# File Type Specification

enocean alliance
Building Smarter Connectivity

## Device Description File

## V 2.0

San Ramon, CA, USA
Original Release 1.0, 2016

Executive Summary

***The Device Description File (DDF) defines a standardized XML format used to describe the technical features and configuration parameters of EnOcean devices. It allows software tools to automatically identify and configure devices based on their unique Product ID and EURID, supporting seamless integration into building and smart home systems.***

***The EnOcean Alliance provides a central GitHub-based DDF server where member companies can upload their own DDFs. An automatically updated index.xml file lists all available DDFs along with their product identifiers and file paths, enabling tools to retrieve the correct files dynamically.***

| Ver. | Editor | Change | Date |
|---|---|---|---|
| 0.1 | TM | First Draft | 28.11.2016 |
| 1.0. | TM | Updated for changes in ReCom v 1.10 | 24.04.2018 |
| 1.1. | TM/AP | Wording, simplification, extensions | 27.07.2021 |
| 2.0 | AP/TWG | Complete restructuring the document, adding new nodes. Issuing a new xsd. Adding Alliance DDF server usage. | June 2025 to January 2026 |
| | | | |



**DISCLAIMER**

parties (including any intellectual property rights, patent, copyright or trademark rights, or (2) any implied warranties of merchantability, fitness for a particular purpose, title or noninfringement.

In no event will the EnOcean Alliance be liable for any loss of profits, loss of business, los of use of data, interruption of business, or for any other direct, indirect, special or exemplary, incidental, punitive or consequential damages of any kind, in contract or in tort, in connection with this document or the information contained herein, even if advised of the possibility of such loss or damage. All Company, brand and product names may be trademarks that are the sole property of their respective owners.

The above notice and this paragraph must be included on all copies of this document that are made.

The EnOcean Alliance Device Description File Type Specification is available free of charge to companies, individuals and institutions for all non-commercial purposes (including educational research, technical evaluation and development of non-commercial tools or documentation.)

This specification includes intellectual property („IPR“) of the EnOcean Alliance and joint intellectual properties („joint IPR“) with contributing member companies. No part of this specification may be used in development of a product or service for sale without being a participant or promoter member of the EnOcean Alliance and/or joint owner of the appropriate joint IPR. EnOcean Alliance grants no rights to any third party IP, patents or trademarks.

These errata may not have been subjected to an Intellectual Property review, and as such, may contain undeclared Necessary Claims.

**EnOcean Alliance Inc.**
**2603 Camino Ramon, Suite 200**

**San Ramon, CA 94583**
USA
Graham Martin
Chairman & CEO EnOcean Alliance

# Table of Contents

# 1. Introduction

## 1.1. Scope and Purpose

This document is intended for device manufacturers creating DDF (Device Description Files) and Companies developing a DDF Parser. DDFs are part of the ReCom (Remote Commissioning Specification) and are used to specify the ReCom features of the Device. Additionally, the DDF is an electronic Datasheet and shall fully describe the capabilities of the device. This can be the used/supported EEP (EnOcean Equipment Profiles) and or GP (Generic Profiles), different other Device Capabilities, the Hardware and Firmware Revision and a written Description of the Device.

DDFs are part of the simplification of integrating and commissioning of EnOcean Devices inside a Building or Smart Home. The aim is that each Device has its own ProductID for identification. This Product ID and the EURID are then used as identifiers. A third-party Tool can then load the self-describing DDF for this Device. In a next step, an installer can plan/ install / commission this Device into his EnOcean network or even the full building network.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

## 1.2. Terms & Abbreviations

| Term / Abbr. | Description |
|---|---|
| AES | Advanced Encryption Standard |
| Base-ID | Alternative Sender Address, which can be used. The Base ID can be modified by a device and can be used to replace one module with another. Using Base ID as base for sending allows a device to use 128 different addresses for sending telegrams. In newer designs, it is always recommended to prefer addressed telegrams instead. |
| DDF | Device Description File |
| EEP | EnOcean Equipment Profile |
| ERP /<br>ERP 1 / ERP2 | EnOcean Radio Protocol version 1 or EnOcean Radio Protocol version 2.<br>If no number is used, then both versions are meant. |
| EURID | EnOcean Unique Radio Identifier – Unique Device Address of each module. This was called ChipID earlier. |
| GP | Generic Profiles |
| LSB / LSb | Least significant byte / Least significant bit |
| MSC | Manufacturer Specific Communication |

| | |
|---|---|
| RLC | Rolling Code |
| ReCom | Remote Commissioning |
| ReMan | Remote Management |

**Table 1.** Abbreviations used in this document.

## 1.3. References

[1] RFC 2119: Key words for use in RFCs to Indicate Requirement Levels
https://datatracker.ietf.org/doc/html/rfc2119
[2] EnOcean Alliance Specifications
https://www.enocean-alliance.org/specifications/

# 2. Device Description File

## 2.1. Motivation

With the increasing complexity of use cases for Building automation/ Smart Homes it is necessary that devices commissioning becomes easier. Due to the limitation of the Energy Harvesting Devices and the EnOcean protocol, it is not possible that devices are describing themselves completely over the air, compared to other protocols. The DDF is a suggested solution for this problem. Each EnOcean Device, which has a Product ID, will have a DDF describing all the necessary features of the Device, allowing easy onboarding and commissioning into an EnOcean (or building) network. ReCom or a Signal telegram can be used to get the ProductID of a device. Additionally, a QR Code can be scanned, or devices could also use NFC to store the product ID in an NDEF message. All the information inside the DDF allows an EnOcean network integrator to link, and configure all devices as needed.

The DDF uses an XML format for the description and an XSD is available for the validation of the XML.

## 2.2. DDF structure

### 2.2.1. Overview

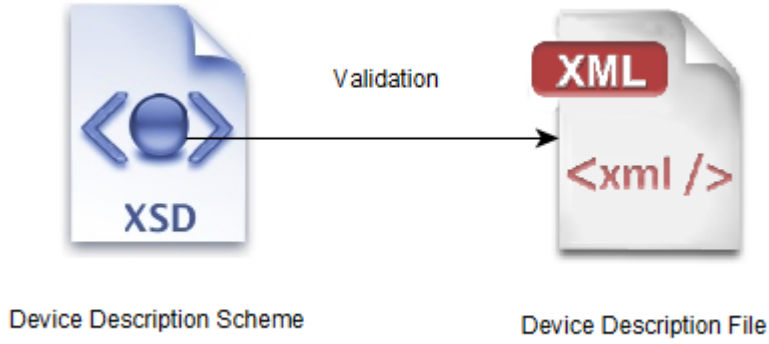



**Figure 1 XSD and XML**

A DDF is XML file which can be validated using the ReCom_Device_Description_Schema.xsd . The xsd contains the rules how the xml should look like and annotations for a self-describing help.

### 2.2.2. Structure Overview

The root node of the DDF is “Enocean_Devices” and contains a fixed attribute schemaVersion, to inform the parser about the schema used.

```
Enocean_Devices (1)
└─ Device (1…n)
   ├─ Name (1)
   ├─ Description (1)
   ├─ Firmware_Version (1)
   ├─ Hardware_Version (1)
   ├─ Frequency (1)
   ├─ ReComVersion (0…1)
   ├─ RemanVersion (0…1)
   ├─ TX (0…1)
   │  ├─ EURID (0…1)
   │  │  ├─ EEP (0…n)
   │  │  │  ├─ Rorg (1)
   │  │  │  ├─ Func (0…1)
   │  │  │  └─ Type (0…1)
   │  │  ├─ GP (0…n)
   │  │  │  ├─ (GP_TeachIN | GP_Channel)
   │  │  │  ├─ GP_TeachIN (0…1)
   │  │  │  └─ GP_Channel (0…n)
   │  │  │     ├─ Channel_Type (1)
   │  │  │     ├─ Value_Type (0…1)
   │  │  │     ├─ Signal_Type (1)
   │  │  │     ├─ EnumList (1)
   │  │  │     │  └─ Enum_Value (1…n)
   │  │  │     │     └─ Description (1)
   │  │  │     └─ Restrictions (0…1)
   │  │  │        ├─ (Max_Value+Min_Value | Resolution)
   │  │  │        ├─ Max_Value (0…1)
   │  │  │        ├─ Min_Value (0…1)
   │  │  │        └─ Resolution (0…1)
   │  │  │           ├─ Eng_Min (0…1)
   │  │  │           ├─ Eng_Max (0…1)
   │  │  │           ├─ Scale_Min (0…1)
   │  │  │           └─ Scale_Max (0…1)
   │  │  ├─ Signals (0…1)
   │  │  │  └─ Signal (0…n)
   │  │  └─ MSC (0…n)
   │  │     └─ Bitfield (1)
   │  │        ├─ Field (1…n)
   │  │        │  └─ (Scaled | Enum) (1) see RecomParameters
   │  │        └─ SubsequentPayload (0…n)
   │  │           └─ Field (1…n)
   │  │              └─ (Scaled | Enum) (1) see RecomParameters
   │  └─ BaseID (0…127) Limitation up to 127 will be removed for secure transmissions
   │     ├─ IDOffset (1)
   │     ├─ EEP (0…n) see EURID
   │     ├─ GP (0…n) see EURID
   │     ├─ Signals (0…1) see EURID
   │     └─ MSC (0…n) see see EURID
   ├─ RX (0…1)
   │  ├─ EEP (0…n) see TX
   │  ├─ GP (0…n) see TX
   │  └─ MSC (0…n) see TX
```

```
├─ LinkTable_MetaData (0…1)
│  ├─ InboundTable (0…1)
│  └─ OutboundTable (0…1)
├─ Device_Parameters (0…1)
│  ├─ Properties (0…1)
│  │  └─ Param (0…n)
│  │     ├─ Name (1)
│  │     ├─ Description (1)
│  │     ├─ Value (1)
│  │     └─ Unit (0…1)
│  ├─ RecomParameters (0…1)
│  │  └─ Param (0…n)
│  │     ├─ Description (1)
│  │     ├─ ParameterFlag (0…1)
│  │     ├─ BitField (0…1)
│  │     │  ├─ Length_In_Bytes (1)
│  │     │  └─ BitsPerChannel (0…1)
│  │     ├─ Key (0…1)
│  │     └─ (Scaled | Text | Enum | Private) (1)
│  │        Scaled:
│  │        ├─ Length_In_Bytes (1)
│  │        ├─ Default_Value (1)
│  │        ├─ Range (1)
│  │        │  ├─ Min (1)
│  │        │  └─ Max (1)
│  │        ├─ Scale (1)
│  │        │  ├─ Min (1)
│  │        │  └─ Max (1)
│  │        └─ Unit (1)
│  │        Text:
│  │        └─ Length_In_Bytes (1)
│  │        Enum:
│  │        ├─ Default_Value (1)
│  │        ├─ Length_In_Bytes (1)
│  │        ├─ EnumList (1)
│  │        │  ├─ Enum_Value (1…n)
│  │        │  │  ├─ ValueKey (0…1)
│  │        │  │  └─ Description (1)
│  │        │  └─ ScaledEnum_Value (0…n)
│  │        │     ├─ Description (1)
│  │        │     ├─ Scale (1)
│  │        │     │  ├─ Min (1)
│  │        │     │  └─ Max (1)
│  │        │     └─ Unit (1)
│  │        Private:
│  │        ├─ Default_Value (1)
│  │        └─ Length_In_Bytes (1)
│  └─ Applications (0…1)
│     └─ App (0…n)
│        ├─ Description (1)
│        └─ Parameters (0…1)
│           └─ Param (0…n)
│              └─ ConfigValue (1)
```

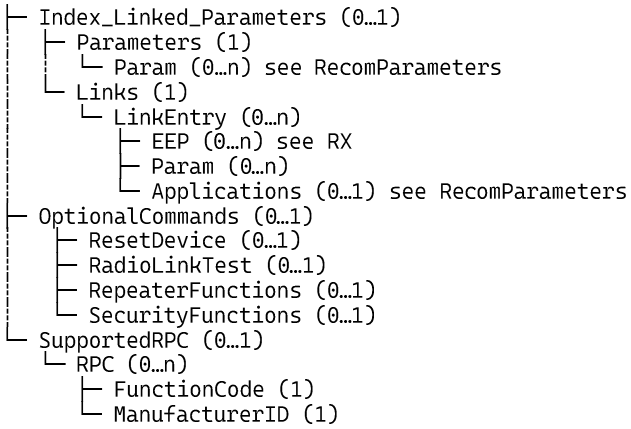



**Figure 2 Root node and Level Entries**

## 2.3. Reference

| Types | Definition |
|---|---|
| **Hex6Bytes** | xs:string, pattern: 0x[0-9A-Fa-f]{12} |
| **HexType** | xs:string, pattern: 0x[0-9A-Fa-f]+ (generic) |
| **HexType1** | xs:string, pattern: 0x[0-9A-Fa-f]{2} (1 byte hex) |
| **HexType2** | xs:string, pattern: 0x[0-9A-Fa-f]{4} (2 byte hex) |
| **HexType3** | xs:string, pattern: 0x[0-9A-Fa-f]{6} (3 byte hex) |
| **HexType4** | xs:string, pattern: 0x[0-9A-Fa-f]{8} (4 byte hex) |
| **BitType2** | xs:string, pattern: [01]{2} (exactly 2 bits) |
| **FreqType** | Enum {315, 868, 902, 928} |
| **AccessLevel** | xs:string, enum: {read, write, readWrite} |
| **UserLevel** | xs:string, enum: {admin, user} |
| **LinkEntryType** | xs:string, enum: {Addressed, None} |
| **LinkDirectionType** | xs:string, enum: {Inbound, Outbound, Both} |

**Table 2.** Types reference

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Enocean_Devices | schemaVersion | xs:string | 1 | Device | Version and location of schema |
| Device | Product_ID | Hex6Bytes | 1…n | Name, Description, Firmware_Version, Hardware_Version, Frequency, ReComVersion, RemanVersion, TX, RX, LinkTable_MetaData, Device_Parameters, Index_Linked_Parameters, OptionalCommands, SupportedRPC | 6-Byte Product ID of the device |
| Name | – | xs:string | 1 | – | Name of the device |
| Description | – | xs:string | 1 | – | Short description of the device |
| Firmware_Version | – | xs:string | 1 | – | Firmware version of the device |
| Hardware_Version | – | xs:string | 1 | – | Hardware version of the device |
| Frequency | – | FreqType | 1 | – | Frequency of the device |
| ReComVersion | – | xs:string | 0…1 | – | Supported ReCom-Version. |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| RemanVersion | – | xs:string | 0…1 | – | Supported Reman-Version. |
| TX | – | – | 0…1 | EURID, BaseID | Definition of profiles to transmit |
| RX | – | – | 0…1 | EEP, GP, MSC | Definition of profiles to receive |
| EURID | – | – | 0…1 | EEP, GP, Signals, MSC | Profiles, which are transmitted by EURID |
| BaseID | – | – | 0…127 | IDOffset, EEP, GP, Signals, MSC | Profiles, which are transmitted by BaseID. Limitation up to 127 is only for standard transmissions and will be removed for secure transmissions |
| IDOffset | - | xs:unsignedShort | 1 | - | Offset to the BaseID |
| EEP | LinkEntry | LinkEntryType | 0…n | Rorg, Func, Type | LinkEntry="None" if Broadcast, otherwise Addressed |
| Rorg | – | HexType1 | 1 | – | Hex value as string, e.g. 0xA5, or 0xD1 for MSC |
| Func | – | HexType1 | 0…1 | – | Hex value as string. Skipped if Rorg=MSC |
| Type | – | HexType1 | 0…1 | – | Hex value as string. Skipped if Rorg=MSC |
| GP | LinkEntry | LinkEntryType | 0…n | GP_TeachIN, GP_Channel | LinkEntry="None" if Broadcast, otherwise Addressed |
| GP_TeachIN | - | xs:string | 0…1 | - | Hex value as string; includes the encoded Teach-In-Message payload |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| GP_Channel | - | - | 0…n | Channel_Type, Value_Type, Signal_Type, EnumList, Restrictions | List of GP channels |
| Channel_Type | | BitType2 | 1 | - | Channel Type definition:<br><br>00 = Teach-in information Teach-in signals / flags,<br>01 = Data Complex bit values,<br>10 = Flag Single bit value,<br>11 = Enumeration Enumerated values, |
| Value_Type | - | BitType2 | 0…1 | - | Value Type definition:<br><br>00 = Reserved<br>01 = Current value<br>10 = Set point absolute<br>11 = Set point relative |
| Signal_Type | - | HexType2 | 1 | | The signal type classifies the origin of the transmitted value itself and its character (e.g. physical unit or field of use).<br>The signal types differ between the channel types. For a list of signal types consult the GP appendix. |
| EnumList | - | - | 1 | Enum_Value | List of enum values |
| Enum_Value | index | xs:unsignedShort | 1…n | Description | Value of the enum |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| ValueKey | - | xs:string | 0…1 | - | Unique identifier for the Enum index. Allows for an easier referral which is more informative than a plain number.<br>This will mostly be useful in the context of Tools for Commissioning or Configuration.<br><br>For compatibility reasons **ValueKey** should use a commonly picked format e.g. Camel Case. |
| Description | - | xs:string | 1 | - | Description of the parameter, which should be displayed. |
| Restrictions | - | - | 0…1 | Max_Value, Min_Value, Resolution | Definition of the parameters ranges |
| Max_Value | - | HexType | 0…1 | - | Maximum value of the parameter |
| Min_Value | - | HexType | 0…1 | - | Minimum value of the parameter |
| Resolution | - | HexType1 | 0…1 | Eng_Min, Eng_Max, Scale_Min, Scale_Max | See GP Specification |
| Eng_Min | - | xs:byte | 0…1 | - | Eng_Min represents the starting-point of the respective data inside the telegram coding. |
| Eng_Max | - | xs:byte | 0…1 | - | Eng_Max represents the end-point of the respective data inside the telegram coding. |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Scale_Min | - | HexType1 | 0…1 | - | The scale', represents the starting-point and the end-point of the respective real value<br>**Conversion: Valid Range ---> Scale**<br>$\text{Multiplier} = \frac{\text{Scale}_{MAX} - \text{Scale}_{MIN}}{\text{Range}_{MAX} - \text{Range}_{MIN}}$<br>$\text{Device value} = \text{Multiplier} * (\text{rawValue} - \text{Range}_{MIN}) + \text{Scale}_{MIN}$ |
| Scale_Max | - | HexType1 | 0…1 | - | See Scale_Min |
| Signals | – | – | 0…1 | Signal | List of signals transmitted |
| Signal | MID | xs:string | 0…n | – | MID of the signal, hex value as string. |
| MSC | – | – | 0…n | Bitfield | Definition of manufacturer specific telegrams<br>When specifying the RORG 0xD1 in the EEP tag, the corresponding manufacturer protocol must be specified. This is done within the MSC tag. |
| Bitfield | - | - | 1 | Field, SubsequentPayload | This node contains all the elements required to interpret the data signals. |
| Field | name | xs:string | 1…n | Enum, Scaled | Name of the bitfield |
| | offset | xs:byte | | | Offset of the bitfield inside the telegram |
| | bitSize | xs:byte | | | Number of bits used for this field |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| SubsequentPayload | - | - | 0…n | Field | If a device requires flexible data content because the multitude of functions cannot be mapped within a single telegram, the SubsequentPayload Node is used. This behavior is comparable to the CMD (command) fields in VLD telegrams. See Appendix. |
| LinkTable_MetaData | - | - | 0…1 | InboundTable, OutboundTable | Lists if an inbound and/or outbound table is supported. And if it's supported how many entries it can have. |
| InboundTable | maxLength | xs:decimal | 1 | - | Maximal number of entries |
| | GPSupported | xs:boolean | | | If yes, the table supports GP entry types |
| | RemoteTeachSupported | xs:boolean | | | If yes, the device can set into remote teach in mode, and store entries in the link table for received teach ins. |
| | SecuritySupported | xs:boolean | | | If yes, the table can store Enhanced Security Information. |
| OutboundTable | maxLength | xs:decimal | 1 | - | Maximal number of entries |
| | GPSupported | xs:boolean | | | If yes, the table supports GP entry types |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| | RemoteTeachSupported | xs:boolean | | | If yes, the device can set into remote teach in mode, and store entries in the link table for received teach ins. |
| | SecuritySupported | xs:boolean | | | If yes, the table can store Enhanced Security Information. |
| Device_Parameters | – | – | 0…1 | Properties, RecomParameters, Applications | Describes Parameters of the device |
| Properties | – | – | 0…1 | Param | Static parameters of the device |
| Param | – | – | 0…n | Name, Description, Value, Unit | Description of one static parameter |
| Name | – | xs:string | 1 | – | Name of the parameter which should be displayed |
| Description | – | xs:string | 1 | – | Description of the parameter which should be displayed |
| Value | – | xs:string | 1 | – | Value of this parameter |
| Unit | – | xs:string | 0…1 | – | Unit of this parameter optional |
| RecomParameters | - | - | 0…1 | Param | Describes parameters, which can be read or set via ReCom. Some parameters could be read or write only. |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Param | index | xs:unsignedShort | 0…n | Description, Key, ParameterFlag, BitField, (Scaled \| Text \| Enum \| Private) | Index of the parameter |
| | accessLevel | AccessLevel | | | Describes if a parameter can be read/written or both via ReCom. Allowed values: read, write, readWrite |
| | recommendedUserLevel | UserLevel | | | Recommended level which sort of user can access the parameter. Allowed values: admin, user |
| | RPCWriteStartAddress | HexType4 | | | Optional attribute if using RPC write, this would be the start address of the parameter. |
| | RPCWriteLength | HexType4 | | | Optional attribute if using RPC write, this would be the length of the parameter |
| Description | – | xs:string | 1 | – | Description of the parameter which should be displayed |
| Key | - | xs:string | 0…1 | - | Unique identifier for the Parameter index. Allows for an easier referral which is more informative than a plain number.<br>This will mostly be useful in the context of Tools for Commissioning or Configuration.<br><br>For compatibility reasons **KEY** should use a commonly picked format e.g. Camel Case. |

| | | | | | |
|---|---|---|---|---|---|
| ParameterFlag | - | xs:string | 0…1 | - | Additional flag for altering interpretation of the transmitted/received Data. Multiple Flags are available and are as follows:<br>• LinkTableInboundBoolean<br>This Flag defines that each bit is interpreted as a Boolean value (True/False) and mapped to a Set of Endpoints for Configuration. Each bit corresponds to the index of an LinkTableEntry, and allows a toggle for specific device logic (e.g. inverse the direction for motor control)<br>• LinkTableInboundChannel<br>This Flag behaves like LinkTableInboundBoolean with the addition of a dynamic channel Range. One Example are multi-channel bridges, where a sensor can be linked to either all or only a singular channel.<br>• LinkTableInboundEnum<br>An extended Version/Combination of LinkTableInboundChannel with the Enum Node. Each value has an assigned enumeration in the range of BitsPerChannel**.** This allows the transport additional information, should the value not correspond to a |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| | | | | | number (e.g. channel) but rather different functional modes.<br><br>Further flags may be added in the future.<br><br>The functionality is similar to Linked Parameters but in a smaller scope and predefined. |
| BitField | - | - | 0…1 | Length_In_Bytes, BitsPerChannel | BitField defines the Boundaries |
| Length_In_Bytes | – | xs:unsignedInt | 1 | – | Length of the parameter in Bytes |
| BitsPerChannel | - | xs:unsignedInt | 0…1 | - | The Field **BitsPerChannel** defies the number of bits used for each Endpoint. This allows for a small range of values in contrast to the two values for a Boolean. The number of **BitsPerChannel** must always fit evenly into the **Length_In_Bytes**, to prevent undefined data. |
| Scaled | – | – | 0…1 | Length_In_Bytes, Default_Value, Range, Scale, Unit | A scaled item is a parameter, which represent a rational number and is scaled to a rawValue. |
| Length_In_Bytes | – | xs:unsignedInt | 1 | – | Length of the parameter in Bytes |
| Default_Value | – | xs:decimal | 1 | – | Default value of the parameter |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Range | – | - | 1 | Min, Max | Defines the range of the parameter |
| Min | – | xs:nonNegativeInteger | 1 | – | Represents the starting-point of the respective data inside the telegram coding. |
| Max | – | xs:nonNegativeInteger | 1 | – | Represents the end-point of the respective data inside the telegram coding. |
| Scale | - | - | 1 | Min, Max | Defines the scale of the parameter |
| Min | – | xs:decimal | 1 | – | Represents the starting-point and the end-point of the respective real value<br>**Conversion: Valid Range ---> Scale**<br>$\text{Multiplier} = \frac{\text{Scale}_{MAX} - \text{Scale}_{MIN}}{\text{Range}_{MAX} - \text{Range}_{MIN}}$<br>$\text{Device value} = \text{Multiplier} * ( \text{rawValue} - \text{Range}_{MIN} ) + \text{Scale}_{MIN}$ |
| Max | – | xs:decimal | 1 | – | See Min |
| Unit | – | xs:string | 1 | – | Unit of the parameter |
| Text | – | – | 0…1 | Length_In_Bytes | Text is in Description UTF-8 coded |
| Length_In_Bytes | - | xs:unsignedInt | 1 | - | Length of the text parameter in Bytes |
| Enum | – | – | 0…1 | Default_Value, Length_In_Bytes, EnumList | Definition of an enum parameter |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Default_Value | - | xs:unsignedInt | 1 | - | Default value of the enum parameter |
| Length_In_Bytes | - | xs:unsignedInt | 1 | - | Length of the enum parameter in Bytes |
| EnumList | – | – | 1 | Enum_Value, ScaledEnum_Value | List of the enum values |
| Enum_Value | index | xs:unsignedInt | 1…n | Description | Index of the enum value |
| Description | – | xs:string | 1 | – | Description of the enum parameter which should be displayed |
| ScaledEnum_Value | indexMin | xs:unsignedInt | 0…n | Description, Scale, Unit | Minimum range of the enum value |
| | indexMax | xs:unsignedInt | | | Maximum range of the enum value |
| Description | – | xs:string | 1 | – | Description of the scaled enum parameter which should be displayed |
| Scale | - | - | 1 | Min, Max | Defines the scale of the scaled enum parameter |
| Min | – | xs:decimal | 1 | – | Represents the starting-point and the end-point of the respective real value<br>**Conversion: Valid Range ---> Scale**<br>$\text{Multiplier} = \frac{\text{Scale}_{MAX} - \text{Scale}_{MIN}}{\text{Range}_{MAX} - \text{Range}_{MIN}}$<br>$\text{Device value} = \text{Multiplier} * ( \text{rawValue} - \text{Range}_{MIN} ) + \text{Scale}_{MIN}$ |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Max | – | xs:decimal | 1 | – | See Min |
| Unit | – | xs:string | 1 | – | Unit of the scaled enum parameter |
| Private | - | - | 0…1 | Default_Value, Length_In_Bytes | The parameter is either not described or it is reserved |
| Default_Value | - | HexType | 1 | - | Default value of the parameter |
| Length_In_Bytes | - | xs:unsignedInt | 1 | - | Length in bytes of the parameter |
| Applications | - | - | 0…1 | App | Defines parameter values to be used for this application |
| App | - | - | 0…n | Description, Parameters | Defines each application |
| Description | - | - | 1 | - | Description of the parameter |
| Parameters | - | - | 0…1 | - | |
| Param | index | xs:unsignedShort | 0…n | ConfigValue | Parameter which will be modified defined by the index |
| ConfigValue | - | xs:string | 1 | - | Raw value of the value in a hex format |
| Index_Linked_Parameters | – | – | 0…1 | Parameters, Links | Defines parameters which can be changed depending on an EEP |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| Parameters | - | - | 1 | Param | See structure in RecomParameters |
| Links | – | – | 1 | LinkEntry | Describes the link between an inbound or outbound EEP and the used params |
| LinkEntry | Direction | LinkDirectionType | 0…n | EEP, Param, Applications | Describes a link between one or more EEPs using the same linked parameter . Allowed values for Direction: Inbound, Outbound, Both |
| EEP | LinkEntry | | 0…n | Rorg, Func, Type | See RX/EURID |
| Param | index | xs:unsignedInt | 0…n | – | See RecomParameters |
| Applications | – | – | 0…1 | App | See RecomParameters |
| OptionalCommands | – | – | 0…1 | ResetDevice, RadioLinkTest, RepeaterFunctions, SecurityFunctions | Optional ReCom commands supported |
| ResetDevice | – | – | 0…1 | – | If this node occurs the commands are supported |
| RadioLinkTest | – | – | 0…1 | – | If this node occurs the commands are supported |
| RepeaterFunctions | – | – | 0…1 | – | If this node occurs the commands are supported |
| SecurityFunctions | NumberOfKeysSupported | xs:decimal | 0…1 | – | If this node occurs the commands are supported. |

| Element | Attribute | Type | Card. | Childs | Description |
|---|---|---|---|---|---|
| SupportedRPC | – | – | 0…1 | RPC | List of supported RPCs. |
| RPC | – | – | 0…n | FunctionCode, ManufacturerID | RPC which is supported |
| FunctionCode | – | HexType3 | 1 | – | Function code of the RPC |
| ManufacturerID | – | HexType3 | 1 | – | ManufacturerID of the RPC |

**Table 3.** Nodes reference

### 2.3.1. SubsequentPayload Node Explanation

If a device requires flexible MSC data content because the multitude of functions cannot be mapped within a single telegram, the SubsequentPayload Node is used. Its content consists of individual field nodes that serve to describe the data area. The data content contained therein always refers to a previously defined type. This type determines how the subsequent data is to be interpreted.

This behavior is comparable to the CMD (command) fields in VLD telegrams. Example:

```
<Field name="TelegramType" offset="16" bitSize="8">
  <Enum>
    <Default_Value>1</Default_Value>
    <Length_In_Bytes>1</Length_In_Bytes>
    <EnumList>
      <Enum_Value index="1"><Description>PDI</Description></Enum_Value>
      <Enum_Value index="2"><Description>PDA</Description></Enum_Value>
      <Enum_Value index="3"><Description>PDCO</Description></Enum_Value>
      <Enum_Value index="4"><Description>PDS</Description></Enum_Value>
      <Enum_Value index="5"><Description>PDCID</Description></Enum_Value>
    </EnumList>
  </Enum>
</Field>
```

After successfully determining the received type, the subsequent data can be interpreted accordingly.
To do this, a SubsequentPayload node is created that references the corresponding type.
Within the SubsequentPayload tag, at least one Field node must be defined for data interpretation.

Example SubsequentPayload with type PDA:

```
<SubsequentPayload id="PDA">
  <Field name="divisor" offset="30" bitSize="2">
    <Enum>
      <Default_Value>0</Default_Value>
      <Length_In_Bytes>1</Length_In_Bytes>
      <EnumList>
        <Enum_Value index="0"><Description>x/1</Description></Enum_Value>
        <Enum_Value index="1"><Description>x/10</Description></Enum_Value>
        <Enum_Value index="2"><Description>x/100</Description></Enum_Value>
        <Enum_Value index="3"><Description>x/1000</Description></Enum_Value>
      </EnumList>
    </Enum>
  </Field>

  <Field name="current value" offset="32" bitSize="32">
    <Scaled>
      <Length_In_Bytes>4</Length_In_Bytes>
      <Default_Value>0</Default_Value>
      <Range>
        <Min>0</Min>
        <Max>4294967295</Max>
      </Range>
      <Scale>
        <Min>0</Min>
        <Max>4294967295</Max>
      </Scale>
      <Unit>n/a</Unit>
    </Scaled>
  </Field>
</SubsequentPayload>
```

If a defined interpretation of a data telegram is required again at another location, it can be referenced using SubsequentPayload.
Example with referenced data definition:

```
<SubsequentPayload id="PDCO" refId="PDA"/>
```

# 3. EnOcean Alliance DDF-Server

The purpose of the DDF server is to provide a centralized, collaborative repository for all device descriptions (DDFs) relevant to the EnOcean ecosystem. It offers versioning, transparency, and ease of use — and supports our shared goal of improved interoperability.

## 3.1. How to upload DDFs to the DDF server

***Get access to the DDF contributor team***

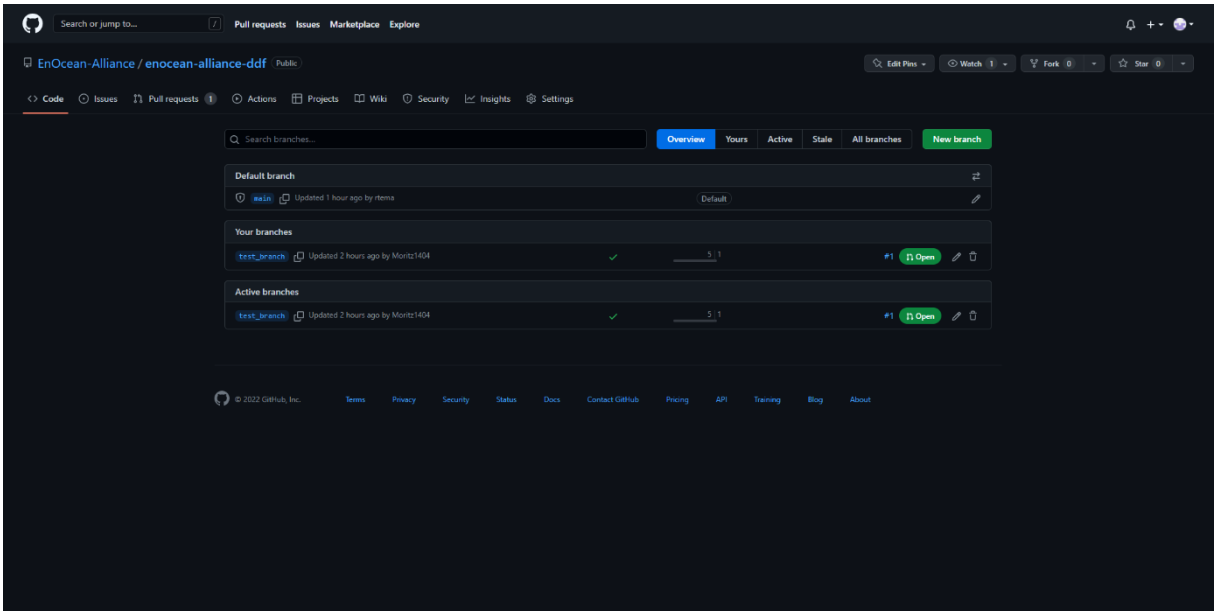
Just E-Mail me your Github username and I will create your company directory and give you access to it to be able to upload your DDFs there.

***Create a workbench (Create a new branch)***

To add/ edit products we need to create a new branch, to work inside.

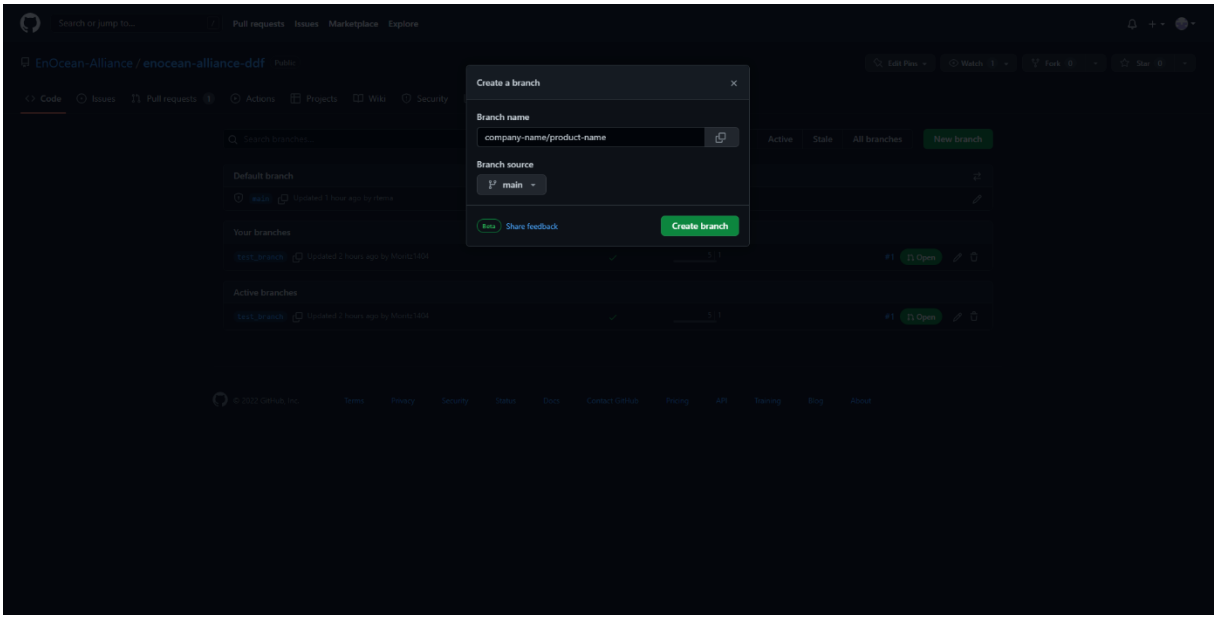
To do so, we "copy" the ***main*** (current state) branch.

1. Go to ***branches*** inside the ***enocean-alliance-ddf*** repository (https://github.com/EnOcean-Alliance/enocean-alliance-ddf/branches).
2. On the upper right sight click ***New Branch***.



3. Now enter a branch name. We recommend to use this formula: ***company-name/product-name***.

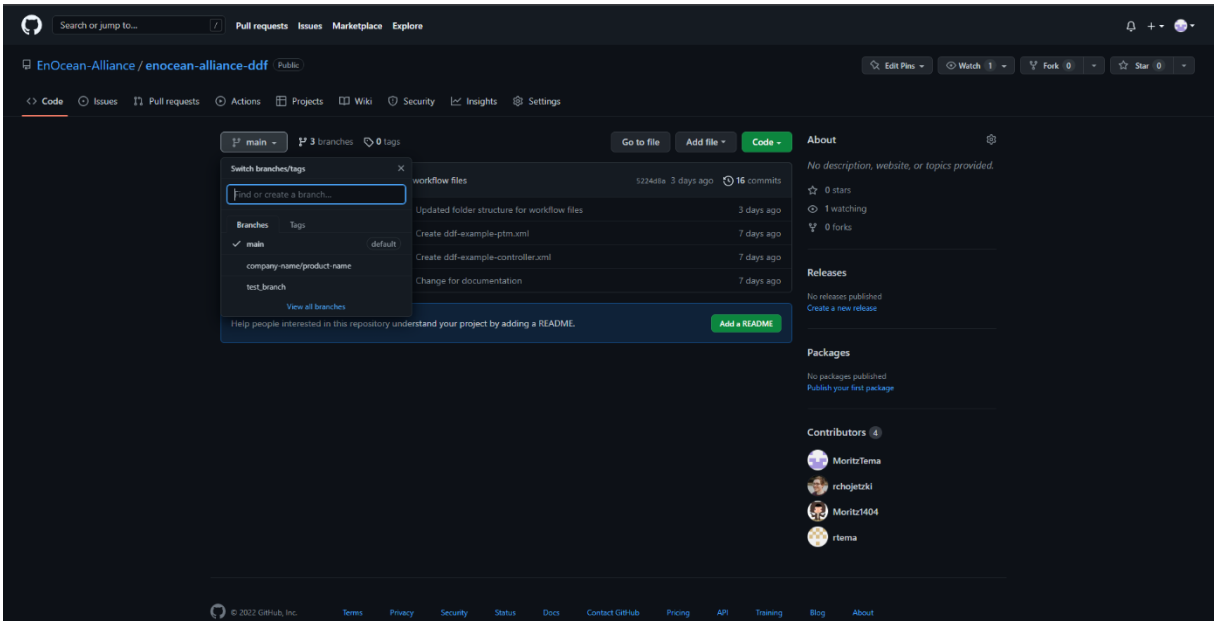
   Always choose ***main*** as ***Branch Source***!




## ***Add your DDFs (Commit files into your branch)***

Next, we switch to our branch that we just created.

1. To do that, click on the ***main*** button on the upper left side. A drop-down menu will open.
2. Then select your branch. In our case ***company-name/product-name***.



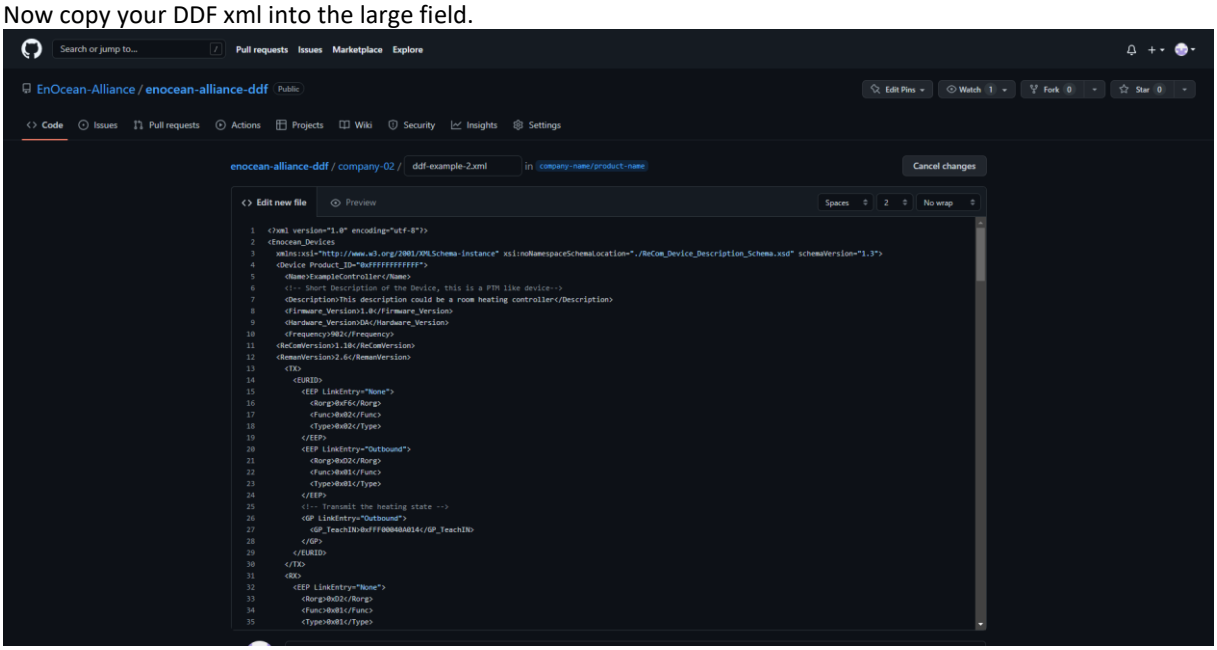

Now we are in our Branch.

3. Now go to your company folder.
4. Click ***Add file*** on the upper right side. Now you can choose whether you want to upload a file or create one directly in the browser.

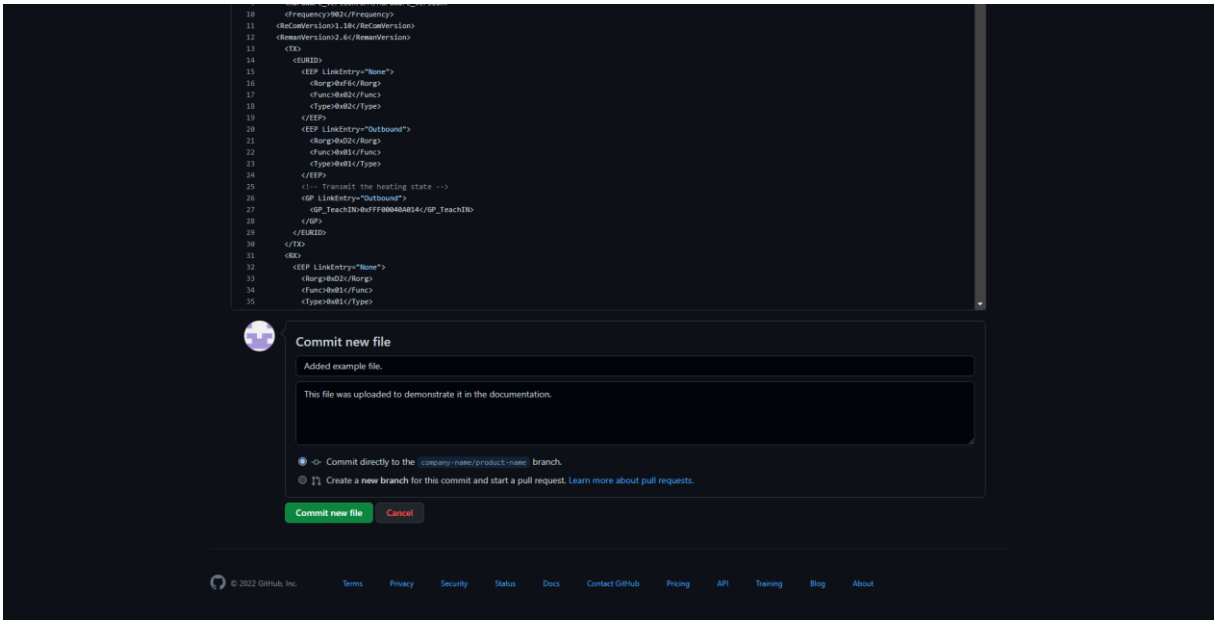
   In this example, we create one in the browser.

   If you want to upload/edit DDFs via an IDE/terminal, please refer to the official GitHub documentation.

5. In the upper area we can specify the file name. Note: only small letters, numbers and hyphens are allowed. The file must end with .xml.
6. Now copy your DDF xml into the large field.




7. In the ***Commit new file*** field, a title and a description are specified. Here should be described what was done.
8. Make sure that ***Commit directly to the <branchName> branch*** is selected.
9. Now click ***Commit new file***.



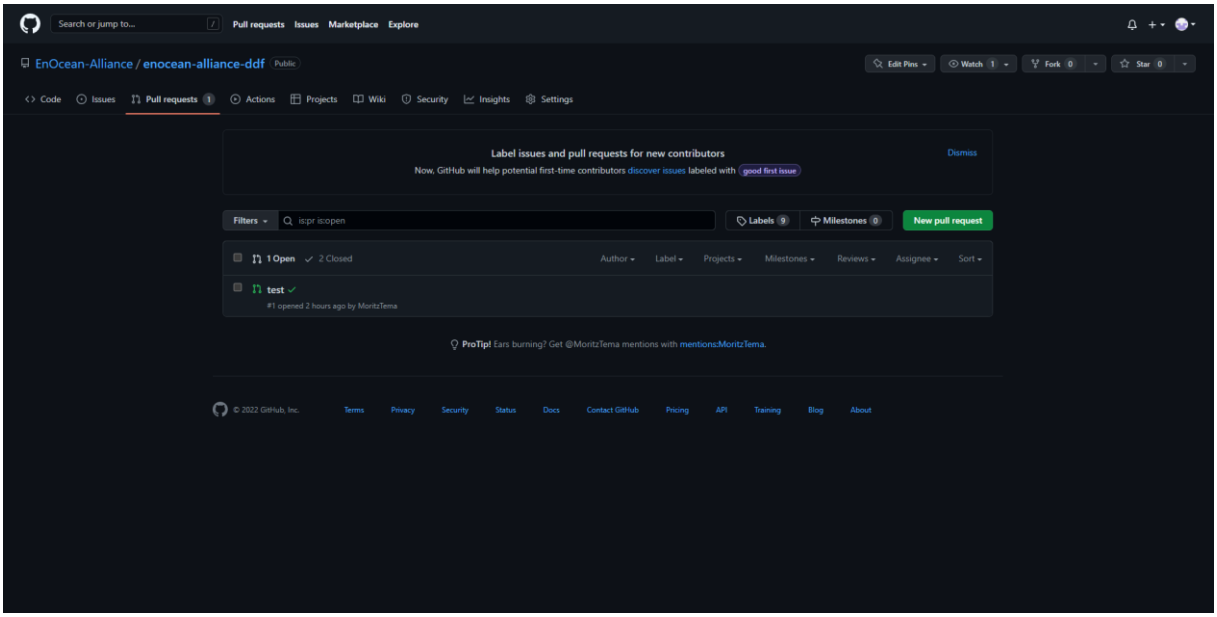

**Note: Please note that everything you upload is already publicly viewable.**

## ***Publish your changes (Create a pull request)***

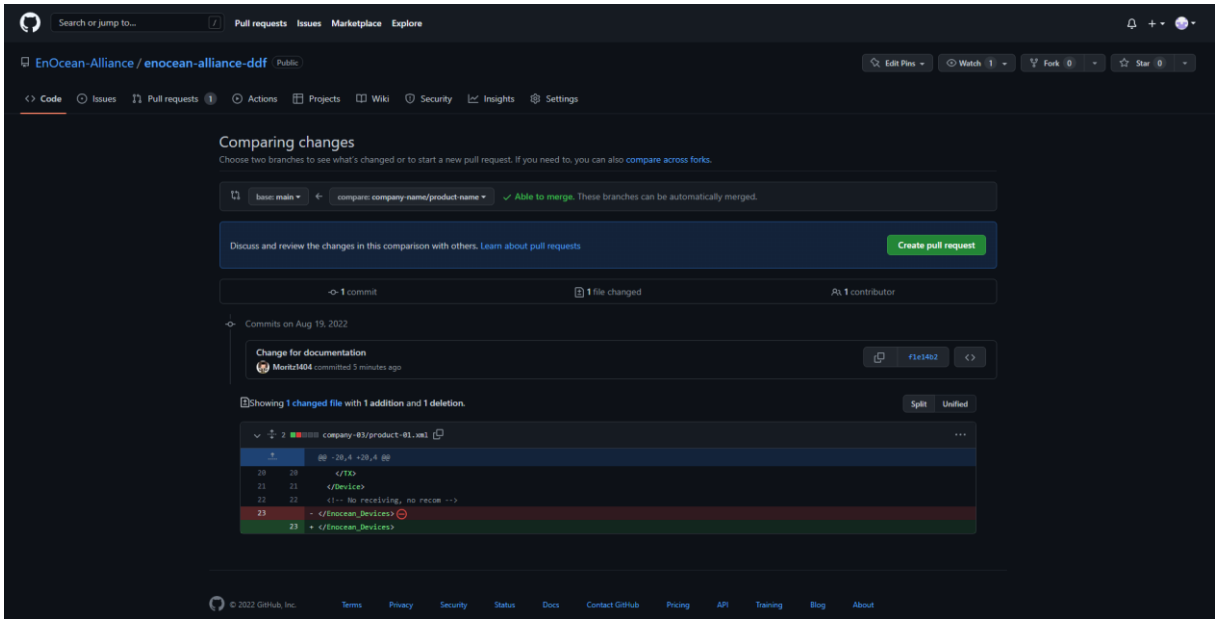
When ready, we want to merge our changes from our branch into the ***main*** branch, so that everyone knows that the DDF is ready for public use.

To do so, we need to create a pull request.

1. Go to ***Pull requests*** inside the ***enocean-alliance-ddf*** repository (https://github.com/EnOcean-Alliance/enocean-alliance-ddf/pulls).
2. On the upper right sight click ***New pull request***.

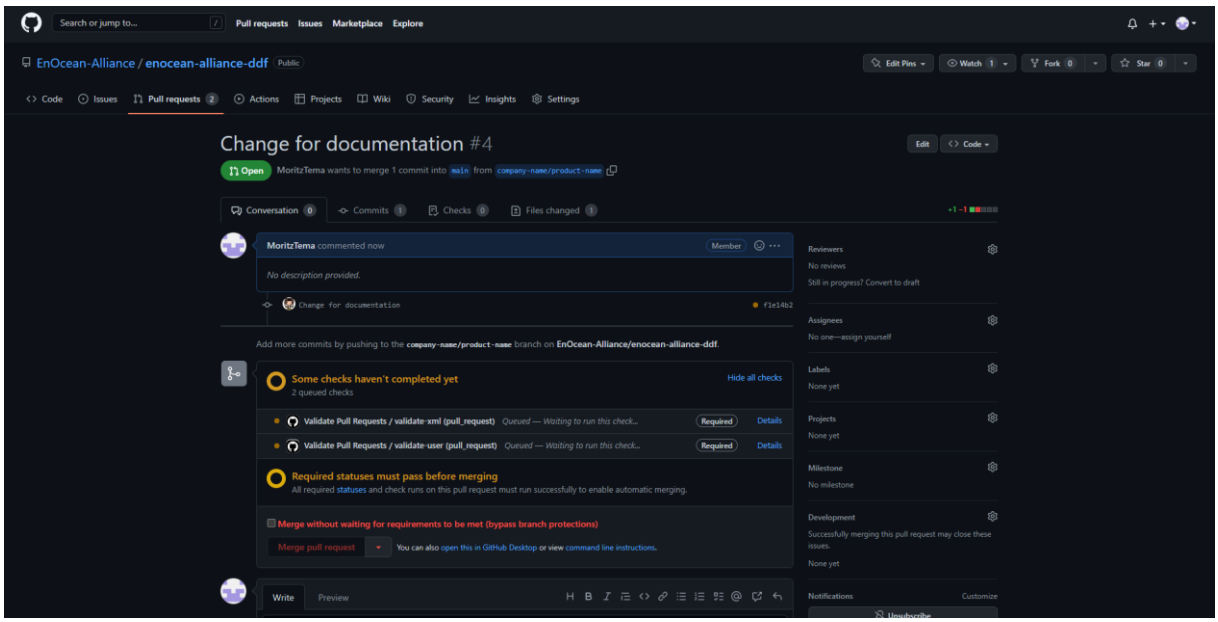


3. Now select the branch you want to merge into ***main***. (For this example: ***company-name/product-name***.)




Below that, you can see a history of all changes made inside this branch.

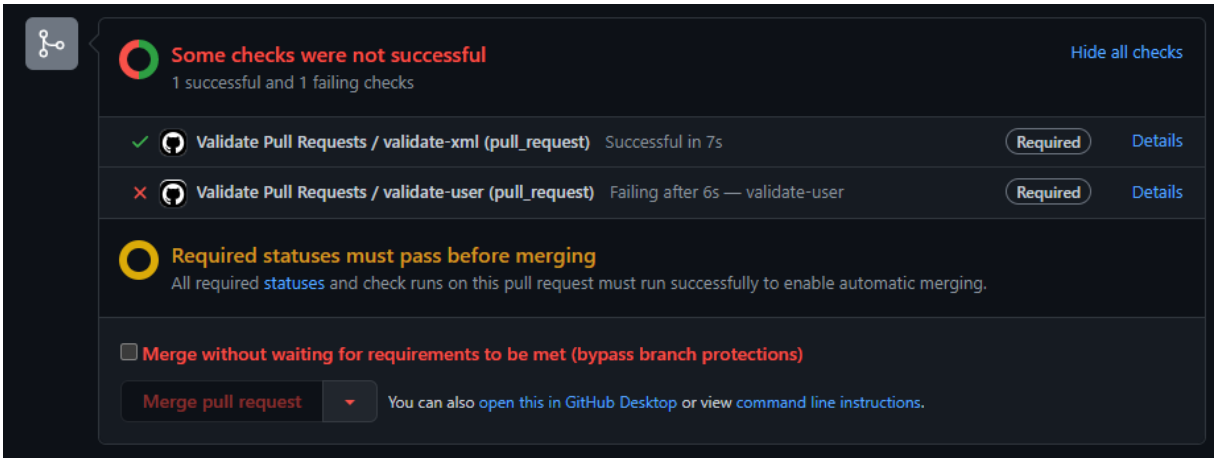
4. If everything looks fine, click ***Create pull request***.
5. Now you can add a title and a comment to describe your changes.
6. Then click ***Create pull request*** again.




After that your changes will be checked automatically.

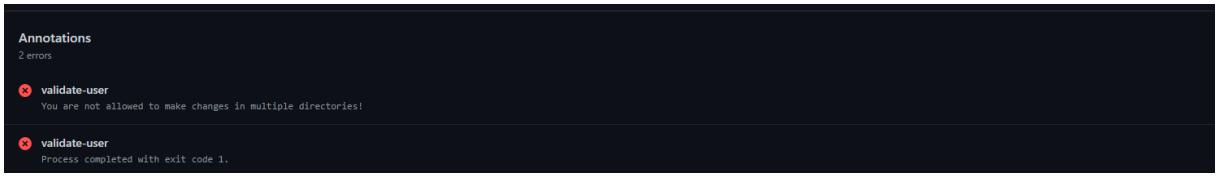
***validate-xml*** will check the DDFs (xml files) against the schema, to verify if they are correct.

***validate-user*** will check if the changes were only made inside the company folders you are authorized for.



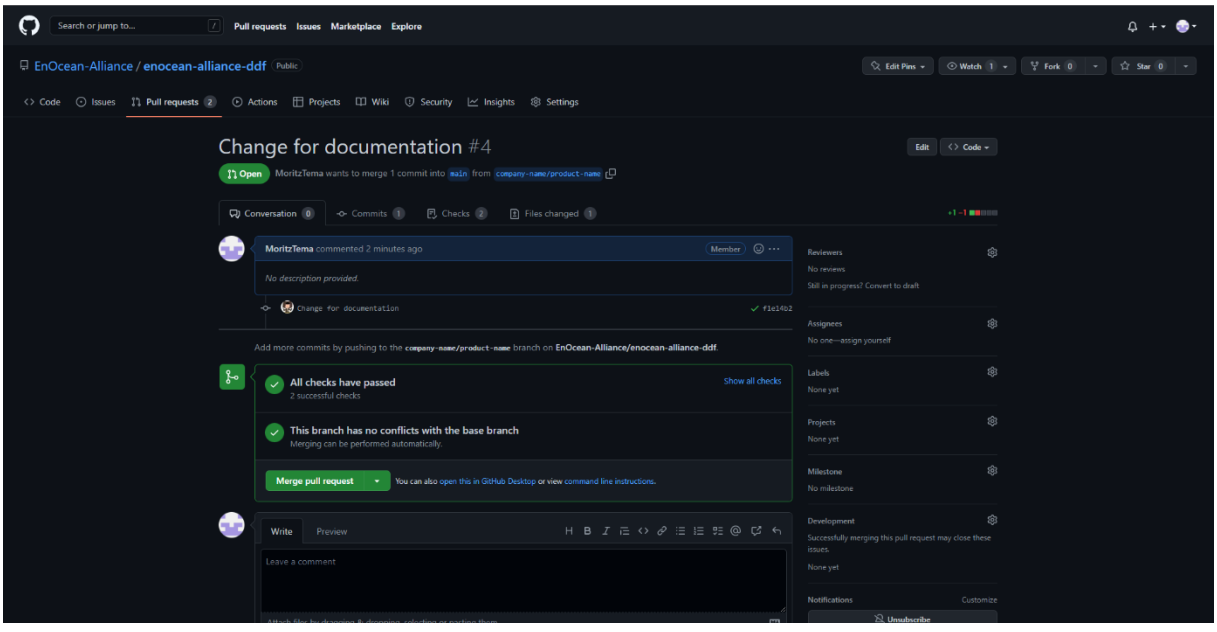

If a check fails, you need to fix the errors inside your branch. Your changes are automatically checked again.

As seen above: ***Bypass branch protections*** can be used to publish changes without the checks successfully completing. This is only available for administrators.




Example error message.

7. When all checks have passed, you are able to merge your pull request. For that, click ***Merge pull request ⇒ Confirm merge***.



8. All changes you made in ***company-name/product-name*** are now in the ***main*** branch.

   The ***company-name/product-name branch*** can be safely deleted, to keep the repository clean.

## 3.2. Automated Retrieval of DDF Files

To enable automated access to Device Description File (DDF) files, a central index file named index.xml is provided and regularly updated on the GitHub server. This file serves as a catalog of all available DDF files and contains essential metadata for each entry.

Each entry in index.xml includes the following information:

- **DDF File URL**: A direct link to the DDF file hosted in the repository.
- **Product Identifier**: A unique ID representing the device or product.

```
<?xml version="1.0" ?>
<index>
        <Product Path="company-01/ddf-example-ptm.xml" Product_ID="0xFFFFFFFFFFFF"/>
        <Product Path="company-02/ddf-exmaple-2.xml" Product_ID="0xFFFFFFFFFFFF"/>
        <Product Path="company-03/product-01.xml" Product_ID="0xFFFFFFFFFFFF"/>
        <Product Path="company-03/ddf-example-ptm.xml" Product_ID="0xFFFFFFFFFFFF"/>
        <Product Path="company-03/product-02.xml" Product_ID="0xFFFFFFFFFFFF"/>
</index>
```

***Access Workflow***

1. **Fetch index.xml**: Programs retrieve the latest version of index.xml from a predefined GitHub URL.
2. **Parse the XML**: The XML structure is parsed to extract entries, typically using an XML parser that supports XPath or DOM traversal.
3. **Select Relevant Entries**: Based on product ID or other criteria, the appropriate DDF entries are identified.
4. **Download DDF Files**: The URLs from the selected entries are used to fetch the corresponding DDF files directly from GitHub.

This mechanism allows clients and tools to stay synchronized with the latest DDF definitions without manual intervention, ensuring consistency and ease of integration.

# A. Appendix

## A 1. Changelog of the DDF

***2.0 – 29 January 2026***

- MSC support
- New key added: Placeholder
- New Parameterflags introduced: LinkTableInboundBoolean, LinkTableInboundChannel, LinkTableInboundEnum; Bitfield added.

***1.3 – 04 August 2020***

- XSD annotations added
- FuntionCode -> FunctionCode (typo)

***1.2 – 24 April 2018***

- XSD contained minor errors and was not valid
- Added attribute “SecuritySupported” in LinkTable MetaData
- Added Tag ReComVersion and RemanVersion to indicate version supported by firmware
- Added in TX Signals tag to indicate MID supported

***1.1 – 29 November 2016***

- ChipIDBased tag in TX has been renamed to EURID
- BaseIDBased tag in TX has been renamed to BaseID
- Properties type has been added to describe Device Property which are not configurable
- Parameters renamed to RecomParameters
- LinkEntry type added for EEP/GP tag for supported profiles to inform if a LinkEntry is used for the profile
- For RecomParameters the accessLevel (read/write or readWrite) and recommendedUserLevel (admin/user) attribute has been added.
- Product_ID: enforcing to use the 6Byte product id (2Byte Manufacturer ID + 4 byte Product Reference)
- LinkEntry Direction Ingoing renamed to Inbound and Outgoing to Outbound

***1.0 – 20 June 2016***

- Tag SupportedRPC added for a list of RPC which are supported by the Device
- Param Type for Recom Device Paramaters has been reworked:
    - Text type added.
    - Flag type removed.
    - Data type has been replaced with Scaled type, which uses the EEP way for describing scaled parameters

- The device parameter can have a RPCWriteStartAddress and RPCWriteLength attribute to use the RPC Write command to modify the parameters.
- Length of parameters uses bytes for length description

***Version 0.9 – 18 February 2015***

First revision of the DDF available to the TWG.