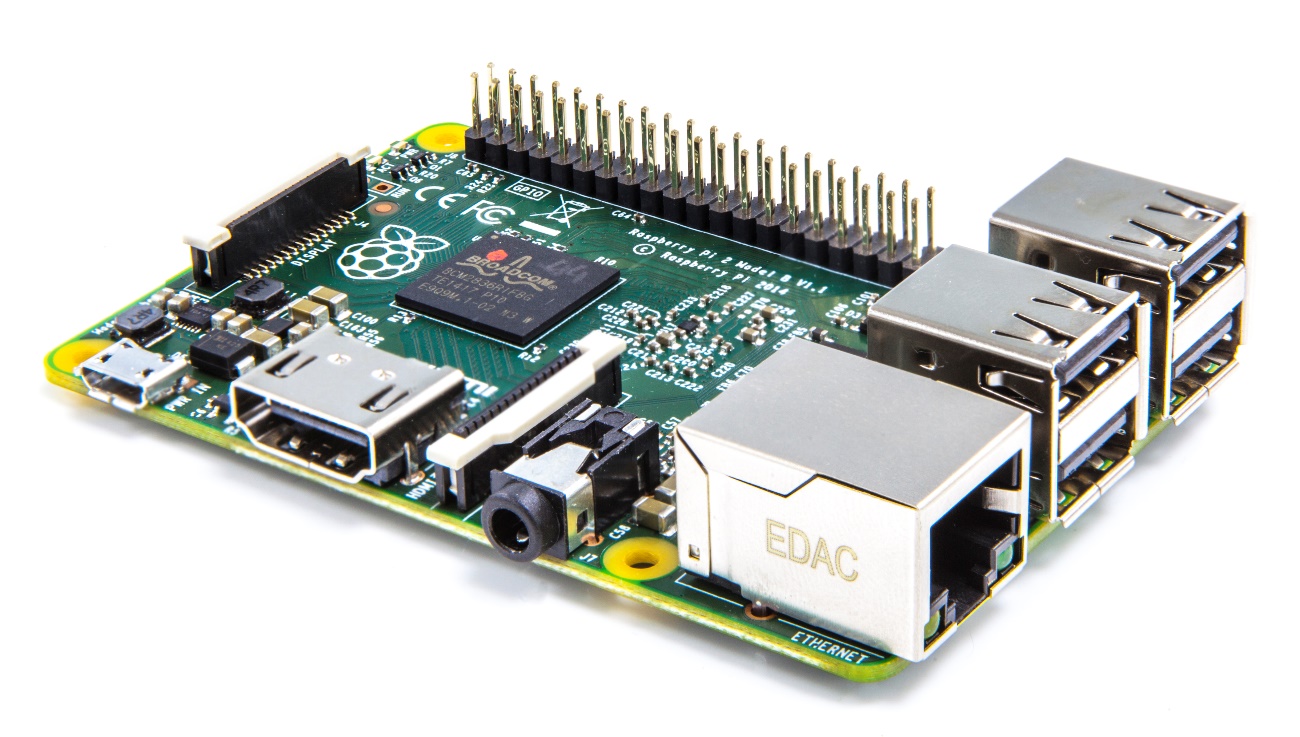


Figure 8: Smart EnOcean Gateway (DC-GW/EO-IP) internal hardware15

### EnOcean Pi

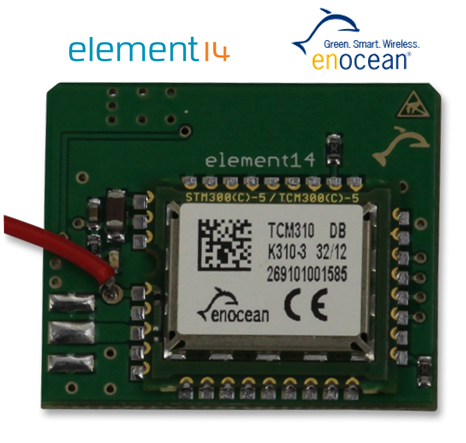
The EnOcean Pi is the key hardware element in the EnOcean gateway, and it is a bidirectional wireless transceiver module created by EnOcean. It is an extension of the smart card which can be connected using its GPIO port with Raspberry Pi. The EnOcean Pi does not require any external power source to work, and it receives power from the Raspberry Pi. The EnOcean Pi is based on the TCM 310 transceiver that makes it possible for Raspberry Pi to communicate with EnOcean devices. 

Figure 9: EnOcean Pi (TCM 310 Transceiver)16

## Software

### Gateway API

The Gateway API provides an interface for the communication between the IP client and EnOcean devices. The API interface is designed in a way so that future EnOcean devices are easy to adopt, and translation of EnOcean devices to Internet of Things is simple.

The communication between the IP client and IP gateway is possible by two APIs, simple and JSON. The gateway interface offers following functionalities18:

* System functions to get the gateway information.
* Profile functions to get information about profiles of supported EnOcean devices.
* Device functions to get information of stored EnOcean devices in the Gateway.
* Transmission functions are GET and PUT methods, with the GET method, the client can be notified of any state change on the Gateway, and with the PUT method commands are transmitted to the Gateway.

#### Simple API19

The Simple API is a way to get information from the EnOcean gateway. The simple API consists of a single TCP socket connection. After the successful connection and authentication, the gateway begins to streaming states. First, the latest status of all connected EO devices is displayed in row-based methods, then ongoing events in the line-based methods. Table 2 summarizes all relevant information about the Simple API.

#### JSON API17

The Smart EnOcean Gateway receives the telegrams sent from smart devices, interpret and translate them into a JSON object, available at (<IP gateway>: 8080) as REST API. The same process in reverse order is performed when the EnOcean gateway sends a telegram to an EO device.

The JSON representation of the EO devices on the REST API gateway simplifies control of smart devices through a web interface, an application (Android or iOS apps) and software. Because it is relatively easy to use REST API by software application compared to the EEP-profile.

The client can get device information (profile, state ...) with Get, and modify or add a device with Put and Post, and remove devices with Delete commands to REST API.

#### Device Profile

Each EnOcean device has its profile (EEP) representation in the EnOcean gateway in the JSON notation. Some devices have more than one profile. All information about device includes in Profile node, and profile node has functionGroups. Function groups have the functions supported by devices. Below is the JSON profile representation of an EnOcean device. This example shows that the device has only one functionality, and it is from the device.

FunctionGroup (From): Device can only send the telegram.

Functions (Co2): Device is capable of measuring Co2 value.

#### Device State:

Each EO device has functionalities (ON, OFF, Open, Close, Boolean, number ...), and these are also the states of devices. It is possible to know about the latest status of all connected devices or a single devices using its device id through the EnOcean Gateway API. For example using GET request to this URL „<IP of Gateway>/devices/{device-Id}/state?“ will give the current state of the device. Similarly, state of the device can be updated using PUT request.

#### Telegrams:

The Telegram is an EnOcean Radio Protocol (ERP) based signal sent to or received from the EnOcean device. Telegram is a command to the device to execute a function or from the device containing the information about the state of the device. Gateway keeps the history of every telegram either received or send.

Below table summarizes the main differences between Simple API and REST API:

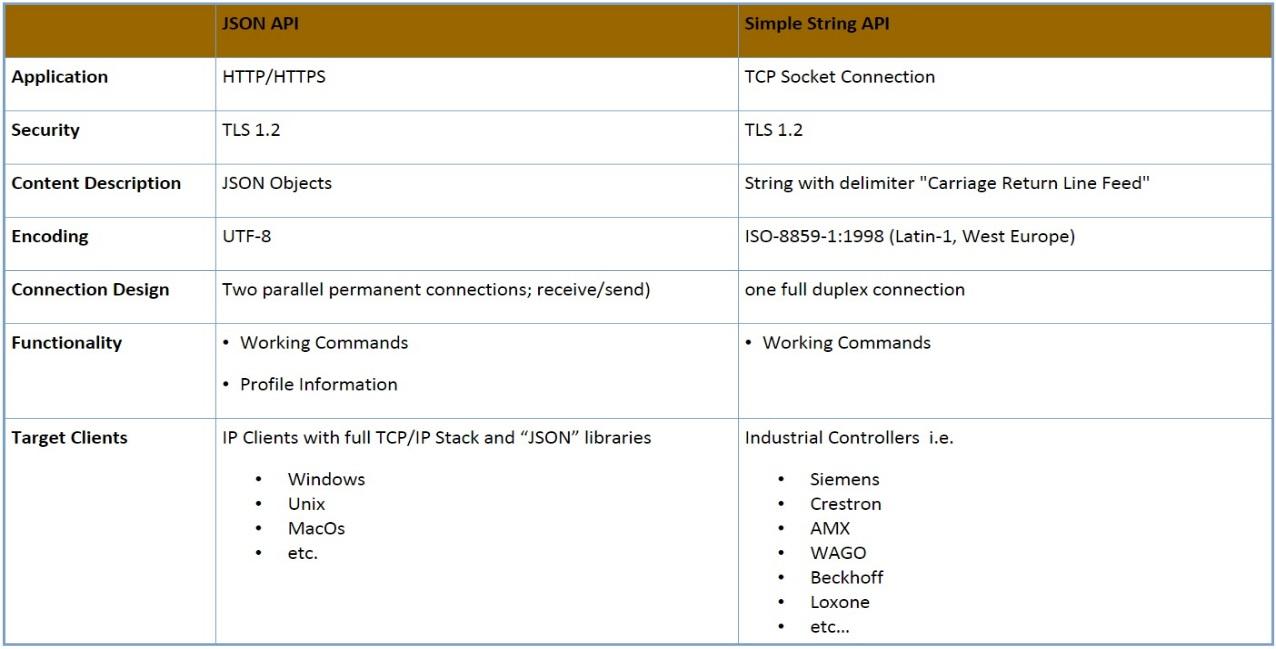


Figure 10: JSON API vs Simple API20

#### Streaming API: Chunked Transfer Encoding Method

The Streaming API is the one stop to know about any change in the EnOcean Gateway. The Streaming API provides the states of all devices or change in state of a device in real time. When the state of a device is changed, a telegram containing the change of state information is soon available on streaming API of the gateway. When Streaming API is accessed, all EO devices know to the gateway are returned containing information about their current state.

#### Communication Design

There are three different ways of communication from the client to the EO gateway. The first is the synchronous communication with keep alive. The second is the synchronous communication with Streaming API method and the last is the asynchronous communication.

#### The synchronous communication with Streaming API21

The primary method for data exchange with the gateway is the JSON API. The Streaming API method has been implemented for the JSON API. The Streaming API method works on the basis of "chunked encoding Transfer’ procedure, it is used by the client for synchronous communication. After a successful connection between the client and the gateway, the client sends a request (GET /<IP-Gateway: Port>/devices/stream) to the gateway. First, the last states of all EO devices know to the gateway are returned, then the occurrence of each event in the gateway. The connection will remain on as long as the gateway is turned on, or the client does not close the connection. The figure below shows the streaming API process.

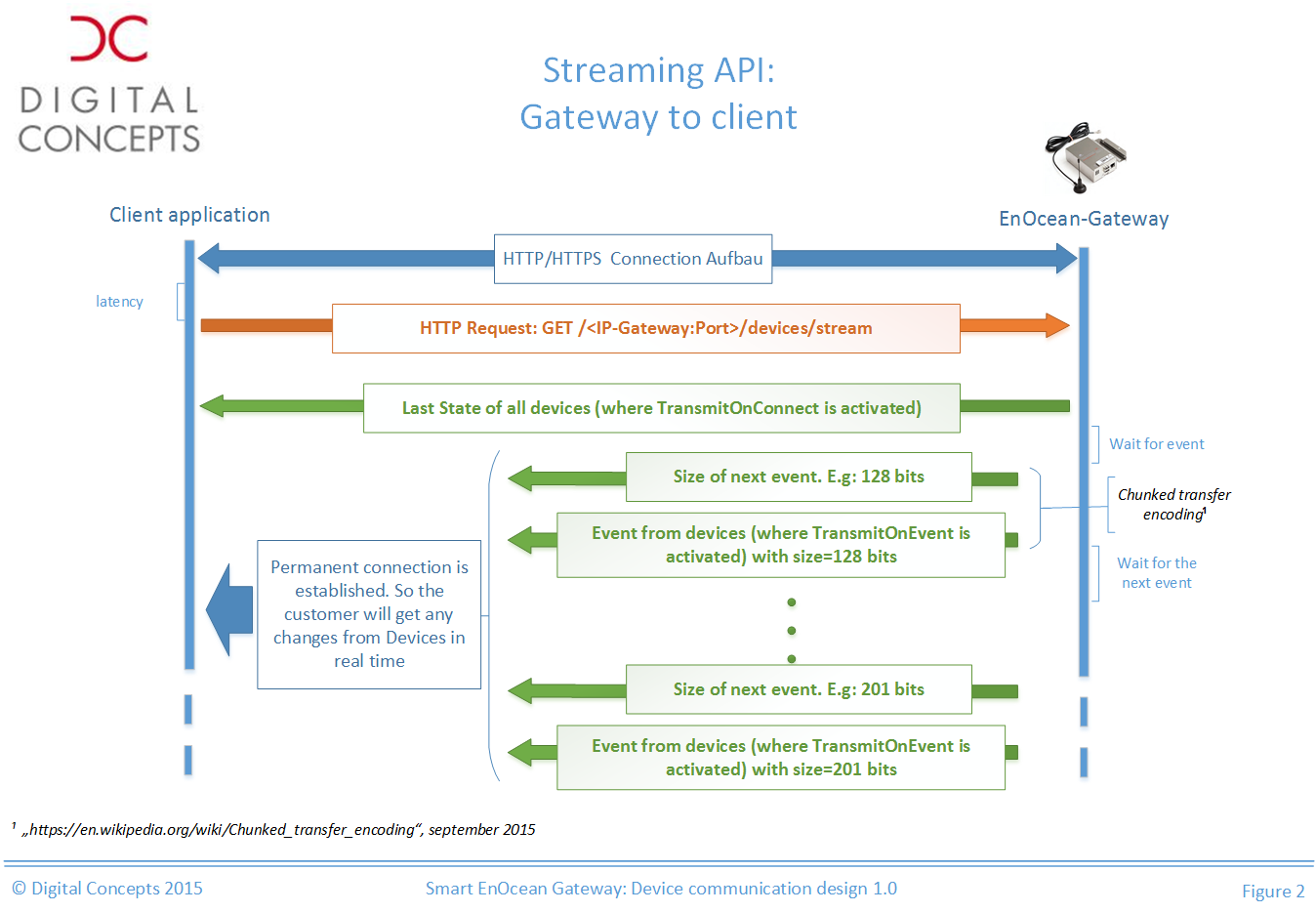


Figure 11: synchronous communication with Streaming API21

#### Synchronous communication22

In synchronous communication after sending a request, the client has to wait for the response from the EO gateway before it can send the next request. The Figure below illustrates the synchronous communication.

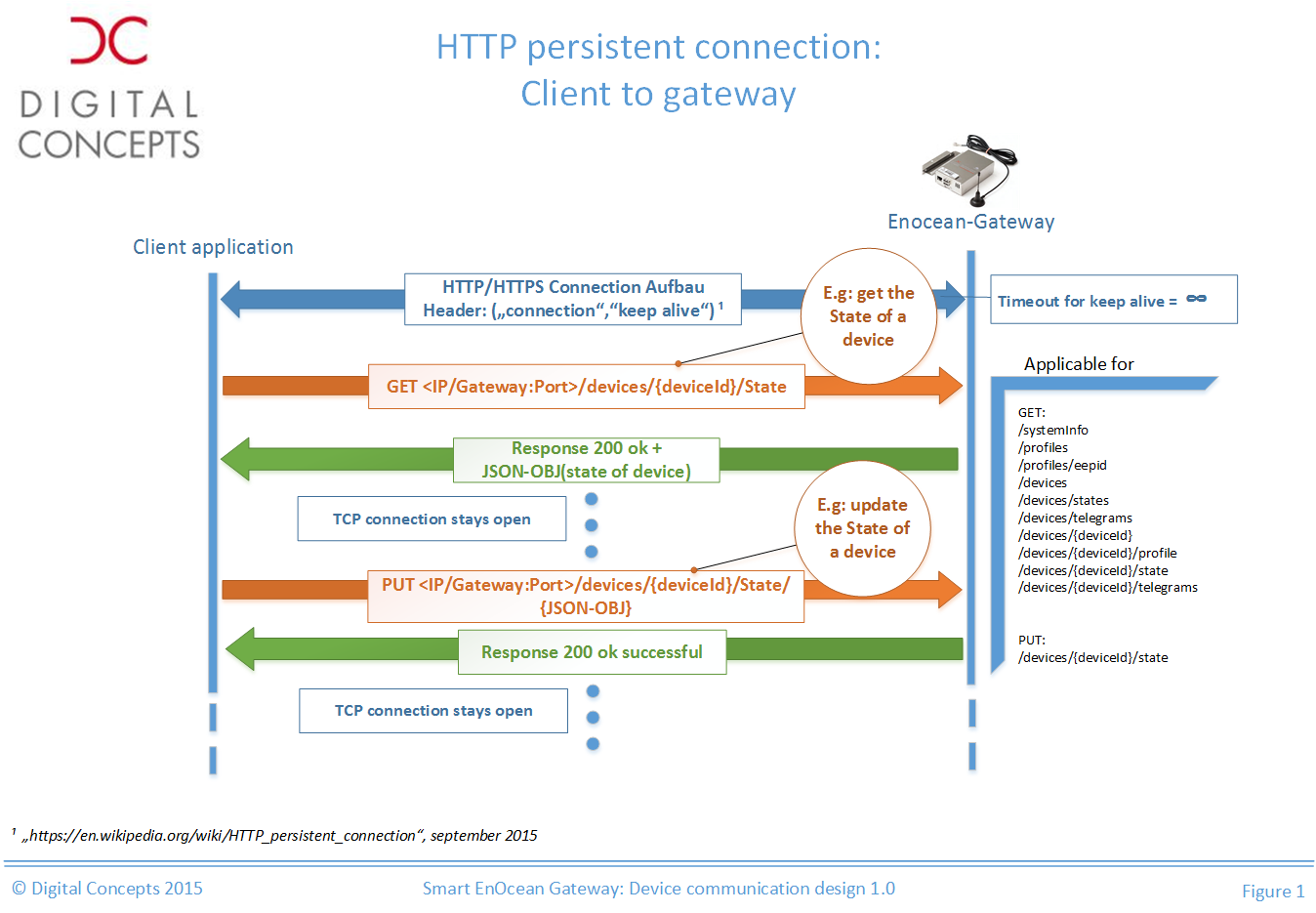


Figure 12: Synchronous Communication22

#### Asynchronous C*ommuni*c*ation23*

The client communicates asynchronously with the gateway for the EO telegrams sent from the gateway. Because some devices do not return a confirmation telegram, and some device goes to sleeping mode to save energy.

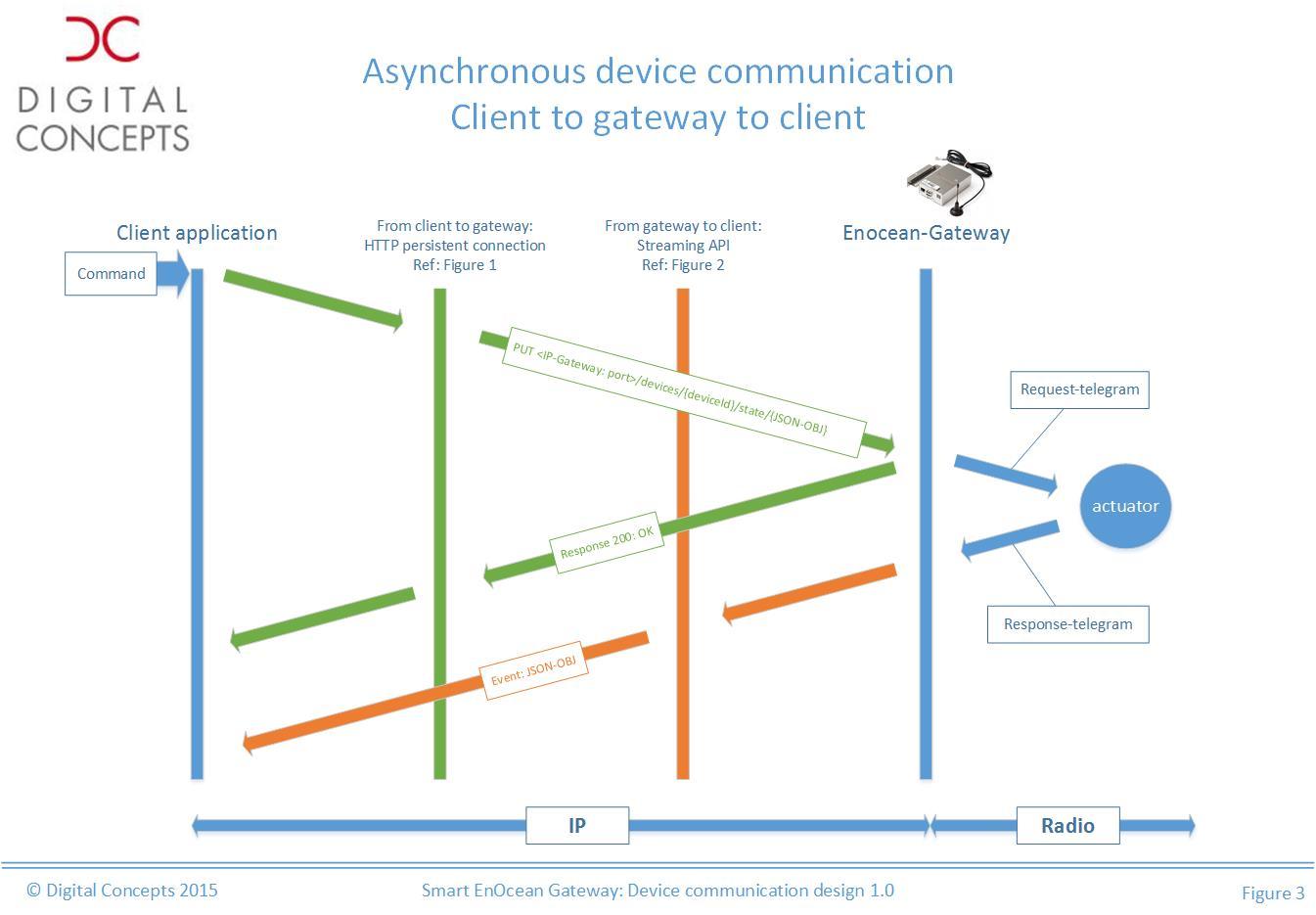
When the gateway receives a request from the client to change the state of a device (Put /<IP-Gateway: Port>/devices/{deviceid}/state), the gateway sends an EO telegrams in the air for the particular device and sends an OK response status to the client, without getting confirmation response from the device. The disadvantage is that telegram can be lost in the air, and the client will not know about it. The reply of the device is available to the client through streaming API. The figure below shows an asynchronous process of the gateway.

Figure 13: Asynchronous Communication23

### Learn-In

One of the most important features of the EO gateway is the learn-in process for a new EO device. The gateway has the web interface and the REST API to learn-in a new EO device.

The learning process can be carried out via the web interface of the gateway (<IP of the gateway>/dcgw/enocean/newdevice) or Admin REST API. There are two ways to learn-in an EO device via the web interface: automatic and manual. A new device is learned in two steps.

In the first step, gateway and device come to know about each other. Learn-in mode of the gateway is activated, and the device sends a learn-in telegram to the gateway. If the gateway receives the learn-in telegram, it will show a device in new device area, but the device is not added to the database.

In the second step, the gateway adds the device database. To save the device, necessary information about the device is required. The user can enter required information in edit device area. After filling required fields, OK or Apply is pressed to complete the learn-in process of the device.

The learn-in process implementation using Admin API is explained in the chapter six.

# AllJoyn Device System Bridge

AllJoyn24 gives freedom to manufacturers to create AllJoyn applications/devices using a broad range of operating platforms and language bindings. However, there are a lot of existing devices which are created using their standard/technologies. In order to bring existing devices to the AllJoyn ecosystem, Microsoft developed the Device System Bridge (DSB). The AllJoyn DSB is cost efficient and fast way to bring non-AllJoyn devices to the AllJoyn world without modifying existing behavior of devices. The DSB translate non-AllJoyn devices using AllJoyn C API into virtual AllJoyn objects and expose them to the AllJoyn bus.

Currently, officially supported DSBs for Home Automation system like Z-Wave, ZigBee and building automation system like BACnet are available. However the DSB is open source, so it is possible to customize the DSB for other home and building automation standards, like the EnOcean Protocol.

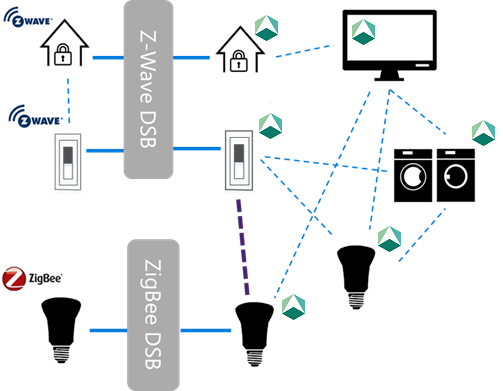


Figure 14: Device System Bridge working design24

The AllJoyn DSB can be used to integrate existing incompatible technologies like Z-Wave, ZigBee and EnOcean. The DSB creates AllJoyn representation of non-AllJoyn devices regardless of their native technology, so they are treated and available as AllJoyn devices on the AllJoyn Bus and can communicate with each other and other AllJoyn devices as shown in the figure. The DSB acts as a middleware to connect non-AllJoyn devices to AllJoyn devices. The DSB only provides the mapping from non-AllJoyn world to the AllJoyn world, not the other way around.

## Device System Bridge Architecture24

The AllJoyn DSB Architecture is based on three main component: Network Stack, Bridge and Adapter.

Network Access Stack: This is a connection to the network of non-AllJoyn devices. The connection to the device network is possible using a library provided by the device manufacturer or via the web service API of the device.

Adapter: The Adapter is responsible for configuring the DSB application and creates and manages abstract representation of devices available on the network stack.

Bridge: The Bridge is responsible for translating each native device using its abstraction created by Adapter into AllJoyn devices and creating individual bus attachment for each device.

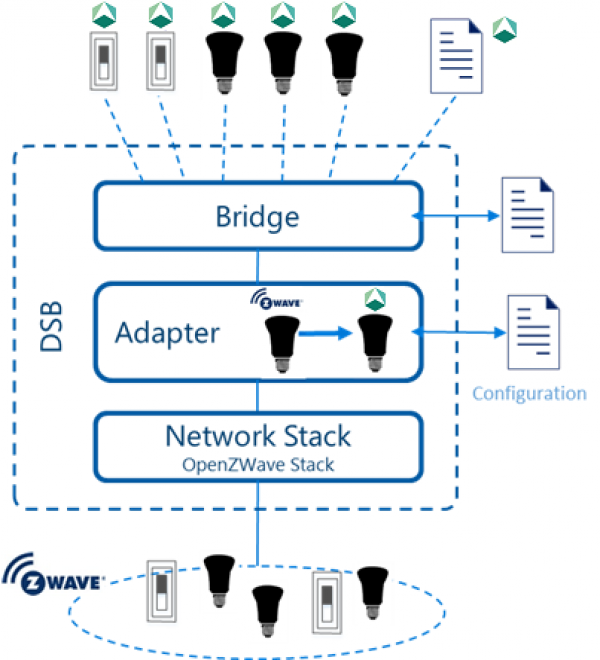


Figure 15: Device System Bridge Architecture24

## Special Handlers24

AllJoyn provides common services in the service framework like Lighting Service Framework (LSF), Control Panel, Home Automation Entertainment and others. The AllJoyn DSB can implement those service and expose with callback handlers. The existing version of DSB template implements Control Panel and LSF services and provides callback functions to connect their interfaces in the bridge. Following figure show the LSF handler implementation and it is same for other services.

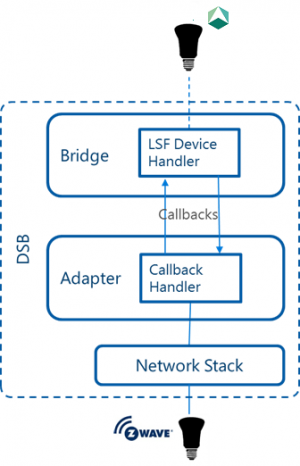


Figure 16: DSB special Handler24

## Lighting Service Framework

The Lighting Service Framework (LSF) is the part of open source “Connected Lighting Project” which provides the standard way of communication between the AllJoyn enabled lighting devices so that manufacturers can create lamps which are compatible with lamps built by other manufacturers. The AllJoyn DSB implements the LSF service, and EO DSB implements the LSF handler for two EO profiles.

LSF provides below features:

**Lamp Details:** Lamp details are metadata about a lamp exposed by lamp service.

**Presets:** Presets are different combinations of lamp properties which are saved for future use.

**Groups:** LSF allows users to bring lamps together in a group or other groups together and control them in a group like a single lamp.

**Control:** Control feature of LSF allows a user to control hue, saturation, brightness and color temperature properties of groups or individual lamp, depending on the properties supported by a lamp and value of these properties.

**Scenes:** Scenes are properties or effect settings for any combination of lamps or groups for a specific lighting mode. Different scenes can be grouped into a Master scene, and they can be used more than once.

**Effects:** The LSF allows users to create transition and Pulse effects for scenes.

**Events:** Every scene created in the LSF has an associated event which is triggered when the scene is applied.

**Actions:** The LSF advertises scenes as AllJoyn actions.

### LSF Architecture25

The LSF is a multi-tier architecture for AllJoyn enabled lamps with below components that reside inside and outside of the lamp, and runs over an AllJoyn supported IP carrier.

1. Lamp Service
2. Lighting Controller Service

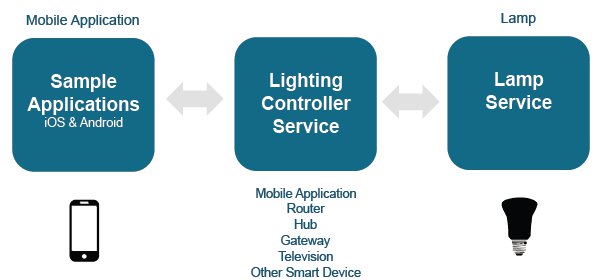


Figure 17: LSF Architecture25

#### Lamp Service:

The Lamp service is embedded in the lamp firmware by the manufacturer to create a LSF enabled Lamp. The AllJoyn DSB implements lamp Service in the Bridge and provides a handler that needs to be implemented in the Adapter project to control the lamp service.

#### Lighting Control Service:

The Lighting Control Service is an important part of the LSF but resides outside the lamp device. It finds and connects to the Lamp Services available on the same network. It can reside in the AllJoyn router, gateway, hub or in the form of a mobile application.

## Adapter Classes24

Adapter classes shown in the figure are provided by the DSB template to create the abstraction of a non-AllJoyn device in the AllJoyn DSB. The implementer of the DSB initiate an Adapter object and pass it to the Bridge so that the Bridge creates AllJoyn bus attachment for each device in the Adapter.devices list.

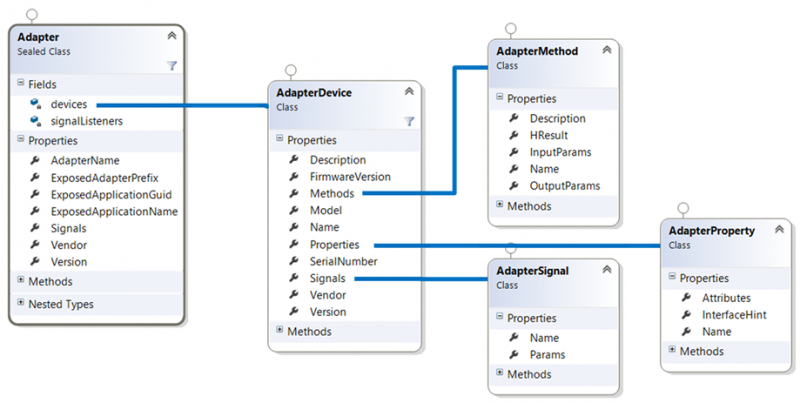


Figure 18: DSB Adapter Classes24

These Adapter classes and other classes of the Adapter Lib project have their interface defined in the Bridge project of the DSB.

## Mapping Bridge Interface Objects to AllJoyn26

The AllJoyn DSB defines interfaces which map to AllJoyn objects. Here are these interfaces explained and how they map to AllJoyn.

### IAdapter

IAdapter interface creates and controls devices which are exposed to the AllJoyn bus. The Bridge uses enumeration method of IAdapter to get the list of devices. Configuration and lifecycle management methods of IAdapter handle configuration and lifecycle of a device.

Adapter class in AdapterLib project implements IAdapter interface so that devices are exposed on AllJoyn bus, and implementation of IAdapter interface depends on devices being advertised to AllJoyn bus.

The Adapter object is available on AllJoyn bus as an AllJoyn Services having a name in below format:

*{ExposedAdapterPrefix}.DeviceSystemBridge.{AdapterName}*

By Default, every Adapter service exposes two com.microsoft.alljoynmanagement.config interfaces, /AdapterConfig for adapter configuration and /BusConfig for the bridge configuration.

Below table shows some properties of IAdapter interface which are required to be implemented, and how they map to AllJoyn properties.

|  |  |  |
| --- | --- | --- |
| IAdapter Property | Description | Bridge Mapping |
| AdapterName | Used as a model of adapter and suffix in the name of adapter. | AllJoyn About Data Model Number |
| ExposedAdapterPrefix | Prefix of the bridge exposed on AllJoyn bus | AllJoyn Bus Attachment’s Advertised Name |
| ExposedApplciationGUID | Unique identification of adapter, and it is also used in about data of devices handled by adapter. | AllJoyn About Data Application ID for adapter and its devices. |
| ExposedApplicationName | Friendly name exposed by adapter, and it is also used by devices of adapter. | AllJoyn About Data Application Name for adapter and its devices. |
| Vendor | Vendor name of adapter | AllJoyn About Data Manufacturer |
| Signals | Signals of adapter | AllJoyn Events |
| Version | Software version of adapter | AllJoyn About Data SW Version |

Table 1: IAdapter Properties

Followings are the most important methods of IAdapter interface:

#### IAdapter::Initialize

This method is called by bridge project to run initialization logic of adapter, and that logic can be an HTTP connection to a service endpoint or running another program to discover devices.

#### IAdapter::Get/SetConfig

These methods are available in adapter to access configuration data by bridge and expose to AllJoyn through “com.microsoft.alljoynmanagement.config” interface.

**IAdapter::EnumDevices**

This method returns a list of devices to the bridge and bridge advertise them as virtual AllJoyn devices on the AllJoyn bus.

### IAdapterDevice

IAdapterDevice interface represents the abstraction of a device that is exposed to the AllJoyn bus as an AllJoyn object. Methods, signals and properties of the adapter device can have any behavior, but usually, these are the replica of original device behavior on native network.

IAdapterDevice is available on AllJoyn bus with the following name:

*{ExposedAdapterPrefix}.DeviceSystemBridge.{AdapterName}*

All Methods, signal and properties of the device are exposed by a single AllJoyn interface, which has a name like below format:

*{ExposedAdapterPrefix}.{AdapterName}.{Name}.MainInterface*

Like IAdapter interface, some properties of IAdapterDevice are required to be implemented. Below table shows those properties, and how they map to AllJoyn.

|  |  |  |
| --- | --- | --- |
| IAdapterDevice Property | Description | Bridge Mapping |
| ControlPanelHandler | Control Panel handler for the device | Exposed as an org.alljoyn.ControlPanel.ControlPanel under a /ControlPanel bus object |
| Description | Description of the device | AllJoyn About Data Description |
| FirmwareVersion | Version number of the device | AllJoyn About Data Firmware Version |
| Icon | Graphical image used by GUI application to represent this device. | AllJoyn About Icon |
| Methods | List of methods supported by device | AllJoyn Actions |
| Model | Model of the device | AllJoyn Bus data model number |
| Name | Name of the device | AllJoyn About Data Device Name |
| Properties | List of properties of the device | AllJoyn Properties |
| SerialNumber | Serial Number of the device | AllJoyn About Data Serial Number |
| Signals | List of signals exposed by device | AllJoyn Signals |
| Vendor | Manufacturer name of the device | AllJoyn About Data Manufacturer |
| Version | Software version of the device | AllJoyn About Data SW Version |

Table 2: IAdapterDevice Properties

### IAdapterProperty

IAdapterProperty interface represents the list of data values of device attributes exposed to the AllJoyn bus. Each data value is represented by IAdapterValue interface, and IAdapterProperty interface contains a list object for IAdapterValue.

By default IAdapterProeprty interface is exposed to AllJoyn as AllJoyn object at the following path and interface name:

*/{PropertyName}*

*{ExposedAdapterPrefix}.{AdapterName}.{PropertyName}*

It is possible to set user defined interface name for IAdapterProperty by setting InterfaceHint variable of IAdapterProperty interface.

Below are the properties of IAdapterProperty interface:

|  |  |  |
| --- | --- | --- |
| IAdapterProperty Properties | Description | Bridge Mapping |
| Attributes | List of IAdapterAttributes | Collection of AllJoyn Properties |
| InterfaceHint | User defined well-known name of the property | AllJoyn Interface name for the Property |
| Name | Name of the Property | AllJoyn Property |

Table 3: IAdapterProperty Properties

### IAdapterValue

IAdapterValue is the child object of IAdpterProeprty, exposed on AllJoyn bus with following object and interface names:

*/{PropertyName}/{ValueName}*

*{ExposedAdapterPrefix}.{AdapterName}.{PropertyName}.{ValueName}*

IAdapterValue has two properties, which need to be implemented.

|  |  |  |
| --- | --- | --- |
| IAdapterValue Propertie | Description | Bridge Mapping |
| Data | Value of property of device exposed to AllJoyn bus | AllJoyn Property |
| Name | Name of value | Name of an AllJoyn Property |

Table 4: IAdapterValue Properties

### IAdapterSingal

In bridge, IAdapterSignal represents an AllJoyn event that is emitted by the virtual AllJoyn. Change of Value signal is common in devices that emit an event to inform another device when a property of device changes. Adapter also contains signals for device arrival and device removal which are emitted when a device joins the AllJoyn network or leave it.

IAdapterSignal is emitted by a device with the signal name.

Like all other interfaces, IAdapterSignal also has following properties which need to be implemented.

|  |  |  |
| --- | --- | --- |
| IAdapterSignal  Property | Description | Bridge Mapping |
| Name | Name of Signal | AllJoyn Signal |
| Params | List of objects with changed value, or null for just a signal | Array of AllJoyn signal argument for this signal |

Table 5: IAdapterSignal Properties

Usage of the Adapter classes and overall DSB regarding thesis topic is explained in the chapter six.

# Tools and Configuration

The AllJoyn Framework provides the support for the variety of operating systems and programming languages. However, the current version of the AllJoyn Device System Bridge is available only for the Microsoft platform and language bindings (C#, C++) supported by the Microsoft platform. Developers need below tools to implement the AllJoyn DSB.

## Operating System

The AllJoyn Framework supports the Microsoft operating systems since its release time, but Microsoft provides native support for the AllJoyn Framework with the release of Windows 10. With the built-in AllJoyn API, it is now possible to run AllJoyn application on every Microsoft supported platforms like Tablets, phones and Xbox. To set up the development environment for the DSB implementation the latest version of any Windows 10 or preview build is required.

AllJoyn is the communication framework developed mainly for IoT devices. Microsoft has released a core version of the Windows for IoT devices called “Windows 10 IoT core” that supports the AllJoyn framework. With the Windows IoT Core, it is now possible to create the AllJoyn application for IoT device running Windows operating system. It is easy to deploy DSB implemented for the EnOcean protocol on Windows IoT Core. By default, every Windows IoT Core has the Z-Wave DSB deployed.

## Visual Studio

The AllJoyn DSB is only available as a visual studio template, therefore, the DSB implementation is only possible with visual studio IDE (Integrated development environment). Visual Studio 2015 version is required, but not express edition, because Express edition does not support the AllJoyn Framework.

## AllJoyn Device System Bridge Template27

The AllJoyn DSB template is a visual studio extension that facilitates developers to create a virtual representation of non-AllJoyn devices on the AllJoyn bus. The template provides a solution with the AllJoyn Bridge to bind AllJoyn and legacy protocols, Adapter as device controller and other projects to create DSB as headed or headless UWP application. The extension provides native (C++) and managed (C#) DSB templates.

The AllJoyn DSB Template can be downloaded from visual studio gallery, or via visual studio menu Tools -> Extension and Updates -> Online -> search “AllJoyn Device System Bridge Template”

The AllJoyn DSB comes with the below project:

#### HeadlessAdapterApp

The HeadlessAdapterApp initializes the Adapter and the Bridge objects and starts an AllJoyn DSB Universal App without user interface on Windows 10 IoT Core.

#### HeadedAdapterApp

The HeadedAdapterApp does the same job as HeadlessAdapterApp, but it creates an AllJoyn DSB Universal App with the user interface on Windows 10 or Windows 10 IoT Core.

#### AdapterLib

In the AdapterLib, developers write their code to connect to the physical devices and create their abstractions so that the Bridge can create their AllJoyn representation.

#### BridgeRT

The BridgeRT creates AllJoyn representation of the EO devices using abstractions created in AdapterLib and advertised them on the AllJoyn bus.

#### BackgroundHost, BackgroundHost.Headed, BackgroundHost.Headless

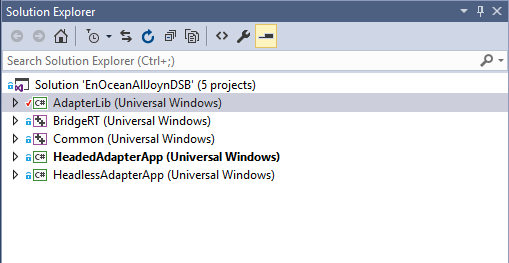
These background threads host and handle life cycle of Bridge and Adapter of the AllJoyn DSB.  


Figure 19: AllJoyn DSB template solution

## AllJoyn Explorer

The AllJoyn Explorer (“IoT Explorer for AllJoyn” since the release for Microsoft Windows store) is the Windows Universal Application developed by Microsoft, which works as the AllJoyn client and facilitates developers to discover and communicate with AllJoyn producers (devices) available in a proximity network. It lists all the services and their interfaces with all the members (signal, methods and properties) exposed to the AllJoyn bus. Users can call methods even with the parameters, receive signals (if enabled) and set or get properties of the device using AllJoyn Explorer.

The AllJoyn Explorer facilitates users to control their AllJoyn devices, but its core purpose is to be used as a developer tool to test and work with the AllJoyn devices. The AllJoyn Explorer is easy to install, search for “IoT Explorer for AllJoyn” in Microsoft Windows store.

## Higgns

One of the primary purposes of IoT devices is that they should be able to interact with each other, and this goal in the AllJoyn system is achievable through Higgns application. Higgns is another AllJoyn client to work with AllJoyn producers and it is developed by “Two Bulls”.

Higgns has powerful features and user-friendly interface. It is designed to be used by end users and currently available for Android and Fire TV. Main Feature of Higgns is stories creation, and users can create their stories. Stories are the logical condition which triggers an action when an event is triggered. For example, a story can be like this: when coffee is made then start toasting. Higgns is the most preferred AllJoyn client but during the AllJoyn DSB implementation for EnOcean protocol, it was found out that the AllJoyn DSB has a bug and because of this, Higgns does not list members of AllJoyn services created through via AllJoyn DSB. So, unfortunately, it is impossible to use Higgns with the current version of the AllJoyn DSB.

## JSON Framework

The REST-API of the EnOcean gateway provides the data in JSON format. Therefore, a JSON framework was needed to read data produced by the gateway. The AllJoyn DSB for the EnOcean protocol is implemented using C# template, so a JSON framework supporting C# was required, and it was easy to find one. Netwonsoft is the most popular and open source JSON framework that is used to read JSON data from the EO gateway.

# Design and Implementation of the EnOcean DSB

Previous chapters provide sufficient knowledge about the technologies used in the thesis topic, especially the most important part of the thesis topic, the AllJoyn Device System Bridge (DSB) and the Smart EnOcean Gateway. With the adequate knowledge about these technologies, it is now the right time to explain how these technologies work together. This chapter describes the integration of these components. First of all, a brief design comparison of the existing solution with the AllJoyn DSB needs to be discussed. Then, design and implementation of the DSB for the EnOcean protocol are discussed in detail.

## Possibilities

The AllJoyn framework provides interfaces to create AllJoyn enabled devices through service frameworks like the Home Appliances and Entertainment (HAE), Lighting Service Framework (LSF) and others. However, it lacks in providing standard interfaces for the devices like switches, sensors, and others. Allseen alliance has started a project to create such interfaces known as “Common Interface”, but no interface is defined yet. Without the standard interfaces for devices, it is hard to create AllJoyn devices which are interoperable with other AllJoyn devices and certify them as AllJoyn compatible devices. AllJoyn certifies only devices which are created using services or interfaces provided by AllJoyn.

The EnOcean technology has about 180 profiles defined, and none of them has a corresponding standard interface defined in the AllJoyn Framework. So, there is no exact representation of any EO profile in the AllJoyn system. However, using AllJoyn core interface definition model discussed in [2.2.3.4], it is possible to create virtual AllJoyn representation of EO devices, but AllJoyn core provides different ways like “Events and Action” and “Properties” to implement functionalities of devices. Without the standard interface, it is confusing which approach to follow while creating virtual AllJoyn representation of EO device.

DC created an own bridge to integrate EO standard with AllJoyn before the release of the AllJoyn DSB. So, now there are two prototypes available to integrate EO standard with the AllJoyn system. These bridges use two different mechanisms provided by AllJoyn core to create virtual AllJoyn representation of EnOcean devices.

### Java Bridge

This bridge is developed using Java API of AllJoyn, and it uses Events and Actions mechanism of AllJoyn to represent EO devices on AllJoyn side as shown in the figure. Java Bridge provides the basic connectivity and communication between AllJoyn and EO protocol, but it is not a standard bridge and does not cover all the EO profiles. However, Java Bridge provides features which are currently lacking in the AllJoyn DSB like running on Linux OS and virtual AllJoyn devices are discoverable through Higgns app.

### EnOcean AllJoyn DSB

This bridge is available in two AllJoyn bindings (C#, C) and developers have the choice to use any of them depend on their knowledge. The EnOcean DSB uses properties mechanism of AllJoyn to represent functionality of EO devices on AllJoyn side as shown in the figure. The main advantage the AllJoyn DSB has over the Java Bridge that it is a standard Bridge, and second it provides a dynamic way to cover all the EO profiles.

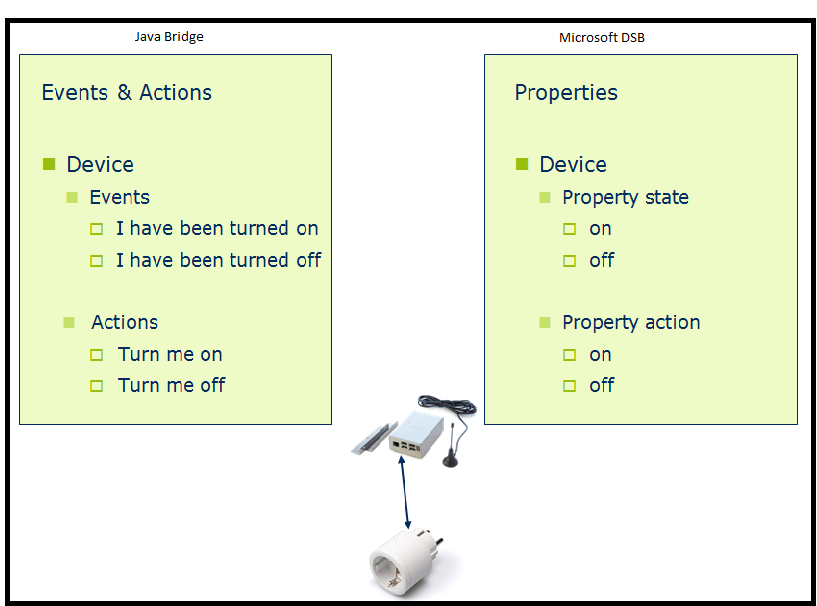


Figure 20: AllJoyn representation of EO device in different Bridges

## EnOcean AllJoyn DSB design

### Communication Architecture

The figure shows the architecture of the overall communication between different components involved in the implementation of the AllJoyn DSB for the EnOcean protocol. EnOcean devices and the gateway communicate using the EnOcean Radio Protocol (ERP). The Gateway provides messages (telegrams) received from EO device in the JSON format via REST API to the client of the gateway and converts JSON data received from the client into an ERP telegram for EnOcean device. In the DSB, the Adapter is the client to the gateway. The Gateway has secure REST API, so for the communication between the client and the gateway, a HTTP connection is required. The Bridge of the EO DSB gets the abstractions of EO devices from the Adapter and advertises them as virtual AllJoyn devices to the AllJoyn bus so that AllJoyn clients can discover them and users can interact with EO device via AllJoyn client.

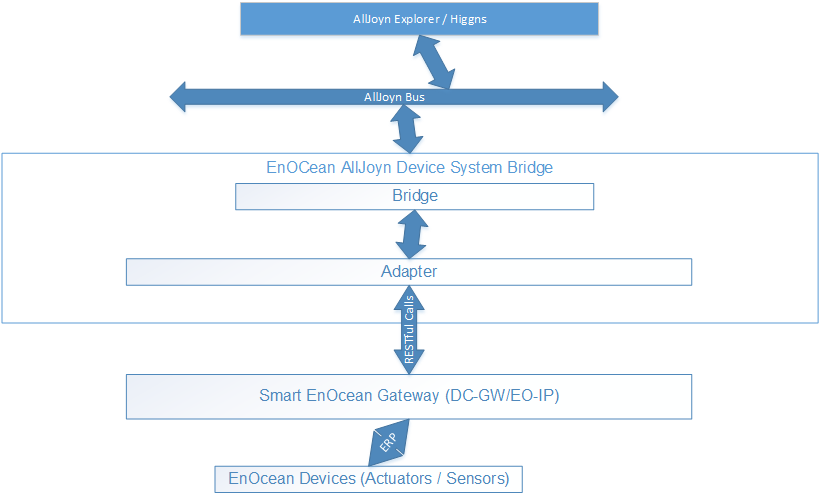


Figure 21: Communication Architecture of the AllJoyn System and EnOcean Protocol

### Use Case Diagram

Following use case diagram shows the actions a user can perform against the EO gateway through the EO DSB.

**Start Bridge**: Start bridge use case starts the EnOcean DSB and runs two more use cases in the background.

**REST API Authentication**: Authentication process runs in the background and makes a HTTP connection to the REST endpoint of the EO gateway.

**Discover Devices**: After completion of “REST API Authentication”, this use case is triggered by the bridge and creates DSB representation of all the devices known to the EO gateway.

**Add Device**: The user can learn in a new EO device to the gateway through the EO DSB.

**Remove Device**: The user can also remove an EO device from the gateway and the AllJoyn client using the EO DSB.

**Control Device**: This is the main feature of the EO DSB. This functionality permits users to change and see states of connected devices.



Figure 22: EnOcean DSB Use Case Diagram

### Class Diagram

The class diagram below shows the important EO DSB classes that are used during the implementation. This class diagram is the extended version of the AllJoyn DSB class diagram shown in figure 14. The EO DSB overrides the predefined classes in the AllJoyn DSB and creates some more classes to handle the Lighting Service Framework (LSF).

The Adapter class contains the logic to connect and control the data flow between the EO DSB and the EO gateway. The Adapter class utilizes other classes to create an abstraction of EO profiles that need to be bridged into the AllJoyn system.

The LightingServiceHandler and the Lamp class are implemented to handle the Lighting Service Framework (LSF) service for two EO profiles (D2-01-09 and A5-38-08).

The LightingServiceHandler class implements the ILSFHandler interface which handles LSF device calls from AllJoyn client.

Lamp class extends AdapterDevice class and initializes the LightingServiceHandler object.

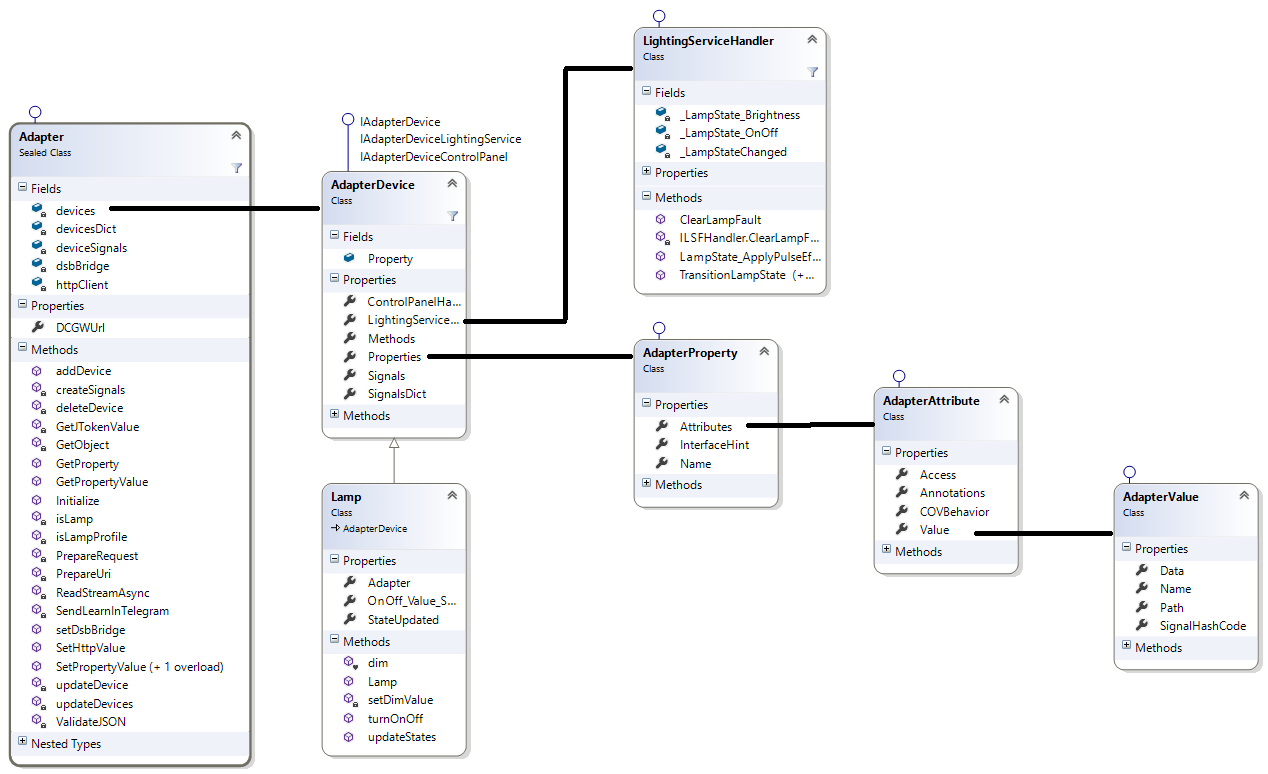


Figure 23: EnOcean DSB Class Diagram

### EnOcean Profile to DSB mapping

AdapterDevice, AdapterProperty, AdapterAttribute and AdapterValue classes are used to create the mapping of EO profile in the EO DSB. One or more EO Profiles implemented by an EO device are mapped to Adapter device. Each function group of the EO profile is translated to an Adapter property. Similarly, Adapter attribute object is created for each function, and key, values of the functions are mapped to Adapter value objects.

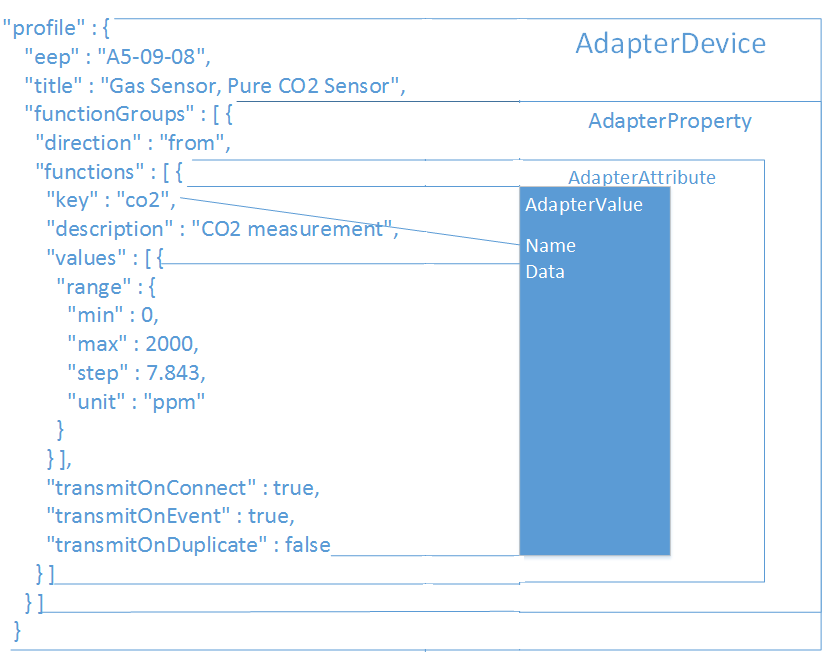


Figure 24: EnOcean Profile to EnOCean DSB Mapping

## EnOcean DSB implementation

When user starts the EO DSB with the URL of the targeted EO gateway. The HeadedAdapterApp creates the objects of the Adapter and the DsbBridge classes defined in the BridgeRT and AdapterLib respectively. These classes are responsible for data exchange between the BridgeRT and AdapterLib projects of the DSB. The Adapter object is initialized with the URL of the EO gateway and pass to DsbBridge constructor as a parameter. Finally, the HeadedAdapterApp starts the bridge by calling initialize method of DsbBridge object.

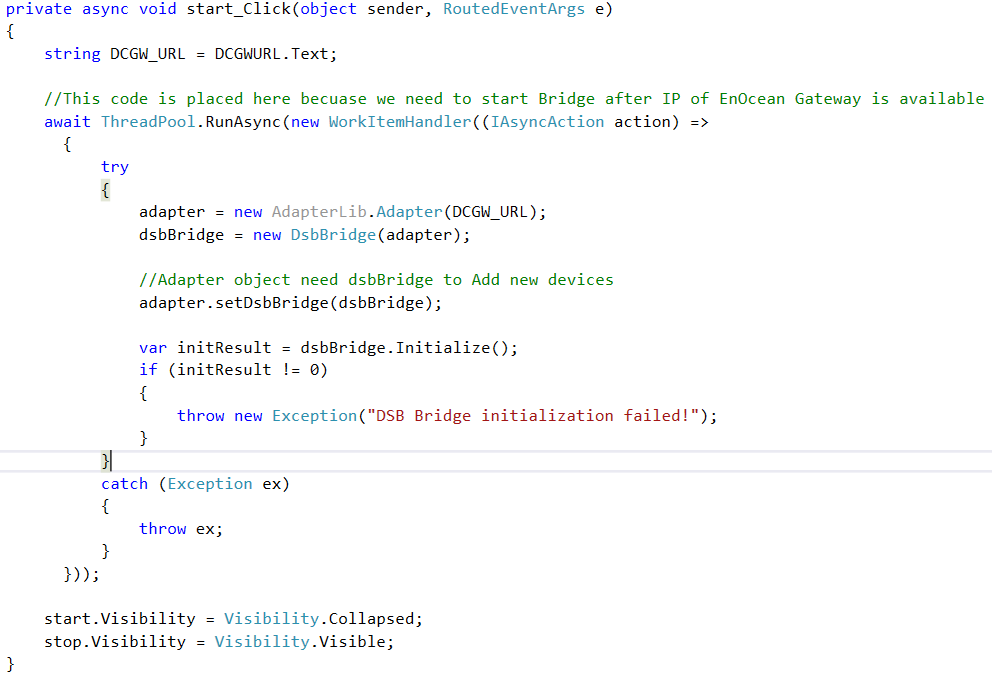


Figure 25: Start Bridge Method in HeadedAdapterApp

The DsbBridge invokes initialize method of Adapter object where Adapter establishes a HTTP connection to the gateway as shown below.



Figure 26: Adapter Initialize Method

The DsbBridge makes another call to EnumDevices method of Adapter to get devices which need to be advertised on the AllJoyn bus.

The Adapter makes a get request via PopuldateDeviceAsync method to the REST API of the gateway to know about EO devices learned into the gateway as shown in the figure 27. Then, the Adapter makes another call in AddDevice method (see Appendix) to get the profiles of every EO device and creates abstraction (AdapterDevice) of EO device from the profile definition. Finally, Adapter returns a list of AdapterDevices to the DsbBridge, and the Bridge creates virtual AllJoyn devices and advertises them on the AllJoyn bus.

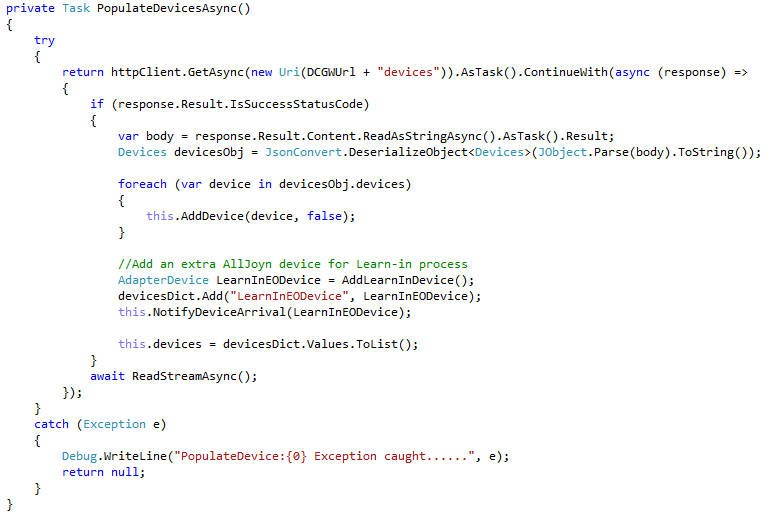


Figure 27: Getting Devices from the EO Gateway

Following sequence diagram shows the complete process of calls when the EO DSB starts.

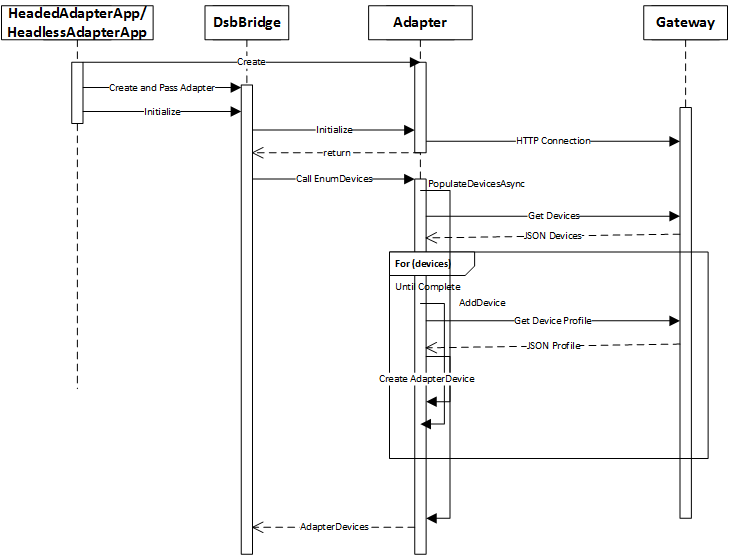


Figure 28: Start EnOCean DSB Sequence Diagram

### AllJoyn Client and EnOcean Gateway Communication

After creating the virtual AllJoyn devices of the physical EO devices, the EO DSB must listen for the changes in both types of devices and update them accordingly. The Bridge has predefined logic to receive change from AllJoyn device and send it to the Adapter so that the Adapter sends an update command to physical device via REST API as shown in the figure. Similarly, Adapter needs a way to capture the change of state from the physical device and send to the Bridge so that Bridge updates the corresponding virtual device. For this purpose, Adapter makes a non-ending asynchronous connection to the streaming API of the gateway and gets device events (status, learn-in, removing, and update) in real time.

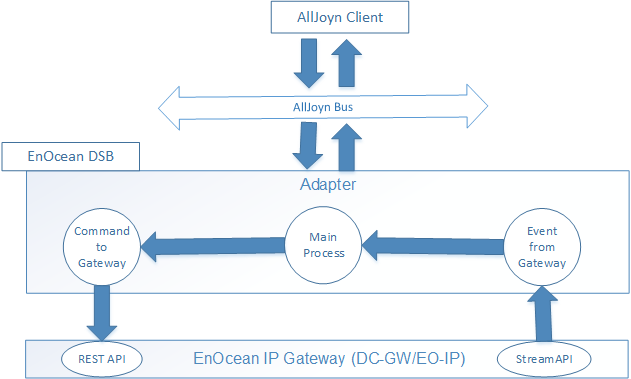


Figure 29: Handling commands by the EO DSB

As shown in the figure 29, EnOcean DSB listens continuously for the change of state of the devices from the AllJoyn client and the gateway to keep both sides of devices updated.

Following method reads the streaming API and updates the EO DSB depending on the content of the streaming API.



Figure 30: Reading Stream from EO Gateway

#### Handling the Action from AllJoyn Client

The action on virtual AllJoyn device is handled in sequence of steps as described below:

The user performs an action on the virtual AllJoyn device like turning on the lamp.

The Bridge in the DSB receives the action for the particular virtual device and calls the appropriate method in Adapter with required parameters.

First, the Adapter identifies the attribute of the device that needs to be updated, then prepares and sends update request (JSON-object) to the gateway.

The EO gateway process the request and sends an EO telegram to the corresponding EO device.

The EO device receives the telegram and updates its state, then sends a confirmation telegram back to the EO gateway

The EO gateway receives the acknowledgement telegram and updates the Streaming API.

The AllJoyn client updates its interface.

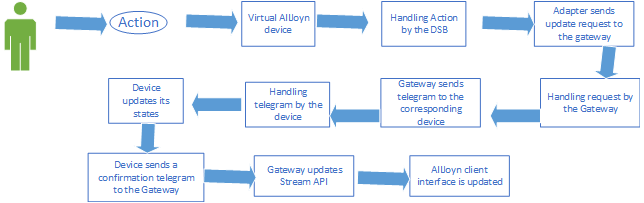


Figure 31: Process of Handling the Action from AllJoyn Client

#### Handling the telegram from the EO device

The telegram of the EO device is handled in a sequence of steps to update the corresponding virtual device as shown in the figure 32.

The user sends an EO telegram to the gateway by performing an action on the EO device.

The EO gateway receives the telegram containing new state of the device.

The EO gateway updates the streaming API with the telegram.

The Adapter receives the telegram from the streaming API.

The Adapter finds corresponding abstraction (AdapterDevice) of EO device and executes UpdateDevice (See Appendix) method to update the particular attribute of the AdapterDevice.

The Bridge updates corresponding virtual AllJoyn device of the abstraction (AdapterDevice) on the AllJoyn bus.

The AllJoyn client gets updated virtual AllJoyn device from the AllJoyn bus.

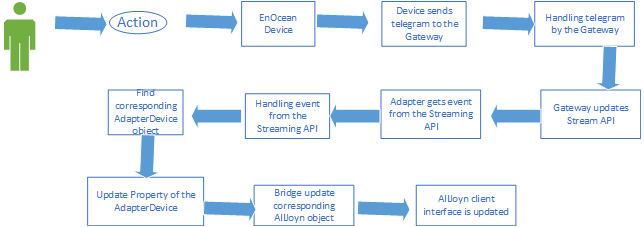


Figure 32: Process of handling the telegram from the EO device

### Processing of JSON objects between the Gateway and the DSB

The exchange of data between the EO gateway and the EnOcean DSB happens in the form of JSON format. The EO DSB has two different processes to handle the JSON object.

#### Sending JSON object from the EO DSB to the EO Gateway:

When the user changes the state of a virtual AllJoyn object, AllJoyn client makes a call to the correspond object in the EO DSB. The Bridge in the EO DSB handles the call and forward it to the Adapter with the required information for further processing. The Adapter calls the SetHttpValue function which further calls PerpareRequest where JSON object is created to update EO device. The figure shows the JSON representation of the JSON object created by PrepareRequest function.

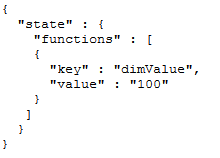


Figure 33: JSON Object to set Device State

Below code sends a request with JSON object to the EO gateway and handles response.

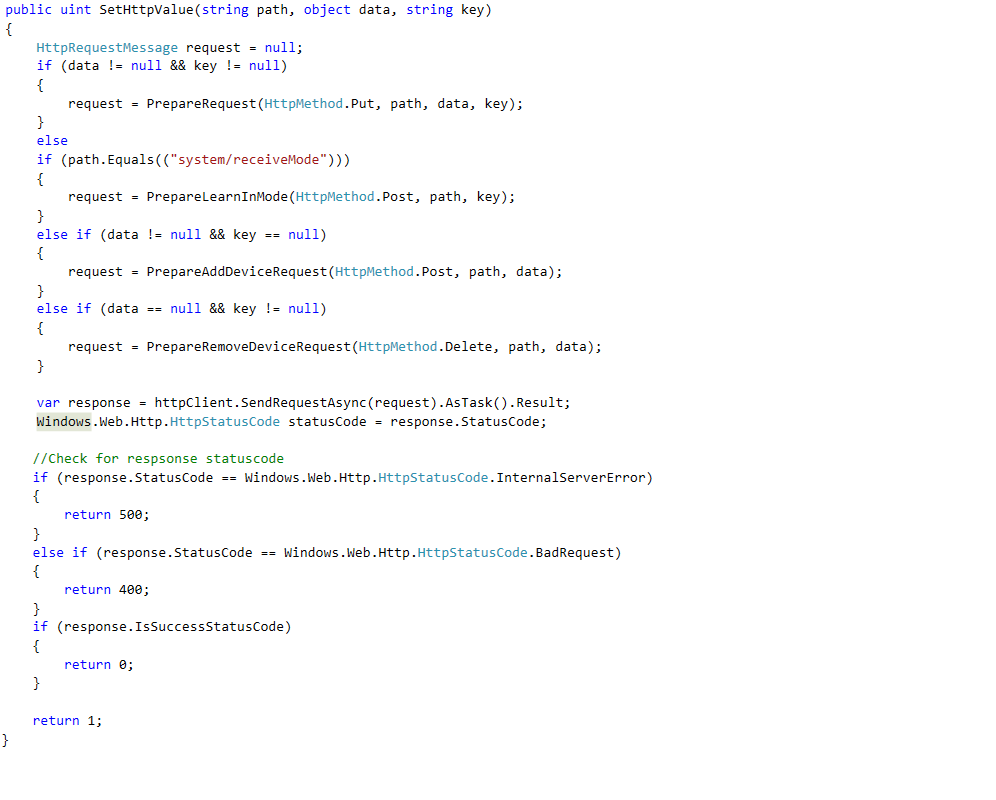


Figure 34: Sending Request to the EO Gateway

Below code creates the JSON object that is sent to the EO gateway.

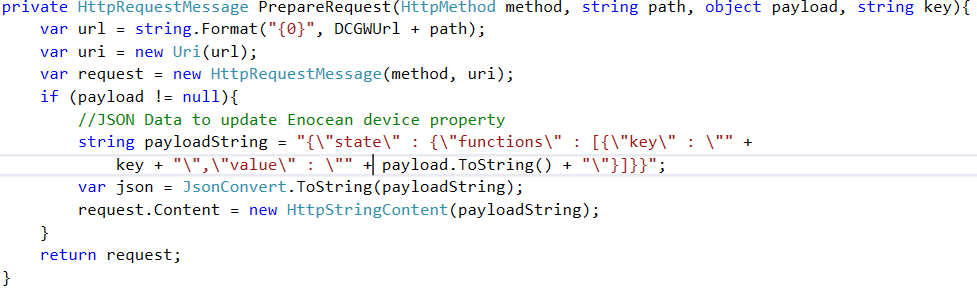


Figure 35: JSON Object Creation

#### JSON object from the EO Gateway to the EO DSB

The EO DSB receives every telegram from the EO gateway in the JSON format, these telegrams (JSON objects) are long and complicated to read. For this reason, first, JSON objects are converted to their corresponding C sharp objects so that it becomes easy to read JSON data. The JSON information is read from the C sharp objects and targeted service executed which in turn updates the AllJoyn client.

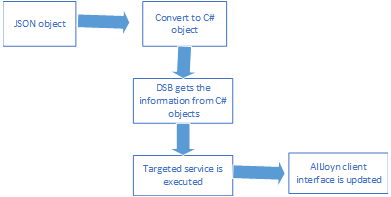


Figure 36: Handling of JSON object form the EO Gateway

### Learn-in Process

The user can learn-in a new EO device to the EO gateway through the EO DSB. The EO DSB creates an additional virtual AllJoyn device (LearnInEODevice) to start the learn-in process.

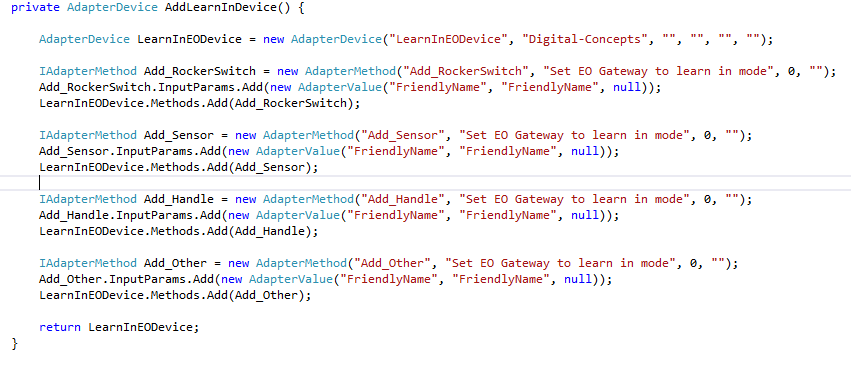


Figure 37: Creating AdapterDevice for Learn-in Process

The figure 41 shows the complete process to learn-in process for an EO device. Following steps are performed to learn-in an EO device.

1. The user invokes a method in LeranInEODevice object to start the learn-in process of a new EO device to the gateway. There are four methods available to learn-in an EO device, the user invokes a particular method depending on the device and enters the friendly name for the device.

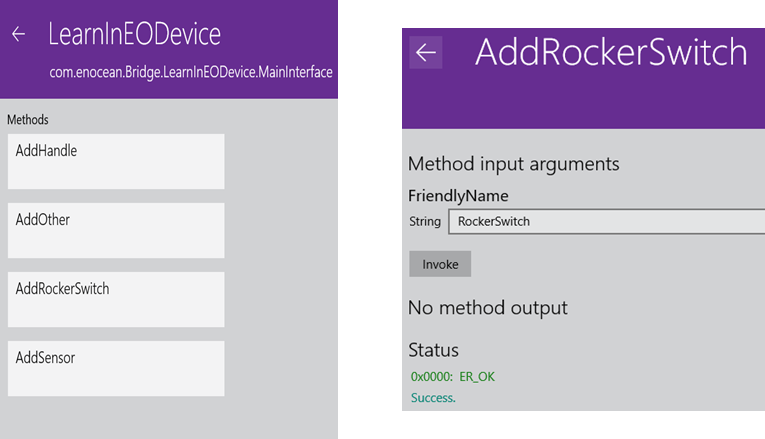


Figure 38: Learn-in EO Device Interface

1. The EO DSB receives user’s action, prepares and sends a HTTP request to the EO gateway to set it in learn-in mode.



Figure 39: User Action Handler in EO DSB

1. The user sends a learn-in telegram from the EO device.

The EO gateway receives, processes telegram and updates its streaming API.

1. The EO DSB reads the learn-in telegram from the streaming API, process it and sends back to the EO gateway.

During processing of learn-in telegram, EO DSB adds friendly name and changes operable value to “true” in the telegram. For some devices, DSB recreates telegram from the profile of the device.

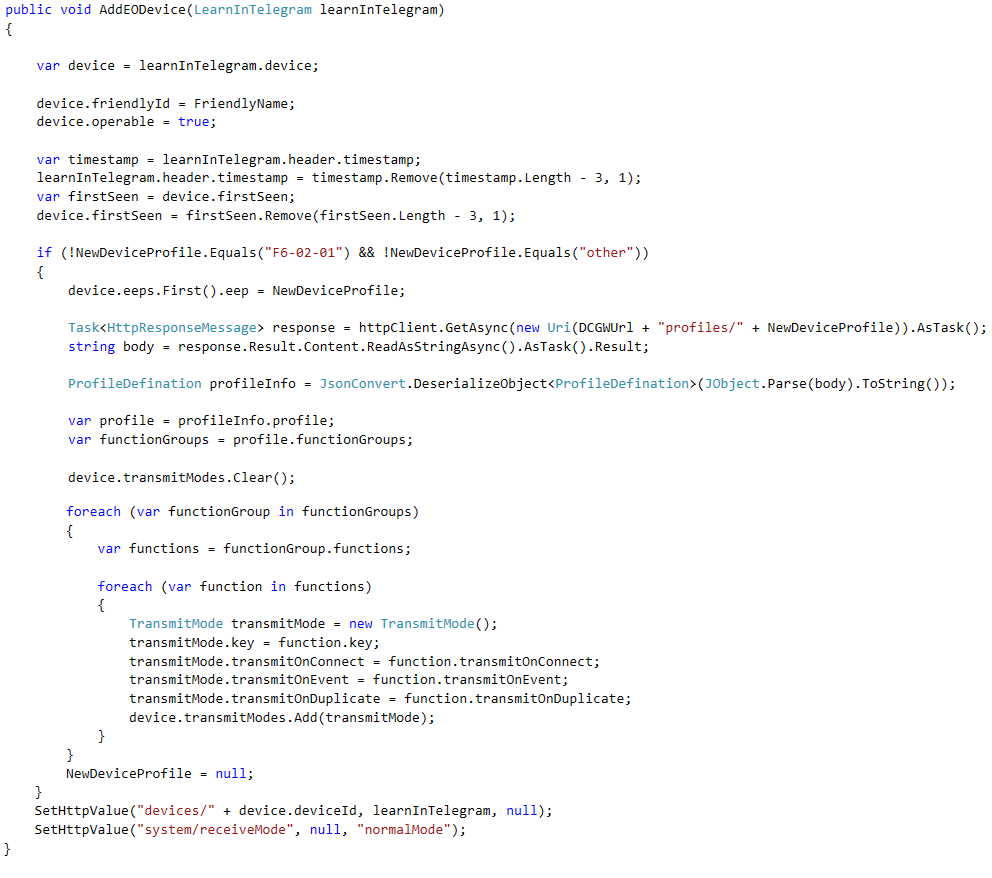


Figure 40: Create and Send Back Learn-in Telegram

1. The EO gateway generates an acknowledgement telegram on the streaming API after adding the EO device in the database.
2. The EO DSB gets the confirmation by reading acknowledgement telegram from streaming API and creates abstraction of the EO device and expose it to AllJoyn bus.



Figure 41: Learn-in Process

# Conclusion

The Internet of Things are growing in number, and there is no single standard protocol defined for the communication between IoT devices. With the number of different IoT protocols and it becomes necessary to integrate them together and with legacy protocols to increase the usability of IoT devices.

The purpose of the thesis topic was to integrate EnOcean Protocol with the AllJoyn system and to achieve this purpose a middleware was needed, and the AllJoyn DSB fulfils this purpose. But the AllJoyn DSB out of the box does not connect any protocol with the AllJoyn system. The AllJoyn DSB requires to create a connector (binding) for a specific protocol. In this case, the connector is designed and implemented for EnOcean Protocol using the AllJoyn DSB. The big challenge was understanding the architecture, implementation methods of the AllJoyn Framework, AllJoyn DSB and EnOcean gateway.

Thesis goal is achieved but during the implementation of the AllJoyn DSB for the EnOcean, it is revealed that right now the Microsoft’s AllJoyn DSB is not mature enough to be used at the commercial level because it limits the usability of the AllJoyn framework.

## Limitations

* Currently, the AllJoyn DSB supports only Microsoft Windows 10 and Windows 10 IoT Core operating systems. The AllJoyn DSB is a window universal application and uses Windows specific libraries so there is no workaround to deploy it on any other operating system.
* The AllJoyn system does not define standard interfaces for the home automation devices (switches, plugs, sensors, etc) and EnOcean built these kind of device. So, there is no standard way to represent EnOcean devices in the AllJoyn ecosystem.
* The current version of AllJoyn DSB is not stable, and some bugs are found during the implementation. Because of these bugs, the AllJoyn DSB is not compatible with Higgns.

## Future work

During the implementation of the AllJoyn DSB for EO protocol, some limitations have been found out. These restrictions need to be resolved, and more features are required to implement to increase the usability of the EnOcean DSB.

#### AllJoyn Home DSB

The Allseen Alliance is aware of the fact that the AllJoyn DSB support for only Windows OS restricts the usage of the AllJoyn DSB, so there is another version of DSB under development called AllJoyn Home DSB which will support other operating systems. Current EO DSB implementation can be used as the basis to implement Portable DSB.

#### Common Interfaces

Currently, AllJoyn does not provide standard interface for generic devices like switches, sensor, smart plug and EnOcean ecosystem contains these type of device. The Allseen Alliance has started a project for creating common interfaces for generic home automation products. Common Interface should be implemented for EnOcean profiles. With the help Common Interfaces, EnOcean devices will be translated into a standard AllJoyn interface, and they will become compatible with other AllJoyn devices and applications.

#### Update EnOcean DSB

The new version (v16.04) of the AllJoyn Framework resolves the majority of bugs in the AllJoyn DSB. However, the new version of the AllJoyn DSB template is not available yet. The EnOcean DSB implementation needs to be updated when updated AllJoyn DSB is available.

# Appendix

Code Samples of the EO DSB Implementation:

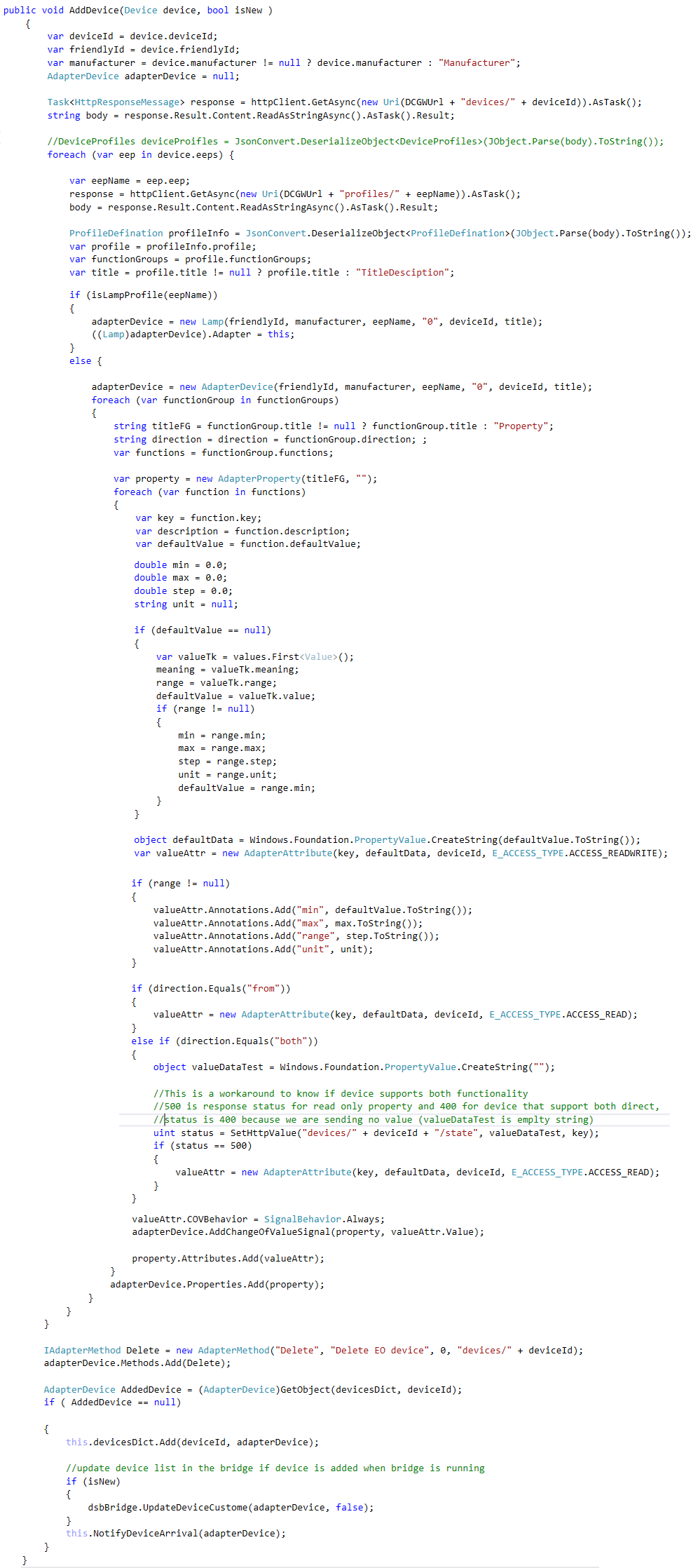


Figure 42: Creation of Adapter Device for EO Device

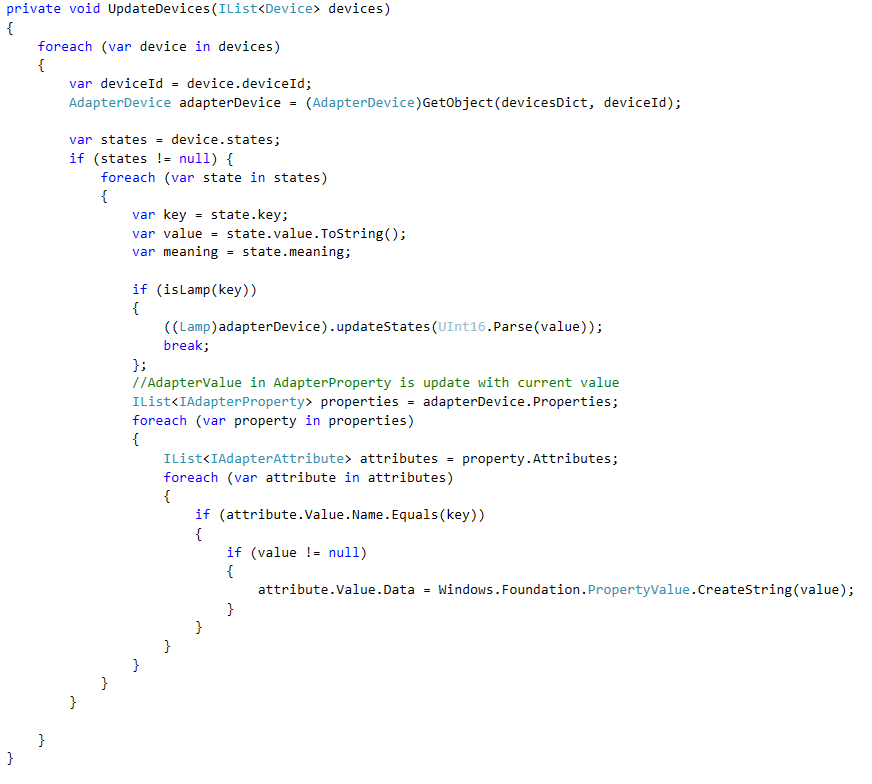


Figure 43: Update All EO Devices Abstractions in the EO DSB



Figure 44: Update Abstraction of an EO Device

Complete code for the EnOcean DSB project is available at the following github location:

https://github.com/Digital-Concepts/EnOceanAllJoynDSB

# Bibliography

1. Smart EnOcean Gateway - Company. Retrieved June 6, 2016, from <http://enocean-gateway.eu/index.php/en/company>
2. Energy harvesting - Wikipedia, the free encyclopedia. Retrieved June 6, 2016, from <https://en.wikipedia.org/wiki/Energy_harvesting>
3. EnOcean Alliance (June, 2015) EnOcean Equipment Profile 2.6.3 Specification. P7.
4. Smart EnOcean Gateway. Retrieved June 6, 2016, from http://enocean-gateway.eu/images/documents/Documentation/index.html?product.html
5. Framework | AllSeen Alliance. Retrieved June 6, 2016, from <https://allseenalliance.org/framework>
6. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. Retrieved June 6, 2016, from http://www.gartner.com/newsroom/id/3165317
7. Smart EnOcean Gateway. Retrieved June 6, 2016, from http://enocean-gateway.eu/images/documents/Documentation/index.html?funcionality\_purpose-of-the-ga.html
8. AllJoyn™ Standard Core | AllSeen Alliance. Retrieved June 6, 2016, from https://allseenalliance.org/framework/documentation/learn/core/standard-core
9. Architecture | AllSeen Alliance. Retrieved June 6, 2016, from https://allseenalliance.org/framework/documentation/learn/architecture
10. Architecture | AllSeen Alliance. Retrieved June 6, 2016, from <https://allseenalliance.org/framework/documentation/learn/architecture#alljoyn-app-code>
11. Architecture | AllSeen Alliance. Retrieved June 6, 2016, from https://allseenalliance.org/framework/documentation/learn/architecture#alljoyn-service-frameworks-libraries
12. Onboarding Service | AllSeen Alliance. Retrieved June 6, 2016, from https://allseenalliance.org/framework/documentation/learn/base-services/onboarding
13. Core Framework | AllSeen Alliance. Retrieved June 6, 2016, from <https://allseenalliance.org/framework/documentation/learn/core>
14. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?quickstart-guide.html>
15. Raspberry Pi 2 Model B. Retrieved June 6, 2016, from https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB\_-comp.jpeg
16. EnOcean Pi: Transforms Raspberry Pi into a Wire... | element14 Community. Retrieved June 6, 2016, from https://www.element14.com/community/docs/DOC-55169/l/enocean-pi-transforms-raspberry-pi-into-a-wireless-gateway
17. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?rest-resources.html>
18. Smart EnOcean Gateway. Retrieved June 6, 2016, from http://enocean-gateway.eu/images/documents/Documentation/index.html?api-function-calls.html
19. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?structure-of-communication2.html>
20. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?when-to-choose-what.html>
21. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?synchronous-communication-with.html>
22. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?synchronous-communication.html>
23. Smart EnOcean Gateway. Retrieved June 6, 2016, from <http://enocean-gateway.eu/images/documents/Documentation/index.html?asynchronous-communication.html>
24. gateway:dsb [wiki]. Retrieved June 7, 2016, from <https://wiki.allseenalliance.org/gateway/dsb>
25. Brian Vogelsang and Marc Alexander. "Connect Lighting Working Group". 2014. Presentation.
26. Alljoyn Device System Bridge - API Guide - Windows IoT. (n.d.). Retrieved Retrieved June 7, 2016, from <https://developer.microsoft.com/en-us/windows/iot/win10/alljoyndsbapiguide>
27. AllJoyn Device System Bridge Template extension. (n.d.). Retrieved from <https://visualstudiogallery.msdn.microsoft.com/aea0b437-ef07-42e3-bd88-8c7f906d5da8>