**Technical & Functional analysis**

**Breakout game with accelerometer controller using Thread/Matter**

|  |  |
| --- | --- |
|  |  |

Date : 5/2/2024

Contacts : Dekimo : Gysen Bart

Table of Contents

[1. Project overview 5](#_Toc165543283)

[1.1. Matter protocol 5](#_Toc165543284)

[1.2. OpenThread network protocol 5](#_Toc165543285)

[2. Game controller 6](#_Toc165543286)

[2.1. EFR32MG24 - BRD2601B Dev Kit 6](#_Toc165543287)

[2.2. EFR32MG24 Wireless SoC 7](#_Toc165543288)

[2.3. ICM-20689 6-Axis sensor 8](#_Toc165543289)

[2.4. User Button 9](#_Toc165543290)

[3. (Open) Thread 10](#_Toc165543291)

[3.1. OpenThread Border Router 11](#_Toc165543292)

[3.2. Rights – OpenThread – Thread 11](#_Toc165543293)

[4. Raspberry Pi 4 12](#_Toc165543294)

[4.1. Radio Co-Processor 13](#_Toc165543295)

[5. Matter protocol 13](#_Toc165543296)

[5.1. Terms 13](#_Toc165543297)

[5.2. About 15](#_Toc165543298)

[5.3. Documents 15](#_Toc165543299)

[5.4. Architecture 16](#_Toc165543300)

[5.4.1. Network 16](#_Toc165543301)

[5.4.2. Network Topology 17](#_Toc165543302)

[5.5. Data Model 19](#_Toc165543303)

[5.5.1. Node 19](#_Toc165543304)

[5.5.2. Endpoint 20](#_Toc165543305)

[5.5.3. Clusters 20](#_Toc165543306)

[5.5.4. Custom Cluster 22](#_Toc165543307)

[5.5.5. Cluster Client / Server 22](#_Toc165543308)

[5.5.6. Device Type 22](#_Toc165543309)

[5.6. Interaction Model 22](#_Toc165543310)

[5.6.1. Read Interaction 24](#_Toc165543311)

[5.6.2. Subscription Interaction 25](#_Toc165543312)

[5.6.3. Write Interaction 26](#_Toc165543313)

[5.6.4. Invoke Interaction 28](#_Toc165543314)

[5.7. Security 29](#_Toc165543315)

[5.7.1. Session establishment 29](#_Toc165543316)

[5.7.2. Message confidentiality and integrity 30](#_Toc165543317)

[5.8. Commissioning 30](#_Toc165543318)

[5.8.1. Prerequisites 30](#_Toc165543319)

[5.8.2. Onboarding payload 31](#_Toc165543320)

[5.8.3. Commissioning steps 31](#_Toc165543321)

[5.9. Fabric 33](#_Toc165543322)

[5.9.1. Multi-Admin 33](#_Toc165543323)

[5.10. Matter Hub & Bridge 34](#_Toc165543324)

[5.10.1. Bridge 34](#_Toc165543325)

[5.10.2. Hub 34](#_Toc165543326)

[5.11. Matter production guide 34](#_Toc165543327)

[5.11.1. CSA Member 34](#_Toc165543328)

[5.11.2. CSA Certification 35](#_Toc165543329)

[5.12. Develop with Matter 36](#_Toc165543330)

[5.12.1. Official Matter SDK 36](#_Toc165543331)

[5.12.2. Manufacturer SDK 36](#_Toc165543332)

[5.12.3. Certification test 36](#_Toc165543333)

[6. Silabs Gecko SDK V4.4.0 – SSv5 36](#_Toc165543334)

[6.1. Installation 36](#_Toc165543335)

[6.2. Matter Examples 38](#_Toc165543336)

[6.3. Zigbee Cluster Configurator 39](#_Toc165543337)

[6.3.1. Adding endpoint 39](#_Toc165543338)

[6.3.2. Modifying endpoint 40](#_Toc165543339)

[6.3.3. Custom Cluster 41](#_Toc165543340)

[6.4. Software Components 42](#_Toc165543341)

[7. Setting up a Matter Hub 42](#_Toc165543342)

[7.1. Silicon Labs Hub image 42](#_Toc165543343)

[7.2. Radio Co-Processor 42](#_Toc165543344)

[7.3. Chip-tool 43](#_Toc165543345)

[8. Matter Tests 43](#_Toc165543346)

[8.1. Simple on/off device 43](#_Toc165543347)

[8.2. Level control test 46](#_Toc165543348)

[8.2.1. Capture data using chip-tool 48](#_Toc165543349)

[8.3. Terminal output test 49](#_Toc165543350)

[9. ICM-20689 Inertial sensor 49](#_Toc165543351)

[9.1. Using the ICM-20689 driver 50](#_Toc165543352)

[9.2. Tests 52](#_Toc165543353)

[9.2.1. Raw data 52](#_Toc165543354)

[9.2.2. Converted data 54](#_Toc165543355)

[10. Issues 54](#_Toc165543356)

[10.1. Gyroscope sensor problem 54](#_Toc165543357)

[10.1.1. Possible solution 54](#_Toc165543358)

[10.1.2. Solution 55](#_Toc165543359)

[10.2. Connection between matter device (using chip-tool) and the “game” 55](#_Toc165543360)

[10.2.1. Possible solution 55](#_Toc165543361)

[10.2.2. Solution 55](#_Toc165543362)

[11. Application game controller 55](#_Toc165543363)

[12. Breakout game 55](#_Toc165543364)

[13. Notes 55](#_Toc165543365)

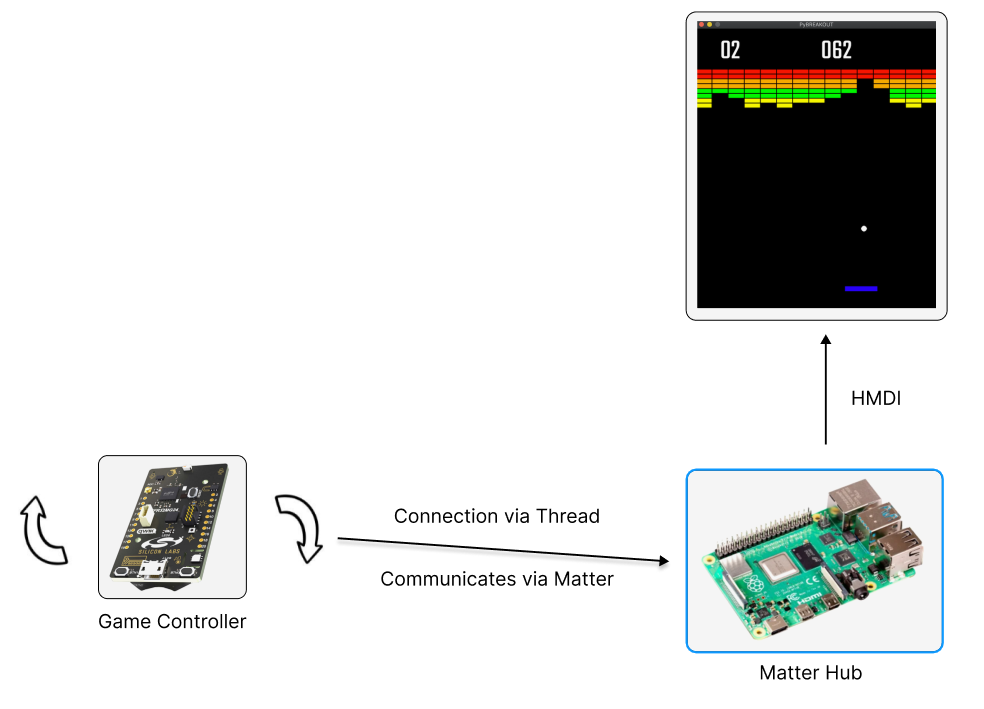
[14. Document Information 57](#_Toc165543366)

[14.1. Version History 57](#_Toc165543367)

[14.1. Related Documents 58](#_Toc165543368)

# Project overview

This project is an interactive breakout game controlled by motion with a wireless controller. The controller is connected to a Thread network and communicates with the Matter protocol. The breakout game is displayed on a tv using a SBC (Single Board Computer) that is also connected to a Thread network.



## Matter protocol

Matter, formerly known as “Project CHIP” (Connected Home over IP) is an open-source, royalty-free connectivity standard designed to make it easier for various smart devices to communicate with each other. It aims to create a unified standard for the Internet of Things devices. A Matter device can connect to a network with Thread, WIFI and Ethernet.

## OpenThread network protocol

OpenThread is an low-power mesh networking protocol that is based on IPv6. OpenThread is an open standard and is built for IoT applications. It uses 6LoWPAN which uses IEEE 802.15.4 (2.4Ghz) wireless protocol with mesh communication.

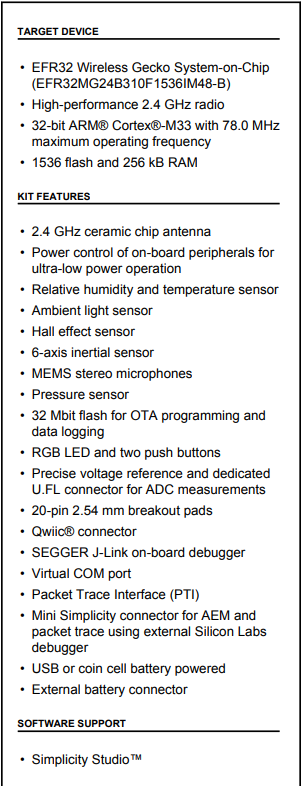
# Game controller

The wireless controller must be equipped with:

* 2.4Ghz wireless controller used for the OpenThread connection
* Accelerometer used to interact with the game (moving the bar)
* Button used for interact with the game (pause, start, etc.)
* A battery to provide power to the controller

## EFR32MG24 - BRD2601B Dev Kit

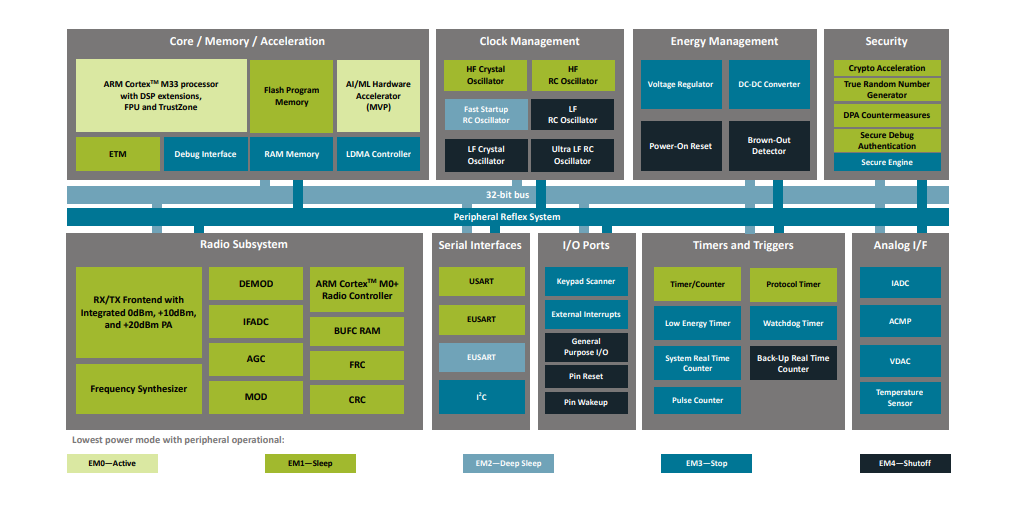
The EFR32MG24 Dev Kit board (BRD2601B) supports all the elements that is necessary for the controller. So there is no need to develop a PCB with all this components.

## EFR32MG24 Wireless SoC

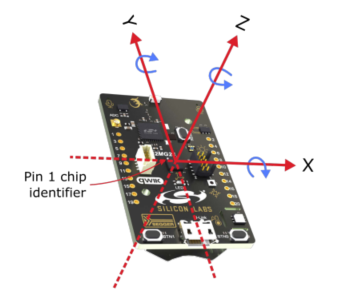
The development board uses the EFR32MG24 wireless SoC (System on a Chip). This SoC is ideal for meshing wireless solutions using Matter and OpenThread. It provides al the built in features that are relevant for this project, like:

* High performance 32-bit 78 MHz Arm Cortex®-M33 processor
* 1536 kB flash and 256 kB of RAM
* High performance 2.4GHz Radio
* OpenThread and Matter support
* Security features for protection against hardware and software attacks
* A wide range of peripherals like I²C, SPI, USART, ADC, Timers, GPIO’s, etc.



## ICM-20689 6-Axis sensor

The development board contains a 6-axis sensor ICM-20689. This 6-axis sensor combines a 3-axis gyroscope and a 3-axis accelerometer. It detects acceleration and angular rate in the X, Y and Z axes.

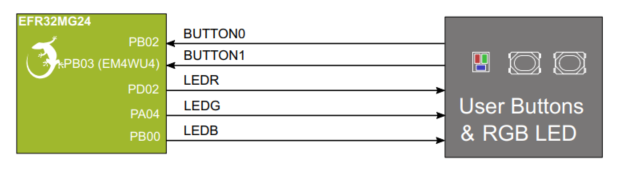


In the development board the sensor is connected an communicates over SPI. The SPI lines are interrupted through a switch to prevent power consumption when not used. Before the sensor can be used in the application it must be enabled by setting PC09 high.

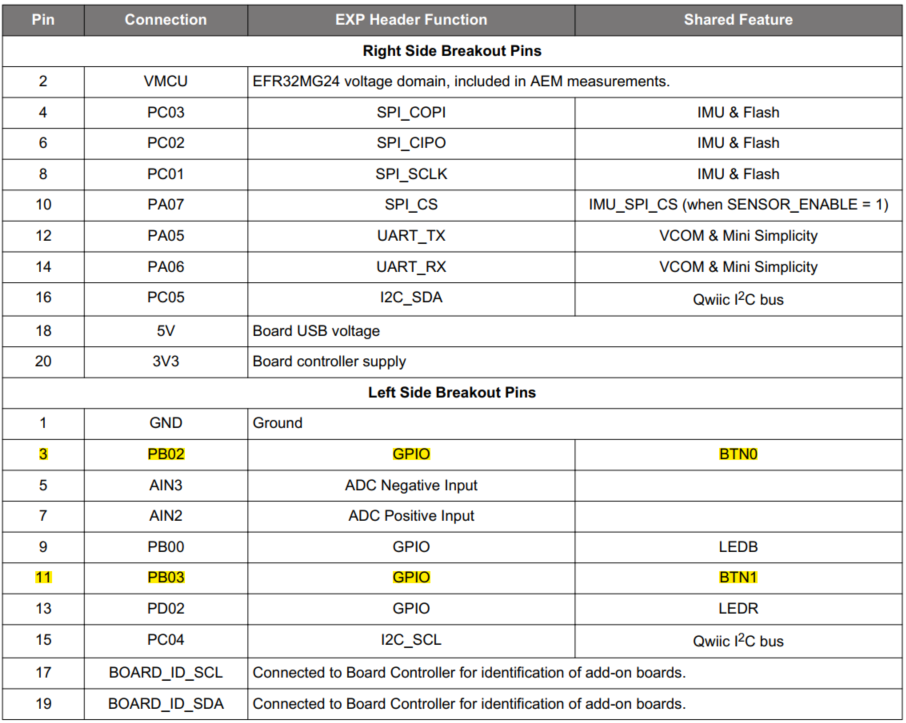


## User Button

The development board has two user buttons, BTN0 and BTN1. They are directly connected to the EFR32MG24 SoC and are debounced by RC filters.



The buttons are connected to the pins PB02 and PB03. The pins are also available on the expansion header. The buttons on the board can be used during the development and later on we can use buttons in a case that are connected to the expansion header.



# (Open) Thread

Thread is a low-power IPv6 mesh networking standard for IoT devices. It is designed for low-power Internet of things devices. The low-power aspect is important for battery-powered smart home devices. However, it’s also low-bandwidth, making it ideal for applications that don’t send a lot of data, like switches or motion sensors.

Thread uses the same technology as Zigbee (IEEE 802.15.4) but provides IP connectivity similar to Wi-Fi. Unlike Zigbee, Thread by itself does not allow controlling devices: It is just a communication protocol. To control the Thread devices, a higher-level protocol is required e.g. Matter. Thread devices use the IPv6 standard to communicate both inside and outside the mesh network.

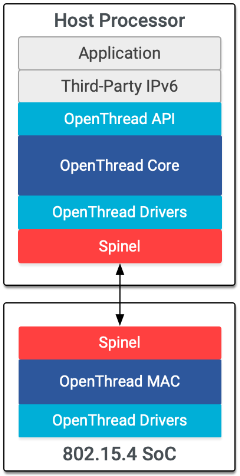
**Thread vs OpenThread**

The Thread protocol specification is available at no cost; however, this requires agreement and continued adherence to an End-User License Agreement (EULA), which states that "Membership in Thread Group is necessary to implement, practice, and ship Thread technology and Thread Group specifications.

OpenThread released by Google is an open-source implementation of Thread®. Google has released OpenThread to make the networking technology used in Google Nest products more broadly available to developers, in order to accelerate the development of products for the connected home and commercial buildings.

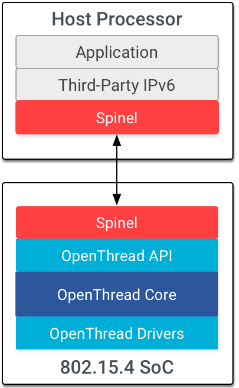
With a narrow platform abstraction layer and a small memory footprint, OpenThread (OT) is highly portable. It supports both System-on-Chip (SoC) and Co-Processor (RCP, NCP) designs.

**RCP**



A OTBR supports a Radio Co-Processor (RCP) design. In this design the core of OpenThread lives on the host processor with only a minimal MAC layer on the processor with the Thread radio. The host processor typically doesn’t sleep in this design, in part to ensure reliability of the Thread network. The advantage of this design is that more resources are available on the more powerful host processor. Communication between the RCP and the host processor is managed by OpenThread Daemon over the Spinel protocol.

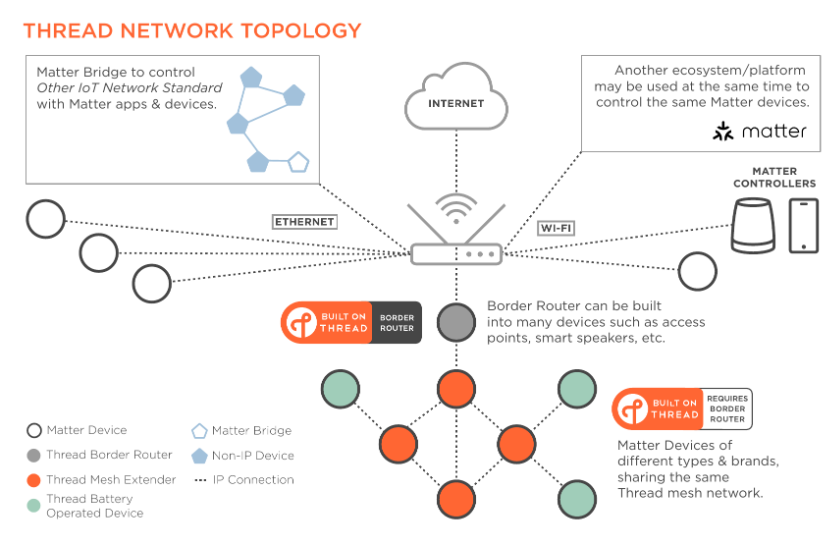
**NCP**



The standard NCP design has Thread features on the SoC and runs the application layer on a host processor, which is typically more capable (but has greater power demands) than the OpenThread device. Communication between the NCP and the host processor is managed by wpantund through a serial interface, typically using SPI or UART, over the Spinel protocol. The benefit of this design is that the higher-power host can sleep while the lower-power OpenThread device remains active to maintain its place in the Thread network. And since the SoC is not tied to the application layer, development and testing of applications is independent of the OpenThread build.

## OpenThread Border Router

A OpenThread Border Router (OTBR) connects a Thread network to other IP-based networks such as Wi-Fi or Ethernet. To communicate outside the mesh network a Border Router is needed. The OTBR routes packets between your local network and the Thread mesh network. It does not look at the content of these packets, it just forwards them.



## Rights – OpenThread – Thread

OpenThread released by Google is an open-source implementation of Thread technology. If a company uses OpenThread to build a product, they need to be a member of the Thread Group in order to gain the Intellectual Property (IP) rights to ship Thread products and to complete product certification, which ensures that products using Thread work together effortlessly and securely right out of the box. If developers choose not to join Thread Group and ship products using Thread technology, they are not conferred the IP rights required to practice and ship Thread technology, and may subject themselves to legal action, including but not limited to licensing fees.

# Raspberry Pi 4

In this project, a Raspberry Pi 4 is used as a Single Board Computer. It not only runs the Breakout game but also functions as a Matter hub using the CHIP tool and as a OpenThread Border Router.

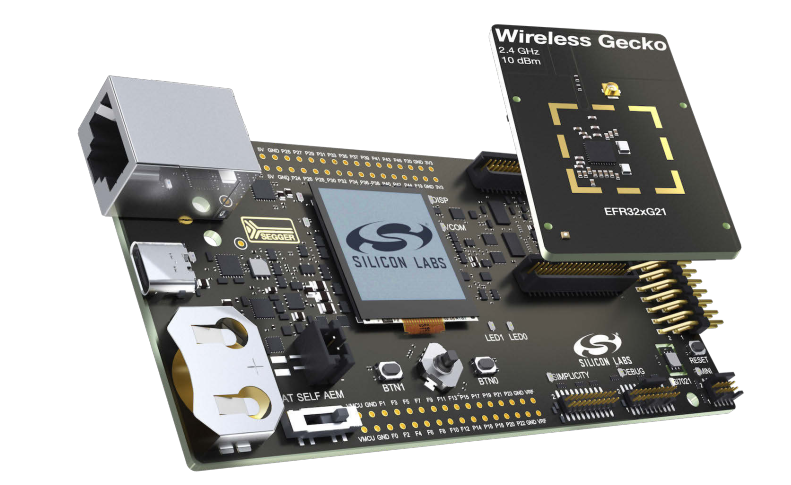
The Raspberry Pi 4 is a single-board computer developed by the Raspberry Pi Foundation. It is a popular choice for hobbyists, educators, and professionals due to its affordability, versatility, and ease of use. It is powerful enough for this use case, the most important specs for us is:

* **CPU:** Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
* **RAM:** 4GB LPDDR4-3200 SDRAM
* **Storage:** Micro-SD card slot for loading operating system and data storage
* **Connectivity:**
  + **USB Ports:** It has two USB 3.0 ports and two USB 2.0 ports
  + **Ethernet:** Gigabit Ethernet (RJ45) port for wired networking
  + **Wireless:** Dual-band 802.11ac wireless networking and Bluetooth 5.0
* **Display output:** It has two micro-HDMI ports

## Radio Co-Processor

The OpenThread Border Router (OTBR) must have physical access to the IEEE 802.15.4 network that is used by the Thread protocol. Raspberry Pi does not have such radio capability, we need to add a Radio Co-Processor to achieve this.

As Radio Co-Processor I used a wireless starter kit mainboard (BRD4001A) developed by Silicon Labs with a EFR32xG21 SoC plug-in board (BRD4181B). This starter kit provides all the necessary features to set up a OTBR.



# Matter protocol

## Terms

In order to understand this section we need to know the definitions of several terms.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Administrator | A Node having Administer privilege over at least the Access Control Cluster of another Node. |
| Attribute | A data entity which represents a physical quantity or state. This data is communicated to other Nodes using commands. |
| Border Router | A router, also known as Edge Router, that provides routing services between  two IP subnets (typically, between a hub network and a peripheral network). |
| Bridge | A Node that represents one or more non-Matter devices on the Fabric. |
| Bridged Device | A non-Matter device that is represented on the Fabric by a Bridge so it can be used by Nodes on the Fabric |
| Certificate Authority (CA) | An entity that issues digital certificates such as a DAC or NOC. |
| Client | A Cluster interface that typically sends commands that manipulate the  attributes on the corresponding server cluster. A client cluster communicates  with a corresponding remote server cluster with the same cluster identifier. |
| Cluster | A specification defining one or more attributes, commands, behaviors and  dependencies, that supports an independent utility or application function.  The term may also be used for an implementation or instance of such a specification on an endpoint. |
| Command | Requests for action on a value with an expected response which may have  parameters and a response with a status and parameters. |
| Commission | To bring a Node into a Fabric. |
| Commissioner | A Role of a Node that performs Commissioning. |
| Commissionee | An entity that is being Commissioned to become a Node. |
| Commissioning | Sequence of operations to bring a Node into a Fabric by assigning an Operational Node ID and Node Operational credentials. |
| Controller | A Role of a Node that has permissions to enable it to control one or more  Nodes. |
| Controlee | A Role of a Node that has permissions defined to enable it to be controlled by one or more Nodes. |
| Device | A piece of equipment containing one or more Nodes. |
| Device Attestation Certificate (DAC) | An RFC 5280 compliant X.509 v3 document with attestable attributes. |
| Discriminator | A 12-bit value used to discern between multiple commissionable Matter device advertisements. |
| Endpoint | A particular component within a Node that is individually addressable. |
| Fabric | A logical collection of communicating Nodes, sharing a common root of trust,  and a common distributed configuration state. |
| Node | An addressable entity which supports the Matter protocol stack and (once  Commissioned) has its own Operational Node ID and Node Operational credentials. A Device MAY host multiple Nodes. |
| Node Operational Certificates (NOCs) | are installed during the Matter network commissioning by the commissioner together with Trusted Root CA Certificates. |
| Onboarding Payload | The information needed to start the process of commissioning a Device. |
| OTA Provider | A Node implementing the OTA Software Update Provider role. |
| OTA Requestor | A Node implementing the OTA Software Update Requestor role. |
| Product Attestation Authority | An entity which operates a root level Certificate Authority for the purpose of  Device Attestation. |
| Product ID (PID) | A 16-bit number that identifies the type of a Device, uniquely among the product types made by a given vendor. |
| Root of trust | a concept within Matter that is centered around a certification authority (CA), identified by Root Public Key (Root PK). |
| Server | A Cluster interface that typically supports all or most of the attributes of the  Cluster. A Server Cluster communicates with a corresponding remote Client  Cluster with the same Cluster identifier. |
| Vendor ID (VID) | A 16-bit number that uniquely identifies the Vendor of the Device. |

## About

Matter is a royalty-free connectivity standard for smart home devices that was announced in December 2019 and officially released in September 2022. It is designed to enable seamless communication and interoperability between smart home devices from different manufacturers, regardless of their underlying technology or platform. Matter was developed by the Connectivity Standards Alliance (CSA), a non-profit organization that promotes interoperability among diverse technology ecosystems. The CSA was formed in 2021 by merging the Zigbee Alliance and the Project Connected Home over IP (CHIP) working group.

## Documents

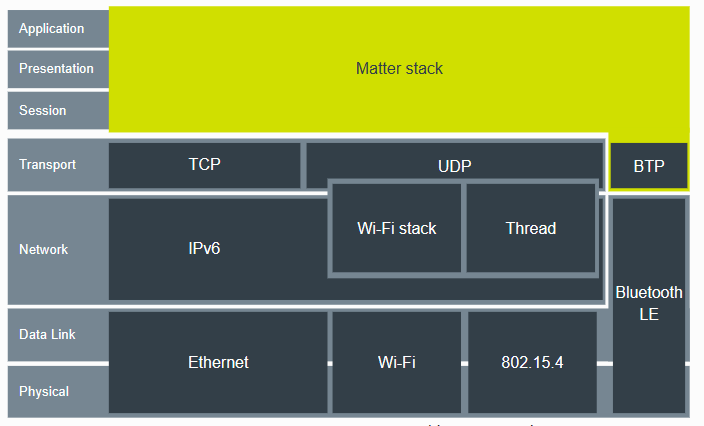
The Matter protocol is specified in Three main documents and can be downloaded from the CSA website. These documents are necessary when developing a Matter product. With every update these documents changes so make sure when developing a product you get the latest documents.

* **Matter-(version)-Core-Specification**In this document you can find all about the features of the protocol, the requirements, technical specifications etc.
* **Matter-(version)-Device-Library-Specification**This document tells which device-types are included in this specific version, it also tells what clusters can and must be used with a specific device-type.
* **Matter-(version)-Application-Cluster-Specification**This document tells all the available clusters in this specific version. It tells what attributes and commands are part of a clusters and which attributes and commands are mandatory or optional for a specific cluster.

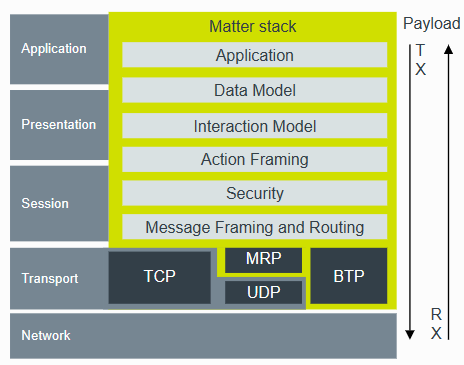
## Architecture

### Network

Matter is a universal IPv6-based communication protocol designed for smart home and the Internet of Things devices. Matter works on the Application Layer of the OSI model.



The Matter protocol stack is divided into layers to separate the different responsibilities and introduce a good level of encapsulation amongst the various parts of the protocol stack.



* **Application Layer:** High-order business logic of a device. For example, an application that is focused on lighting might contain logic to handle turning on/off the light as well as its color characteristics.
* **Data Model:** The data layer corresponds to the data and verb elements that help support the functionality of the application. The Application operates on these data structures when there is an intent to interact with the device.
* **Interaction Model:** The Interaction Model layer defines a set of interactions that can be performed between a client and server device. For example, reading or writing attributes on a server device would correspond to application behavior on the device. These interactions operate on the elements defined at the data model layer.
* **Action Framing:** Once an action is constructed using the Interaction Model, it is serialized into a prescribed packed binary format to encode for network transmission.
* **Security:** An encoded action frame is then sent down to the Security Layer to encrypt and sign the payload to ensure that data is secured and authenticated by both sender and receiver of a packet.
* **Message Framing & Routing:** With an interaction encrypted and signed, the Message Layer constructs the payload format with required and optional header fields; which specify the message's properties and some routing information.
* **IP Framing & Transport Management:** After the final payload has been constructed, it is sent to the underlying transport protocol for IP management of the data.

### Network Topology

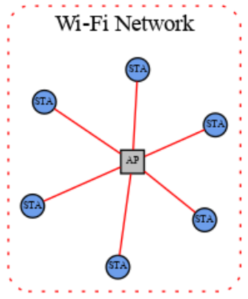
In principle, any IPv6 network is suitable for Matter deployment butt the focus is on three link layer technologies: Ethernet, Wi-Fi and Thread. Matter treats networks as shared resources: it makes no stipulation of exclusive network ownership or access. As a result, it is possible to overlay multiple Matter networks over the same set of constituent IP networks.

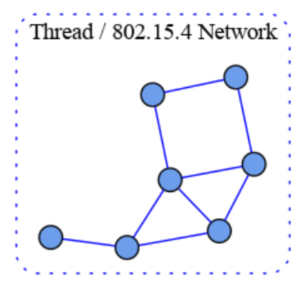
This protocol may operate in the absence of globally routable IPv6 infrastructure. This requirementenables operation in a network disconnected or firewalled from the global Internet. It also enablesdeployment in situations where the Internet Service Provider either does not support IPv6 on consumer premises or where the support proves otherwise limiting.

This protocol supports local communications spanning one or more IPv6 subnets. Canonical networks supporting a fabric may include a Wi-Fi/Ethernet subnet, or one or more low power and lossy network (LLN) subnets. In this version of the specification, Thread is the supported LLN standard.

**Single network**

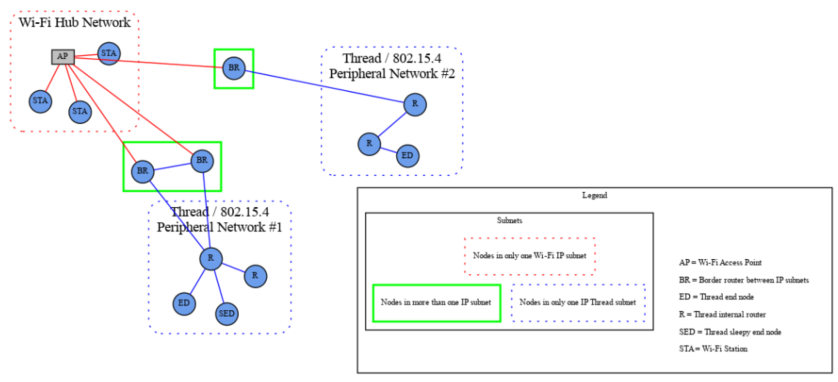
In the single network topology, all Matter devices are connected to a single network. This could by a Thread network, Wi-Fi or Ethernet network. In the case of Wi-Fi/Ethernet, the network could in fact span multiple Wi-Fi and/or Ethernet segments provided that all the segments are bridged at the link layer.





**Star network topology**

The star network topology consists of multiple peripheral networks joined together by a single hub network. The hub network will be the customer’s home network. while the peripheral networks can be of any supported network type. A peripheral network must always be joined directly to the hub network via one or more Border Routers.



In the star network topology any number of peripheral network may be present in a single fabric, including networks of the same type. Nodes may have interfaces onto any network and can communicate directly to other nodes on the same network. If a node needs to cross a network to communicate it must flow through a Border Router.

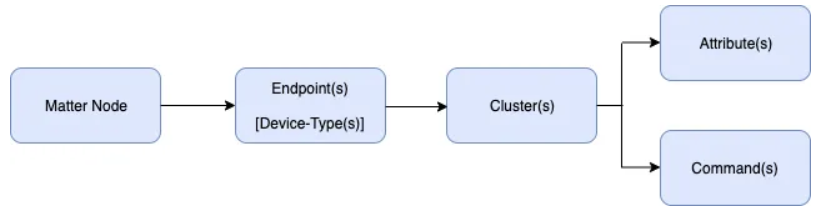
Regardless of the Network topology being used, Matter has a concept of Fabrics. A Matter Fabric is a security domain that contains a collection of nodes. These nodes can be identified and can communicate with each other within the context of that security domain. Each Matter Fabric has a unique Node ID for each node within the fabric and has a unique Fabric ID. Any Matter device can be a part of multiple Matter fabrics, and in turn will have multiple associated Node IDs / Fabric IDs depending on the fabric it is communicating with.

This protocol places a set of requirements on the Border Router. These requirements pertain to

address assignment, route assignment and advertisement, multicast support, and discovery proxying.

## Data Model

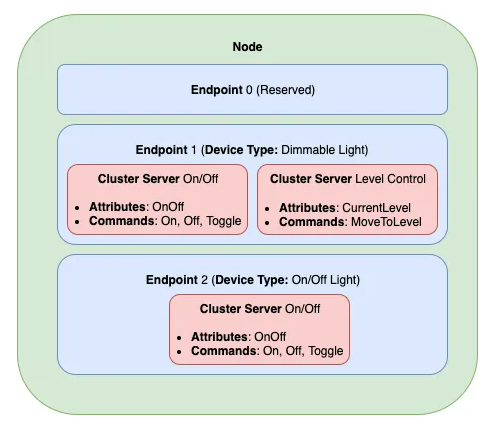
The data model in Matter is a hierarchical modeling of a devices features, including nodes, endpoints, clusters and device types where the node is the highest level data element. A single device can be represented by one or more nodes. An environment where multiple Matter nodes interoperate is referred as a Matter fabric.



### Node

This is a uniquely network addressable entity that exposes some functionality. This is typically a physical device that a user can recognize as a whole device. The role of a node is a set of related behaviors. A node can contain one or more roles including:

* Commissioner: Refers to the process of assigning Fabric credentials to a new device.
* Controller: A node that can control one or more nodes such as a On/Off switch
* Controlee: A node that can be controlled by one or more nodes. Such as an actor. Devices that have the controller role cannot be a controlee.
* OTA Provider: Provides OTA software updates.
* OTA Requestor: Requests OTA software updates.



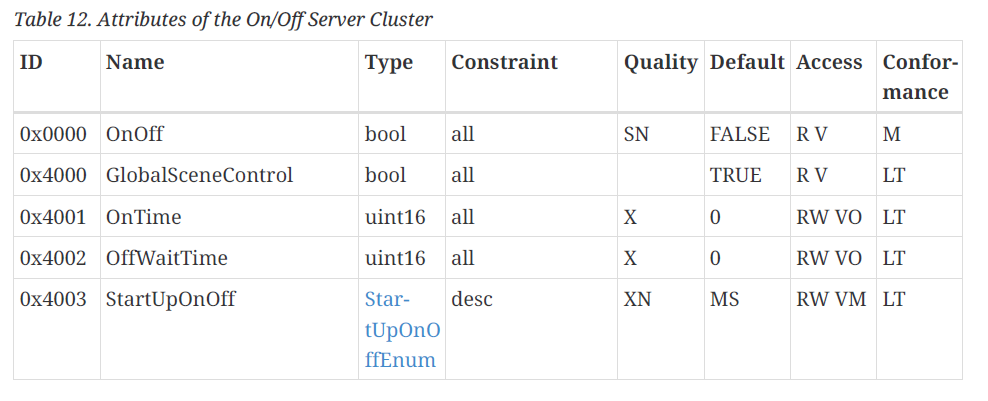
### Endpoint

Each node has one or more endpoints. A endpoint contain a set functionality’s of a single device. In the example above endpoint 1 is a dimmable light that have the functionality turning on or off and have a functionality level control, that controls the brightness of the light. Endpoint 2 have only the functionality turning on or off. Note that endpoint 0 is reserved for utility functions like diagnostics, updates Over The Air…

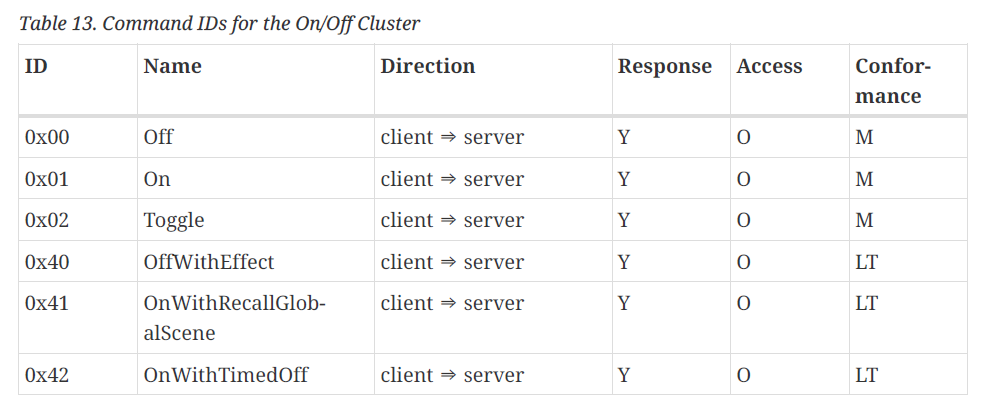
### Clusters

A cluster groups together commonly used functionality in a reusable building block. Endpoint 1 has two cluster which describes their own functionality. Within the clusters they contain:

* **Attributes**: Attributes are data entities that represent a physical quantity or state. Each attribute is listed in a table with data quality columns: ID, Name, (Data) Type, Constraint, other Quality, Access, Default (value), and Conformance. An attribute also defines its associated semantics and behavior.



* **Commands**: A cluster command provides an ability to invoke a specific behavior on the cluster. A command may have parameters that are associated with it. Each command SHALL be listed in a table with data quality columns: ID, Name, Direction, Response, Access, Conformance. The command table SHALL define the direction of the command as either client to server or server to client. The command table SHALL define the access privileges for each request command or omit the privileges for the default.



* **Events:** Events are a type of attributes that communicate device state changes. They can also be treated as historical data records of something that happened on the device in the past.

The clusters that are supported can be found at the **Matter-(version)-Application-Cluster-Specification** document.

### Custom Cluster

It is possible to create a custom cluster that is suitable for a specific use case. For example, for a project you need a specific attribute that is not available in the cluster that you going to use. You can extend that cluster so the attribute is available. While creating a custom cluster is possible, the cluster will only work between your device and the custom Matter controller expecting the cluster. The custom cluster will not work with a non-custom Matter controller. For interoperability, the cluster needs to be published in the cluster specifications, which only occurs if your organization is at least an adoption-level member of the CSA.

If considering working with a custom cluster, I recommend reading **section 7.10.4** in the core specification.

### Cluster Client / Server

A cluster server is stateful and holds attributes, events and commands while a client is stateless and is responsible to initiate interactions with a cluster server.

### Device Type

A Matter device type is an officially defined collection of requirements for one or more endpoints. Device types are intended to ensure interoperability of different device brands on the market. Each device type definition is composed of the following elements:

* Device type ID
* Device type revision
* One or more mandatory clusters, including each cluster’s minimum revision

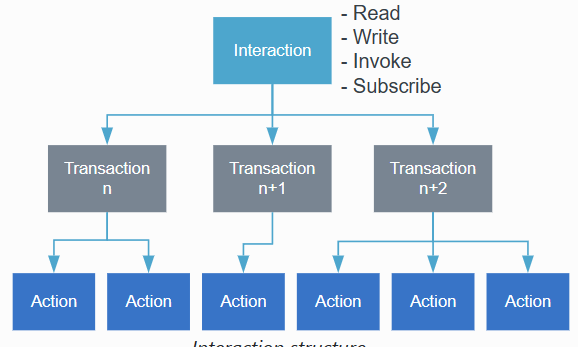
All the device types are defined in the **Matter-(version)-Application-Cluster-Specification.pdf** document.

**Relating Matter to Zigbee**

Ultimately, Matter serves to extend existing protocol stacks to maintain and bolster their architecture for future use. Thus, the Data Model originates from and resembles the Dotdot Architecture Model and Chapter 2 of the Zigbee Cluster Library Specification found here: <https://csa-iot.org/developer-resource/specifications-download-request/>. The Matter Data Model better defines the architecture in the Zigbee Cluster Library while keeping the certifiable cluster specifications.

## Interaction Model

The Matter Interaction Model defines the methods of communication between nodes and serves as the common language for node to node transmission. Node communicate with each other through interactions. Interactions are a sequence of one or more transactions, which in turn are a sequence of actions.



For example, when a client cluster initiate a Read Transaction and requests to read an attribute, the server cluster can respond by giving the attribute. The client request and the server response are separated actions but they are part of the same Read Transaction, these actions and transactions belong to the Read Interaction.

There are four types of interactions:

* **Read Interaction**
* **Write Interaction**
* **Subscribe Interaction**
* **Invoke Interaction**

These types except for Subscribe Interaction consist of one transaction. The Interaction Model supports five types of transactions:

* **Read**: Get attributes and/or events from a server.
* **Write**: Modify attribute values.
* **Invoke**: Invoke cluster commands.
* **Subscribe**: Create subscription for clients to receive periodic updates from servers automatically.
* **Report**: Maintain the subscription for the Subscribe Interaction.

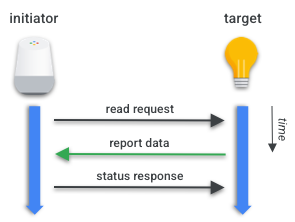
There are several concepts that is important for understanding transactions before explaining the different Interactions.

* **Initiators and Targets**: Interactions happen between initiator and target nodes. The initiator starts the transaction and the target responds. Usually the client cluster is the initiator node and de server the target node.
* **Transaction ID**: The transaction ID field must be present in all actions that are part of a transaction to indicate the logical grouping of the actions as part of one transaction. All actions that are part of the same transaction must have the same transaction ID.
* **Groups**: Groups allow an initiator to send an action to multiple targets. This type of communication is known as a groupcast, which leverages IPv6 multicast messages.
* **Paths**: Paths are the location of the attribute, event or command an interaction seeks to access. For example:

****

When groupcasting, a path may include the group or a wildcard operator to address several nodes simultaneously, decreasing the number of actions and thus decreasing the response time of an interaction. Without groupcasting there is a chance of latency between multiple devices reacting to an interaction.

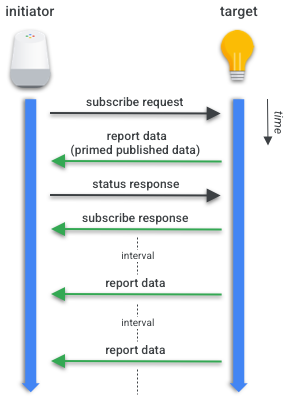
### Read Interaction

When an initiator wishes to determine the value of one or more attributes or events on a node, an Read Interaction is generated. In this interaction the following actions occur:

* **Read Request:** First action of a Read transaction. The initiator requests a list of the target’s attributes and/or events.
* **Report Data:** Generated in response to the Read Request action. The target sends the requested list of attributes and/or events back along with a suppress response and a subscription ID.
  + Suppress response: a flag that determines whether the status response to this action should be suppressed.
  + Subscription ID: Integer that identifies the subscription transaction, only included if the report is part of a Subscription Transaction.
* **Status Response Action (optional):** Only generated if the suppress response flag is not set. A status Response Action will be generated with a status code:
  + SUCCESS: to continue the interaction
  + INVALID SUBSCRIBTION: if the action is part of a Subscribe interaction and the Subscription ID is invalid
  + FAILURE: to terminate the interaction

Read Transactions are restricted to unicast, the Read Request and Report Data actions cannot target groups of nodes because the Status Response Action cannot be generated as a response to a groupcast.

### Subscription Interaction

Subscription is used by a initiator to automatically receive updates of an attribute or event. This creates a relationship between two nodes where the target/publisher periodically generates Report Data Actions to the initiator/subscriber.

A subscribe Request Action contains the following elements:

* **Min Interval Floor:** The minimum interval between reports
* **Max Interval Ceiling:** The maximum interval between reports
* **Attribute Reports:** A list of zero or more of the reported Attributes requested in the Read Action Request.
* **Event Reports:** A list of zero or more reported Events.

After the Subscribe request by the initiator, the target responds with a Report Data Action containing the first batch of reported data, the primed published data. Then the initiator acknowledges with a Status Response Action send to the target. Once the Target receives a Status Response Action reporting no errors, it sends a Subscribe Response Action. Then the target will send Report Data Action periodically at the negotiated interval until the subscription is lost or cancelled.

Subscription interactions have some restrictions:

* The subscribe Request and Response Action are Unicast-only
* Report Data Actions in the same Subscription Interaction must have the same Subscription ID.
* If the subscriber does not receiver a Report Data Action within the maximum interval between Actions, the subscription will be terminated.

### Write Interaction

The Write Interaction is generated when a initiator wants to modify values of one or more attributes located on one or more nodes. The initiator has the option to use a Timed or a Untimed Write Transaction.

**Untimed Write Transaction**

In the Untimed Write Transaction there are two actions:

* **Write Request Action:**A Write Transaction works similar to the Read Request Action. The initiator provides the target with:
  + **Write Request:** A list of one or more tuples containing Path and data.
  + **Timed Request:** A flag that indicates whether this action is part of a Timed Write Transaction.
  + **Suppress Response:** A flag that indicates whether the Response Status Action should be suppressed
* **Write Response Action:**When the target receives the Write Request Action it will respond with the Write Response Action containing:
  + **Write Response:** A list of paths and error codes every Write Request send on the Write Request Action.

The Write Request Action may be a groupcast, but in this case the Suppress Response flag must be set. The rationale is that otherwise the network might be flooded by simultaneous responses from every member of a group. To enable this behavior, the Path used in the Write Requests list may contain Groups and alternatively they may contain wildcards, but only on the Endpoint field.

**Timed Write Transaction**

In this transaction there where added a few steps to the untimed write transaction.

* **Timed Requests Action:**When the initiator starting a transaction, the action contains:
  + **Timeout:** how many milliseconds this transaction may remain open. During this period the next action sent by the Initiator will be considered valid.

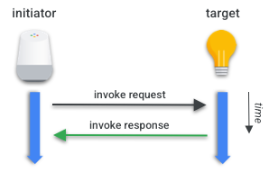
When the target receives the Timed Request Action, he must acknowledge with a Status Response Action. When the initiator receives a response with no errors, it will send a Write Request Action.

The rest of the actions are the same as the Untimed Write Transaction.

### Invoke Interaction

This interaction is used for invoking one or more cluster commands on a target. It is a similar way of the write transactions, Invoke transactions support also timed and untimed transactions.

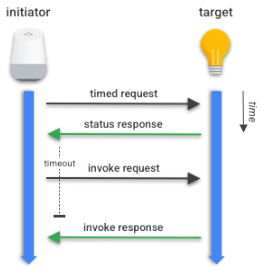
**Untimed Invoke Transaction**

Like in the untimed write transactions, the untimed invoke transactions contains two actions:

* **Invoke Request Action:**Similar to the Read Request Action and Write Request Action, in this Action the Initiator provides the Target with:
  + **Invoke Requests:** a list of paths to Cluster Commands, as well as optional arguments to the commands, named Command Fields.
  + **Timed Request:** a flag that indicates whether this action is part of a Timed Invoke Transaction.
  + **Suppress Response:** a flag that indicates whether the Invoke Response Action should be suppressed.
  + **Interaction ID**: an integer used for matching the Invoke Request Action to the Invoke Response Action.
* **Invoke Response Action:**After the target receives the invoke request action it will respond with an invoke response action that carries:
  + **Invoke Responses:** a list of command responses or status for every invoke request sent.
  + **Interaction ID:** a integer used for matching the Invoke Response Action to the Invoke Request Action.

**Timed Invoke Transaction**

Similar to the timed write transaction, the timed invoke transaction added the following step.

* **Timed Request Action:**A Initiator starts the Transaction sending this Action that contains:
  + **Timeout:** how many milliseconds this transaction may remain open. During this period the next action sent by the Initiator will be considered valid.

Once the Timed Request Action is received, the Target must acknowledge the Timed Request Action with a Status Response Action. Once the Initiator receives a Status Response Action reporting no errors, it will send a Invoke Request Action.

## Security

To protect the Matter fabric, there are several security features implemented to make sure that only trustworthy device can join the network and protecting the messages that are exchanged between the fabric nodes.

### Session establishment

This is used the exchange encryption keys that are required for a secure communication between nodes. It also involves mutual node authentication, which assures both nodes that they initiate communication with a trusted peer.

There are two session establishment methods available:

* **PASE**: Passcode-Authenticated Session Establishment  
  When using PASE, both nodes share the same secret in the form of 8-digit passcode. The shared secret is used by the SPAKE2+ algorithm to ensure a safe exchange of keys over non-secure channel. This process takes place when commissioning the device.
* **CASE**: Certificate-Authenticated Session Establishment  
  When using CASE, both nodes own Node Operational Certificates that chain back to the same root of trust. The NOCs are used by the SIGMA algorithm to ensure a mutual node authentication and a safe exchange of keys over non-secure channel. This process takes place while establishing the secured communication between nodes that are already commissioned.

Root of trust is a concept within Matter that is centered around a certification authority (CA), identified by Root Public Key (Root PK). The CA is a device tasked with issuing and assigning Node Operational Certificates (NOCs) or Intermediate Certificate Authority Certificates (ICACs). NOCs are installed during the Matter network commissioning by the commissioner together with Trusted Root CA Certificates.

### Message confidentiality and integrity

After exchanging the keys and establishing secure channel, the 128-bit AES-CCM algorithm is used to provide both confidentiality and integrity of exchanged messages.

A Matter message consists of the following elements:

* **Message Header:** Carries session and transport-related information.
* **Protocol Header:** Describes semantics of the message.
* **Payload:** Actual protocol-specific content of the message.

While the AES-CCM algorithm ensures the integrity of all three elements, only Protocol Header and Payload get encrypted. This is because Message Header contains fields, such as Security Flags and Message Counter, which are used to calculate the AES-CCM Nonce that is necessary to decrypt the remaining part of a message.

## Commissioning

When we want to add a device to a Matter network, we first need to commission it. You can think of it as the initial pairing. Although Matter communication in general uses Wi-Fi, Ethernet and Thread, commissioning can be performed over Bluetooth Low Energy (BLE).

The commissioning process takes place between a commissioner and a commissionee. Where the commissioner is the controller that is carrying out of the commissioning and the commissionee is the device that wants to join the fabric.

### Prerequisites

The controller must get onboarding payload from the commissionee to start the procedure. The data payload includes the following information, among other things:

* 16-bit Vendor ID and 16-bit Product ID
* 12-bit device discriminator
* 27-bit setup passcode
* 8-bit Discovery Capabilities Bitmask

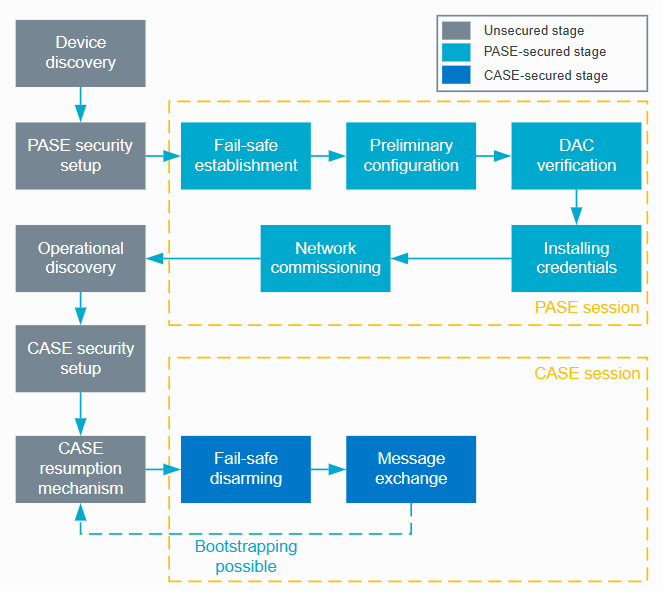
### Onboarding payload

The Onboarding Payload is the information used by the Commissioner to ensure interoperability between commissioners and devices. The commissioner can collect this information in different ways:

* **QR Code:** which you can scan using a mobile device with the ecosystem of your choice.
* **QR Code Payload:** which is an alphanumeric code that you can use in command-line tools. For testing purposes, it can be printed to the UART console or be shared using an NFC tag. This code is represented visually by the QR Code.
* **Manual Pairing Code:** provides the onboarding information as a sequence of digits that can be used with most Matter commissioners

### Commissioning steps

The commissioning process consists of steps shown in the figure below.



**Device discovery**

The commissioner discovers devices that can be commissioned onto the network. The commissionee needs to advertise their presence so the commissioner knows about their existence. This can happen using the following methods:

* **Bluetooth Low Energy:** This method is used especially if the node is being added to its first Matter fabric.
* **DNS-SD:** This method is commonly used if the node is connected to Ethernet or is already a member of a Wi-Fi or Thread network.

**PASE security setup**

The commissioner runs the Passcode-Authenticated Session Establishment (PASE) protocol, which is exclusive to the commissioning process. This protocol is used to establish the first session between devices that take part in commissioning. The session is established with a passcode provided out-of-band and that is used to derive encryption keys. This passcode is known only to the commissioner and the commissionee. There can be only one ongoing PASE sessions at a time.

**Fail-safe establishment**

The commissioner requests the commissionee to back up its original configuration. The fail-safe acts as a back-up, but it also starts a timer that sets a limit for the whole commissioning process. The timer is disabled with the disarming of the fail-safe at the end of commissioning.

**Preliminary configuration**

The commissioner reads the Basic Information Cluster attributes of the commissionee and its device type. It then configures the commissionee with regulatory information, such as location and country, and the current UTC time.

**Device Attestation Certificate verification**

The commissioner checks whether the commissionee is a certified Matter device. As part of this verification, the commissioner generates a random 32-bit attestation nonce and sends it to the commissionee, who should return the signed attestation information that includes the nonce. Usage of a nonce prevents replay attacks against commissioners. The commissioner then validates the attestation information.

The verification succeeds if the device is able to prove the validity and ownership of the mandatory Matter Device Attestation elements. If the validity and ownership cannot be proven, the verification fails. The commissioner can then either terminate or continue the commissioning procedure.

**Installing credentials**

The commissioner installs Node Operational Certificate (NOC) and Operational ID on the commissionee. The commissionee becomes the new node of the Matter fabric. The node is identified by a tuple consisting of the Root PK, Fabric ID, and Node ID. (While the fabric is identified by a tuple consisting of the Root PK and the Fabric ID.)

**Network commissioning**

The commissioner provisions the commissionee node with the operational network credentials, either Wi-Fi or Thread, and requests the commissionee to connect to the network.

**Operational discovery**

The commissioner discovers the commissionee node on the operational network using DNS-SD. This way, the commissioner learns the IP address of the node.

**CASE security setup**

The commissioner and the node use the Certificate-Authenticated Session Establishment (CASE) protocol to establish secure communication. The CASE protocol is in charge of exchanging NOCs to set up a session secured with a new pair of keys. The CASE connection is reset each time a device breaks the connection.

**Fail-safe disarming**

The commissioner requests the commissionee node to remove the stored configuration backup. This also stops the fail-safe timer.

**Message exchange**

The commissioner and the commissionee start exchanging AES-encrypted messages on the operational network.

## Fabric

Whenever a set of Devices in a network shares the same security domain, and thus allows secure communication between Nodes, this set is called a Fabric. Fabrics share the same Certificate Authority (CA) top-level certificate (Root of Trust) and within the context of the CA, a unique 64-bit identifier named Fabric ID. Thus the commissioning process is the assignment of the Fabric credentials to a new Node so it may communicate with other Nodes in the same Fabric.

### Multi-Admin

Nodes may also be commissioned on more than one Fabric. This property is often referred to as multi-admin. For instance, we may have a Device commissioned to both the manufacturer's Fabric and a cloud ecosystem's Fabric, with each Fabric handling a different set of encrypted communications and operating independently.

As several Fabrics may coexist, a Device might have several sets of Node operational credentials. However, the Node's Data Model is shared: the Cluster Attributes, Events, and Actions are common between Fabrics. Thus, although Thread and/or Wi-Fi credentials are set during the commissioning process, they are part of the Networking Operational Cluster, being shared between all the Fabrics and part of the node's DM, not the Fabric credentials.

## Matter Hub & Bridge

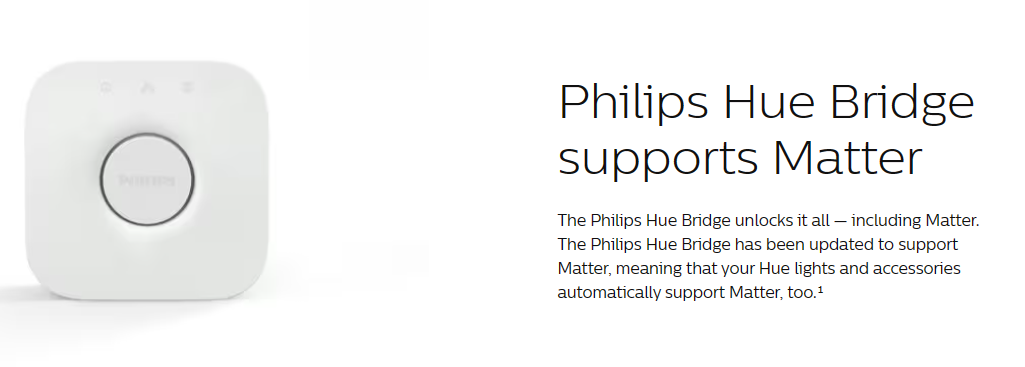
### Hub

A Matter hub essentially serves as the controller of the network, necessary for managing your Matter devices. It typically isn't tied to a specific ecosystem, allowing users the freedom to choose their preferred ecosystem. Additionally, the Matter Hub often functions as a Thread Border Router but not always. Example of a Matter Hub: Apple HomePod Mini



### Bridge

A Matter bridge act as a translator, it allows non Matter devices to connect and work with Matter devices in a fabric. With a Matter bridge Zigbee and/or Z-Wave devices can work with Matter devices, this allows more flexibility in an existing smart-home. Example of a Matter Bridge: Philips Hue Bridge.



## Matter production guide

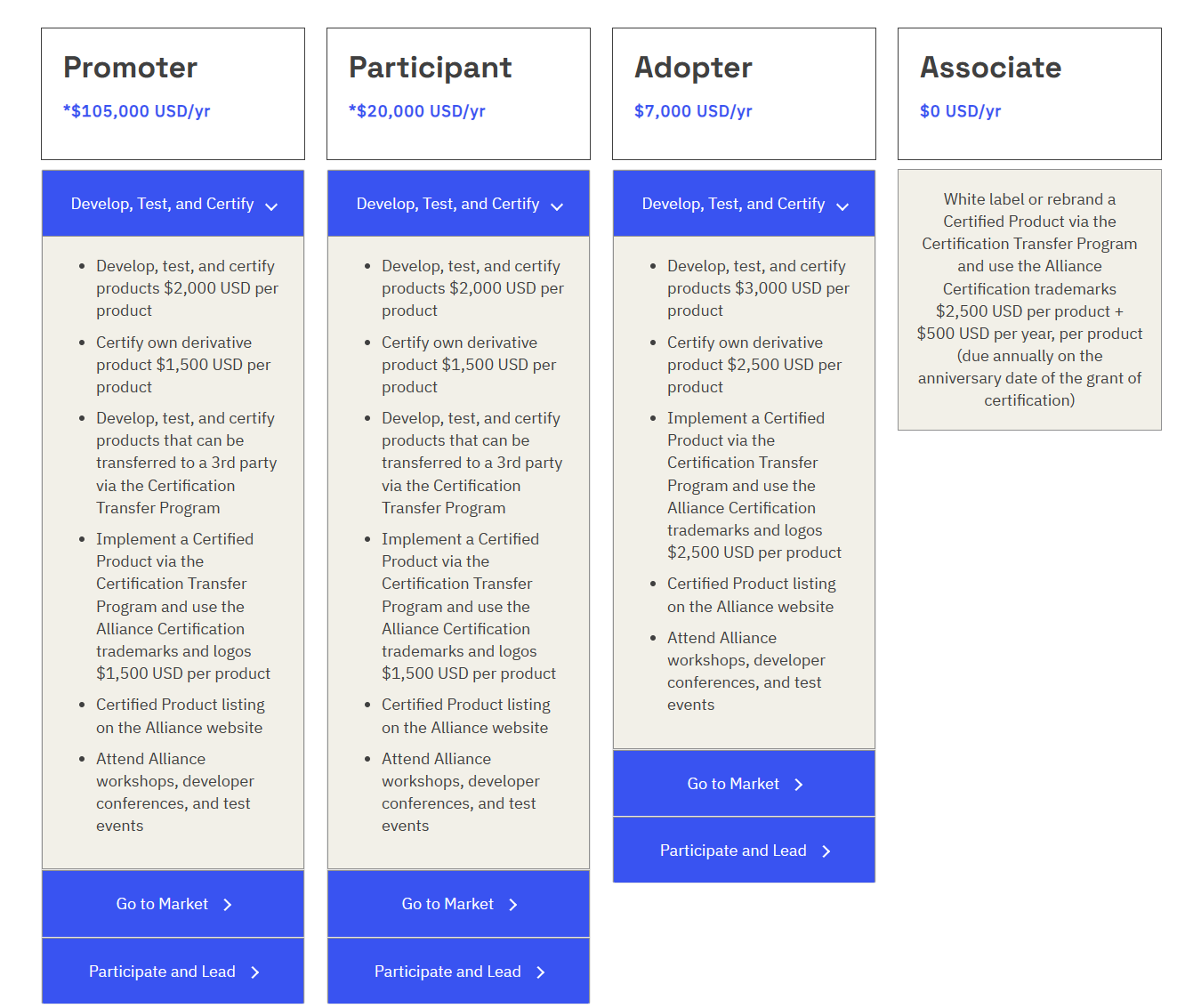
To develop a Matter end product, this topic lists the prerequisites and next steps to facilitate your production journey through Matter.

### CSA Member

Associated membership is necessary to get a product certified by a CSA-approved testing facility. As a member you will receive membership perks like:

* Official resources to assist you in developing Matter products.
* Authorization to contribute to the [Matter Github repository](https://github.com/project-chip/connectedhomeip).
* Once approved, CSA will reserve a unique Vendor ID (VID) chosen by your organization. This VID will be needed to provision your device. Your unique VID will be added to the [CSA Distributed Compliance Ledger](https://webui.dcl.csa-iot.org/) (DCL).
* Matter [Certification tool access](https://csa-iot.org/certification/tools/certification-tool/). This allows you to evaluate your product for certification before the official certification process

Become a member at [CSA Membership](https://csa-iot.org/become-member/). You can see a list of the different memberships offered at CSA.



### CSA Certification

Once a product is ready for production, the product must be certified by a CSA-approved testing facility to certify compliance with the Matter Standard. Check the [CSA Certification process](https://csa-iot.org/certification/why-certify/) for more information. For additional resources on the certification process, refer to the [Certifying your Matter Product](https://community.silabs.com/s/article/Certifying-your-Matter-Product-an-Overview?language=en_US) article by Silicon Labs.

## Develop with Matter

Matter is a very complex protocol and there are different ways to develop with Matter.

### Official Matter SDK

The CSA provides the official Matter SDK that can be find on <https://github.com/project-chip/connectedhomeip>. Documentation, that contains guides to develop with VS Code, to build clusters, controllers, tools, etc. can be found on [https://project-chip.github.io/connectedhomeip-doc/index.html](https://project-chip.github.io/connectedhomeip-doc/index.html%20). On <https://github.com/project-chip> there are very useful repo’s that you can use when developing.

### Manufacturer SDK

Manufacturers like Silicon Labs, NXP, and Espressif have their own SDKs developed on top of the official SDK. Depending on which product you are going to use, you have the option to use their SDKs with VS Code or with their IDEs. Documentation is available, just search “Espressif matter SDK” for example, everything you need to know is available.

### Certification test

Before certification you can use the certification tool available at <https://github.com/project-chip/certification-tool>. With this set of tools you can test your device before certification.

# Silabs Gecko SDK V4.4.0 – SSv5

Because this project uses a Silicon Labs development board (EFR32MG24) we will also use their IDE and Software Development Kit (SDK).

Simplicity Studio v5 is the IDE being used. It is a eclipse based IDE that is optimized for all Silicon Labs technologies, SoCs and modules. It provides access to device-specific and SDK resources, software and hardware configuration tools. It also contains a various set of advanced tools what can be used by their products.

Gecko SDK 4.4.0 is a SDK designed for developing applications for Silicon Labs 32-bit IoT products. It integrates various wireless SDKs along with the Gecko Platform into a single package. This version contains Matter version 1.2, this is the current Matter version at time of writing.

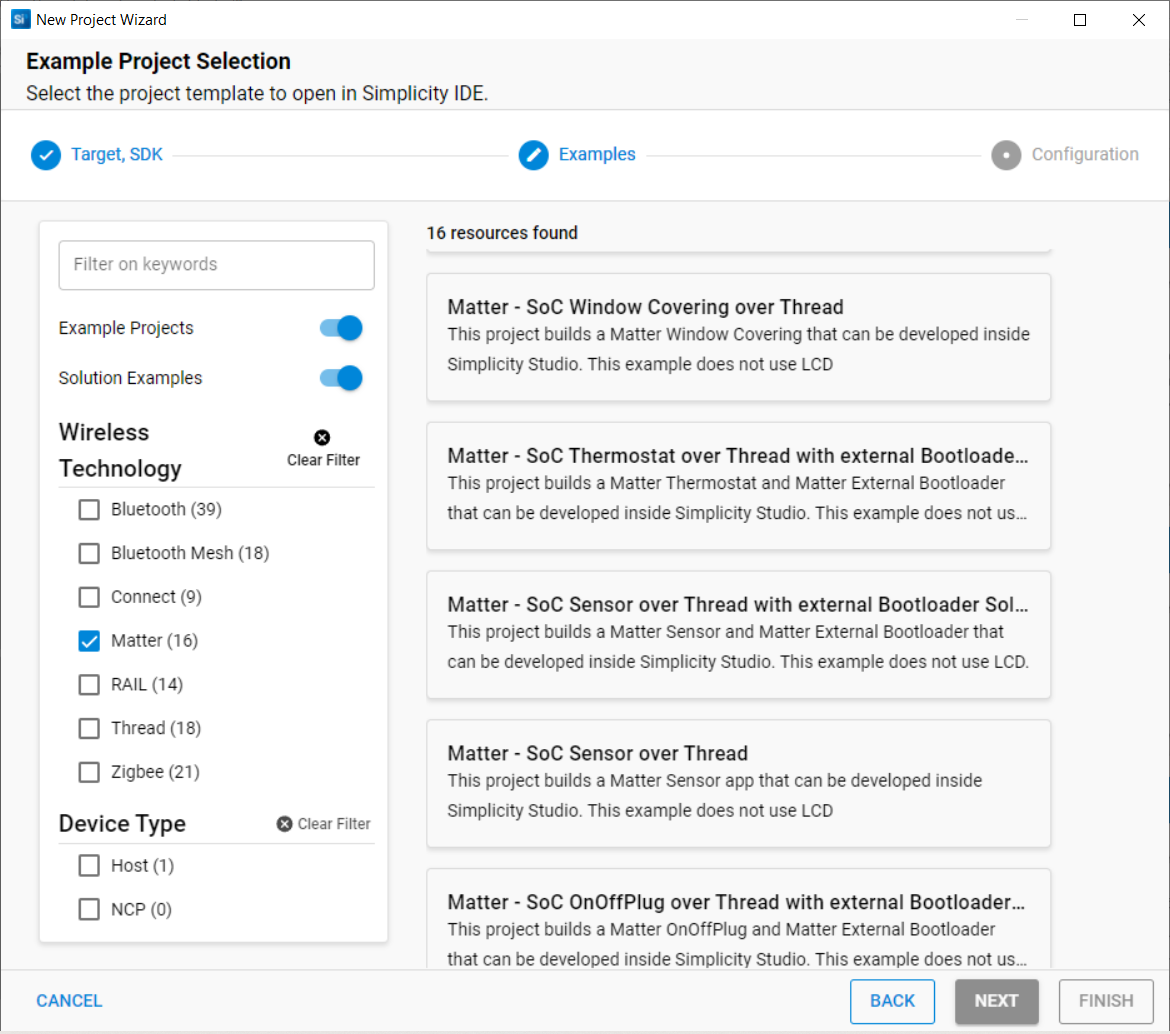
## Installation

First install the Simplicity Studio 5 IDE. When installed you will be asked to install SDK’s. To use Matter you have to install the Silicon Labs Matter – 2.2.0 (This is Matter version 1.2) extension.



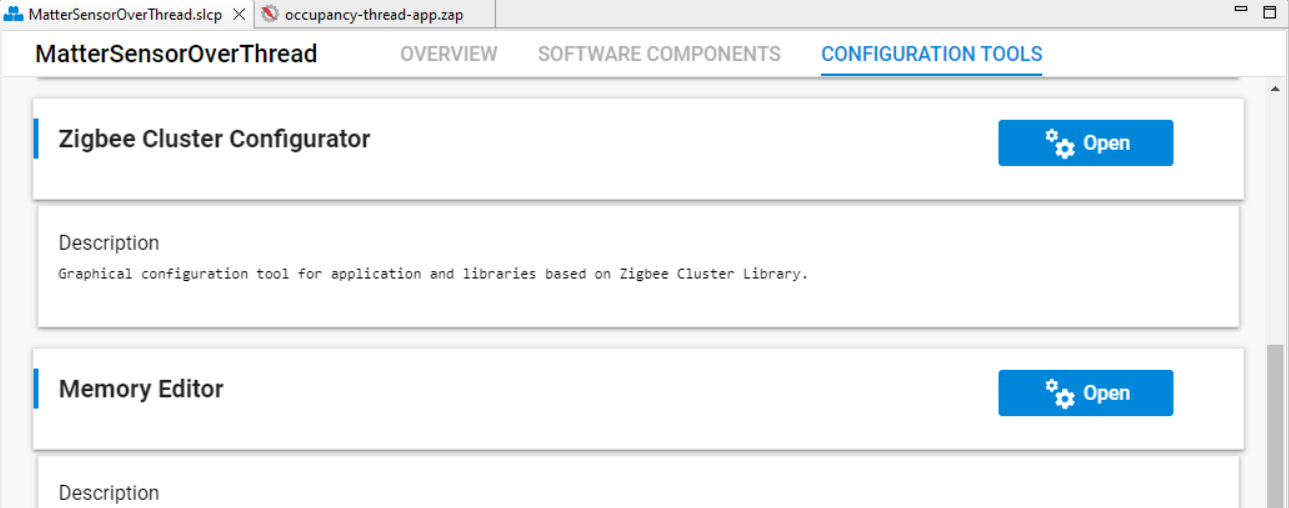
## Matter Examples

A good thing about the IDE and the SDK being used is that there are several Matter examples available that can be tested and modified. This can be created by the project wizard, when selecting the target board, device and the SDK, you will go to the next tab examples. Under wireless technology you can select Matter and all the available examples will be shown.



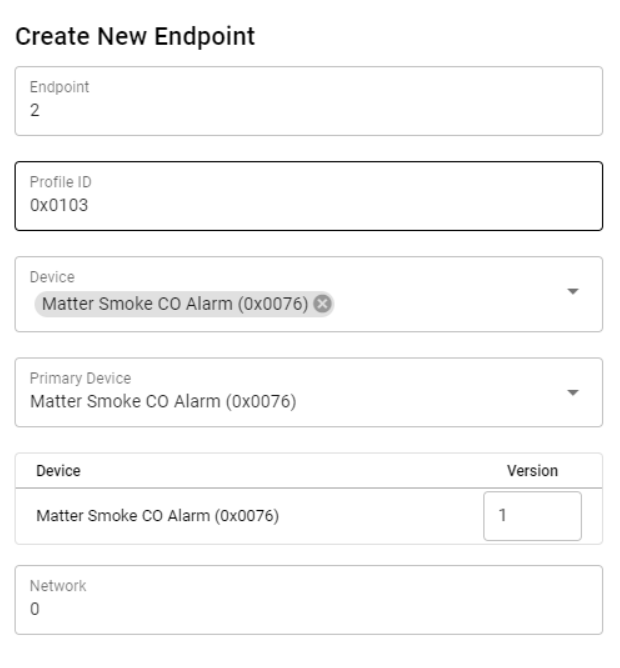
## Zigbee Cluster Configurator

When a Matter example needs to be modified to add an endpoint, for example. This can be done by the Zigbee Cluster Configurator (ZCL), it can be found in the project file (.slcp) under configuration tools.



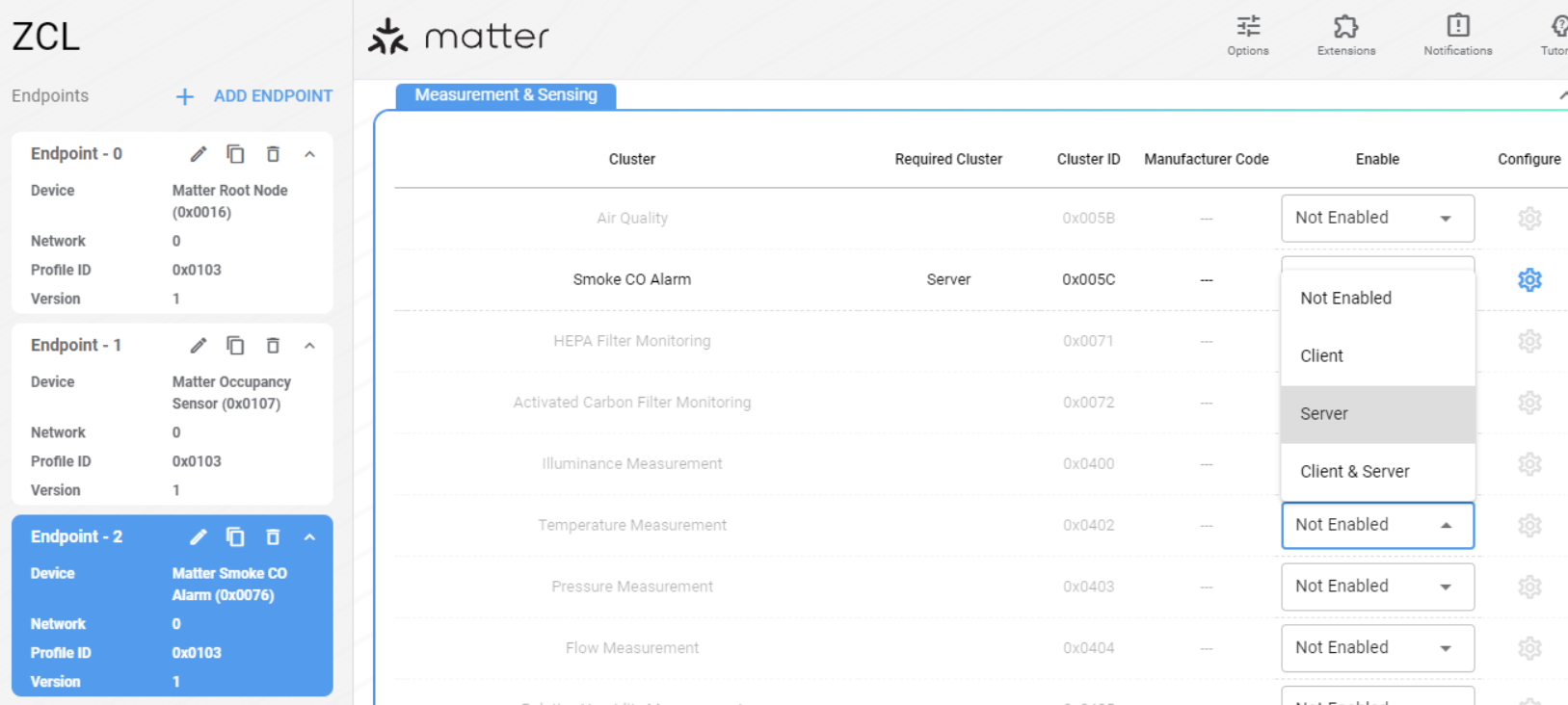
### Adding endpoint

When the configurator is opened, you can add a new endpoint. To do this click on ‘Add Endpoint’, a window should appear. In that window you can give this endpoint an ID and you can select a device type under ‘Device’ e.g. ‘Matter Smoke CO Alarm’. Click then on ‘Create’, the endpoint is added.

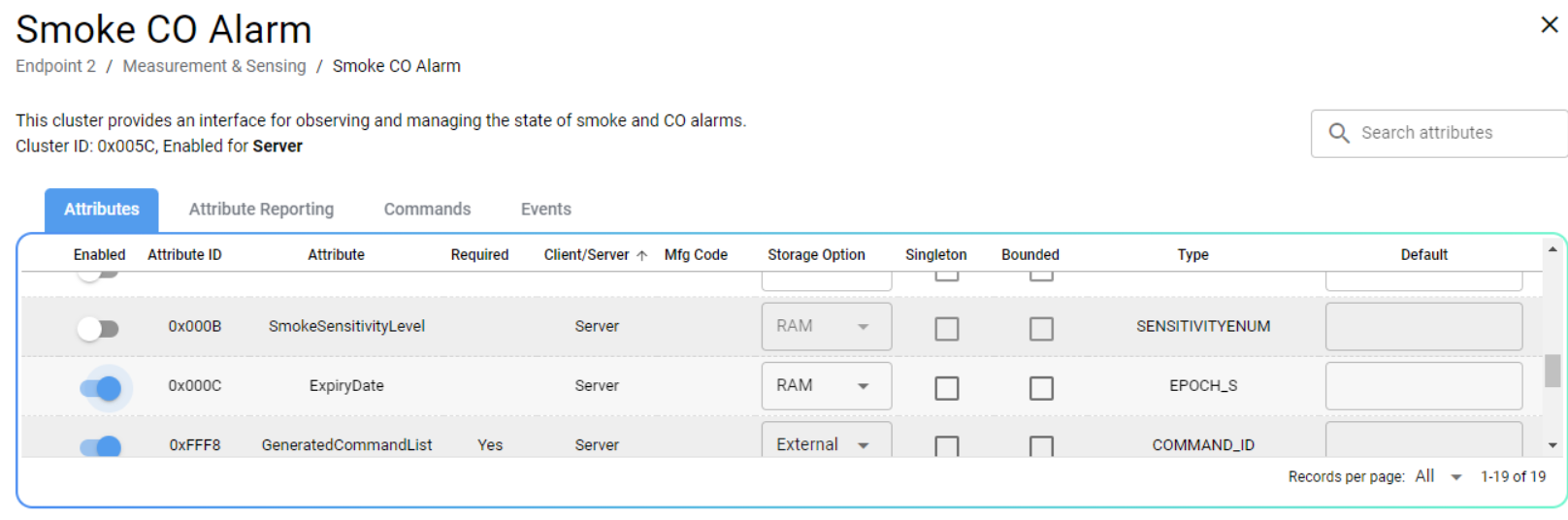


### Modifying endpoint

It’s also possible to modify an endpoint and add or delete clusters. For example, we are creating a smoke sensor but we want also measure the temperature (not a mandatory cluster), first check the application cluster specification so you know what cluster you need. Than we can add this cluster to this endpoint. To do this click on the endpoint and at the right side you can see the same tabs as in the application cluster specification. Click on ‘Measurement & Sensing’, you can see ‘Temperature Measurement’, click on ‘Enable’ and select server. We chose server because only want to read the temperature. By doing this, the cluster is added.



By clicking on the configure button of the cluster, it is possible to enable or disable Attributes. For example the smoke sensor can only be used for 10 years because the life cycle of the sensor. There is an attribute ‘ExpiryDate’ in the ‘Smoke CO Alarm’ cluster that we want to use to achieve this. Click on the configure button, search for the ‘ExpiryDate’ Attribute and enable it. It is also possible to choose the storage location.



In this way you can modify your endpoint as you wish. New clusters needs to be installed in the software components, normally when using the ZCL this is automatically done.

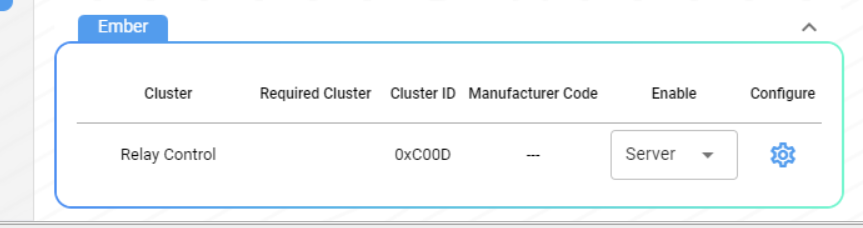
### Custom Cluster

when a Attribute and/or Command is not available in an existing cluster it is possible to add a custom cluster. There are several options to do that but the best way is to use sample-extensions.xml located in user\SimplicityStudio\SDKs\gecko\_sdk\app\zcl\sample-extensions.xml OR [gecko\_sdk/app/zcl/sample-extensions.xml at 911f6cdefccbae03bc66e8c790ceb7e67ca07417 · SiliconLabs/gecko\_sdk (github.com)](https://github.com/SiliconLabs/gecko_sdk/blob/911f6cdefccbae03bc66e8c790ceb7e67ca07417/app/zcl/sample-extensions.xml) as a reference where everything is explained.

After writing a cluster configuration go to ZCL and then to extensions. A window should appear, you can browse your file and add it.

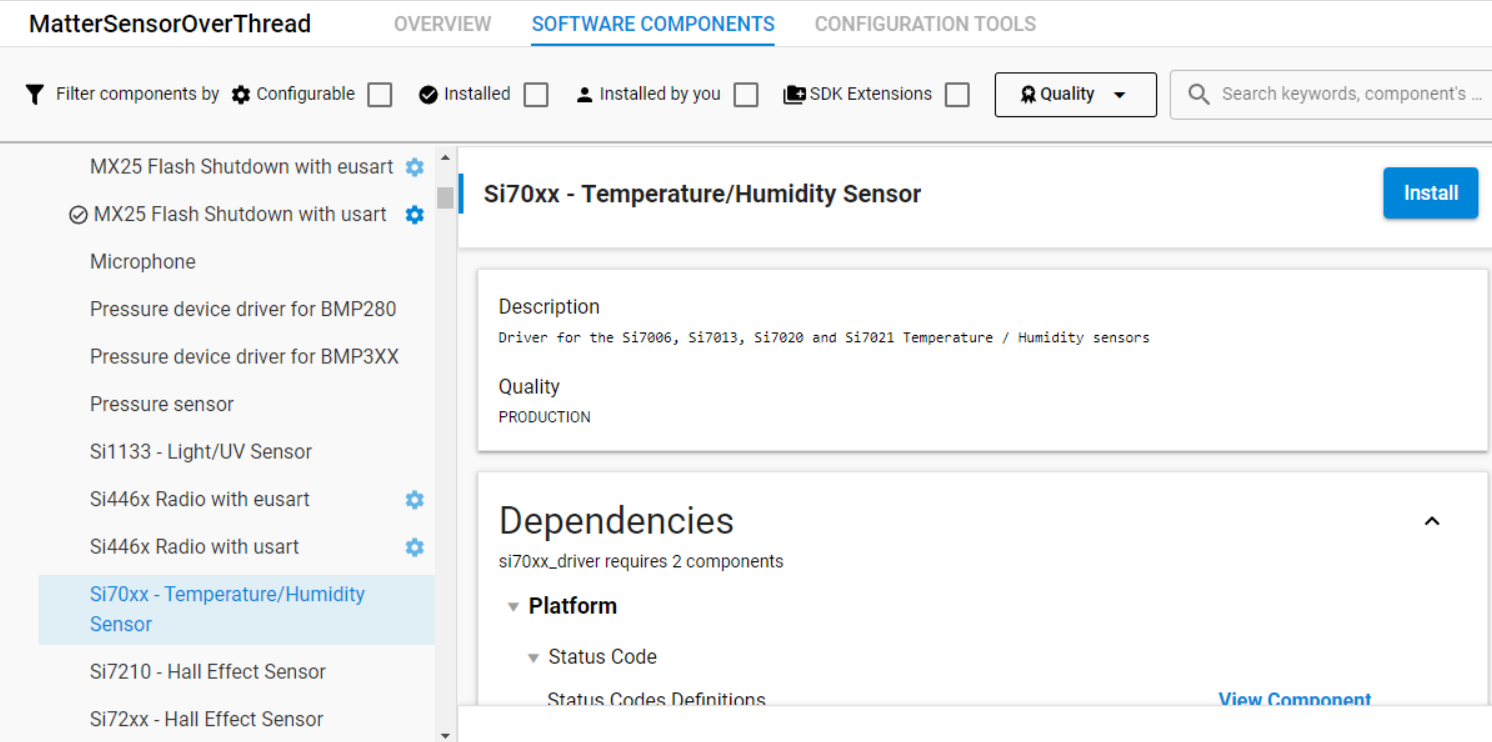


After this is done your cluster should appear. Enable it by Server or Client or both, save the file and the necessary changes will be automatically done.



## Software Components

Projects are configured by installing and uninstalling components, and configuring installed components. On the left side you can see all the components in categories and on the right all the details about the selected component. For example I want to measure the temperature with the “on board sensor” for the cluster that I have configured in the ZCL, this can be find under ‘Board Drivers’ on the left side, on the right side this can be installed.



# Setting up a Matter Hub

As mentioned in section 4, I have used a Raspberry Pi as Matter hub in this project. For ease of use I chose a pre-built Matter hub developed by Silicon Labs. the Matter Hub consists of the Open Thread Border Router (OTBR) and the chip-tool running on a Raspberry Pi. The chip-tool is a Matter controller that allows easy development and testing by using the command line.

## Silicon Labs Hub image

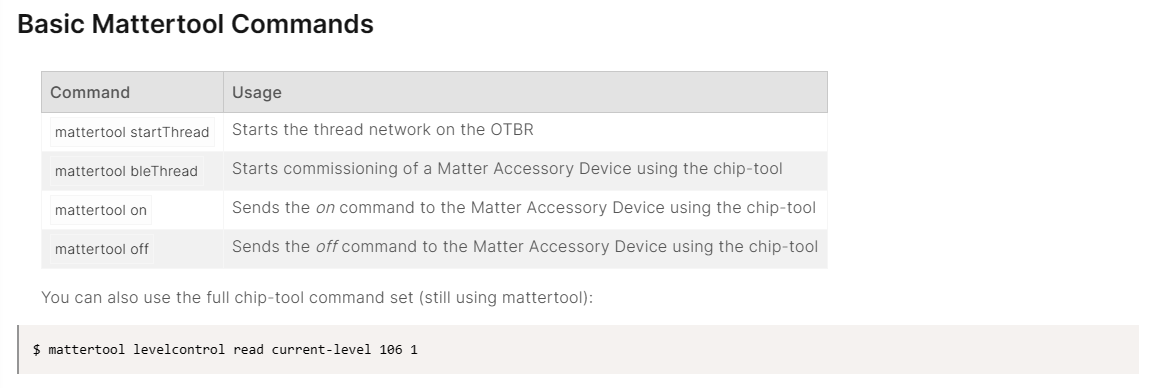
Silicon Labs has developed a Raspberry Pi image combining the OTBR and chip-tool that can be downloaded and flashed onto an SD Card, which is then inserted into the Raspberry Pi. In the docs of Silicon Labs you can find all the instructions and the link to the image. Docs and image can be found here <https://docs.silabs.com/matter/latest/matter-thread/raspi-img>

## Radio Co-Processor

The Radio Co-Processor (RCP) is a Thread device that connects to the Raspberry Pi via USB. To flash the RCP, connect it to your laptop via USB. Thereafter, it should be connected to the Raspberry Pi via USB as well. Prebuilt RCP images developed by Silicon Labs and instructions are located here: <https://docs.silabs.com/matter/latest/matter-thread/matter-rcp>

## Chip-tool

The chip-tool is a Matter controller implementation that allows to commission a Matter device into the network and to communicate with it using Matter messages, which may encode Data Model actions, such as cluster commands. Within the prebuilt Matter Hub image, silicon labs has simplified this.



As they say, the full use of the chip-tool command set is still available, Just replace “./chip-tool” with “mattertool” because this is an alias and makes it easier. More information about commands is located here:

<https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/chip_tool_guide.md>

# Matter Tests

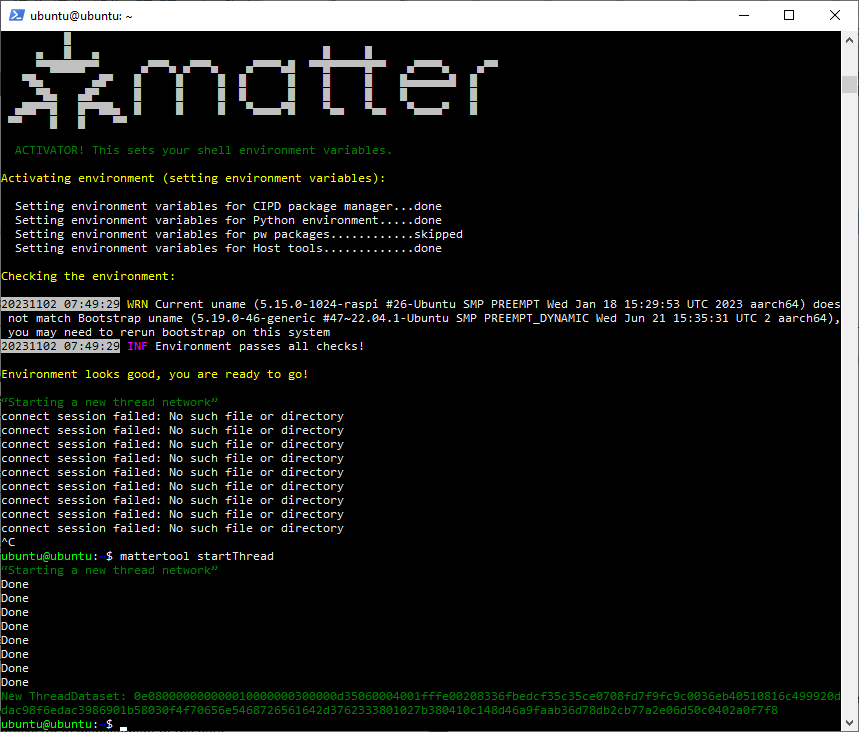
## Simple on/off device

The first test was a simple on/off device to see what the protocol does.

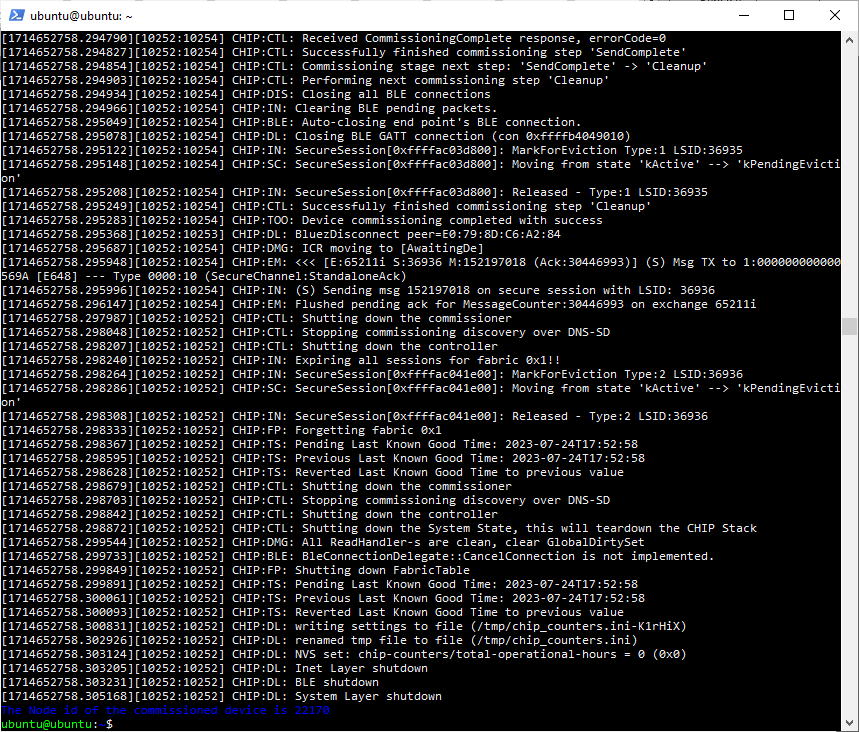
The first step is to setup a Matter Hub. The seconds step was flashing the Radio Co-Processor (RCP) into the board. This was very easy with Simplicity Studio v5. Open SSv5, connect the board its auto detected. Then under “Example projects & demos” select “Thread” in Wireless Technology and then click on create. After the project is created, flash the project into the board. The third step was flashing the demo “SoC Lighting over Thread” into the board. The demo is located under “Example projects & demos” under the section Matter. The demo can be directly flashed into the board.

After setting up the Matter Hub and flashing the demo, we'll need to login to the Matter Hub and start testing the protocol using the chip-tool.

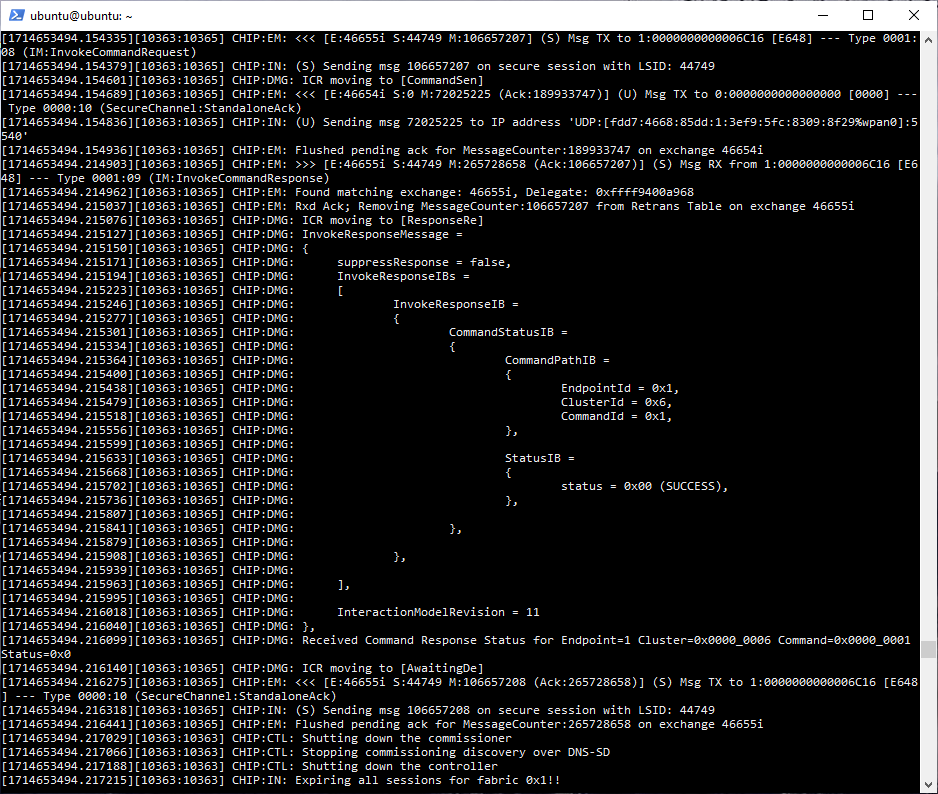
First we need is to start the Thread Network using **mattertool startThread** command. This creates a new Thread Network.



When the network is created, the device can be commissioned by using the command **mattertool bleThread**. When the device is successful commissioned it gives the Node-id.



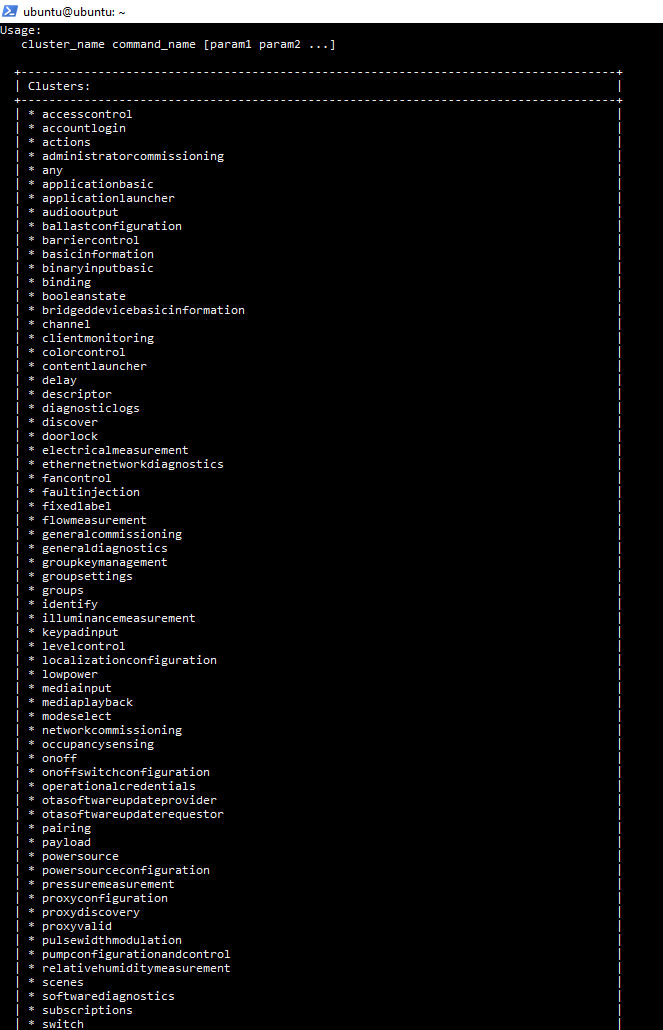
After that we can control the light using the command **mattertool on** and **mattertool off**. With this command the devices changes there on/off state to on or off.



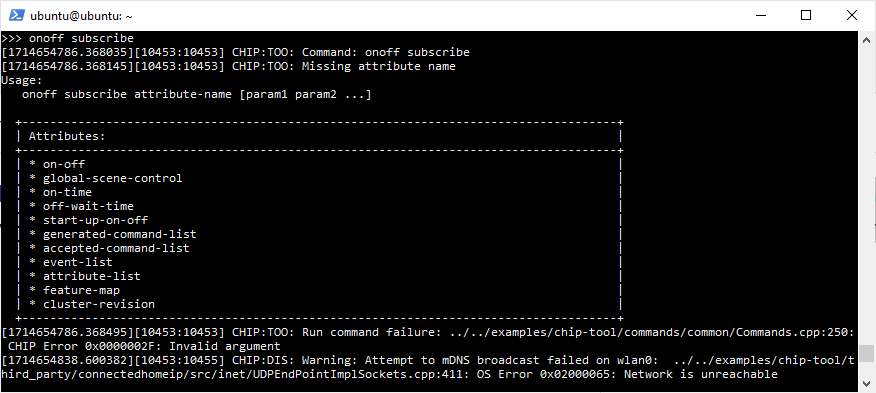
With this example we can see that the command **mattertool on** uses the invoke interaction which is explained in section [5.6.4](#_Invoke_Interaction). As we see in the cluster specification ClusterId 0x6 is the on/off cluster and the CommandId 0x1 is the on command.

So now I can control the light using this commands but I want to know if I can monitor this. After a little research in the Matter docs under chip-tool guide, with the subscribe interaction in mind, I found how to do this. In this guide there is described how to subscribe to an attribute and that was exactly what I needed. When subscribing to an attribute every change displayed in the command line. To subscribe we have to follow a few steps.

1. Start the Chip-tool in interactive mode using **mattertool interactive start** after this command we can see “>>>” this means that we are in interactive mode. When the interactive mode is started it shows a list of all available clusters.



1. We want to monitor the on/off cluster. Use the command “**<cluster-name of choice> (onoff) subscribe**”. The list of all available attributes for the cluster will appears.



1. I need the On/Off attribute to monitor the changes. The base command that I used is:

**<cluster-name> subscribe <argument> <min-interval> <max-interval> <node\_id> <endpoint\_id>**

****

The parameters of this command:

<cluster-name> is the name of the cluster.

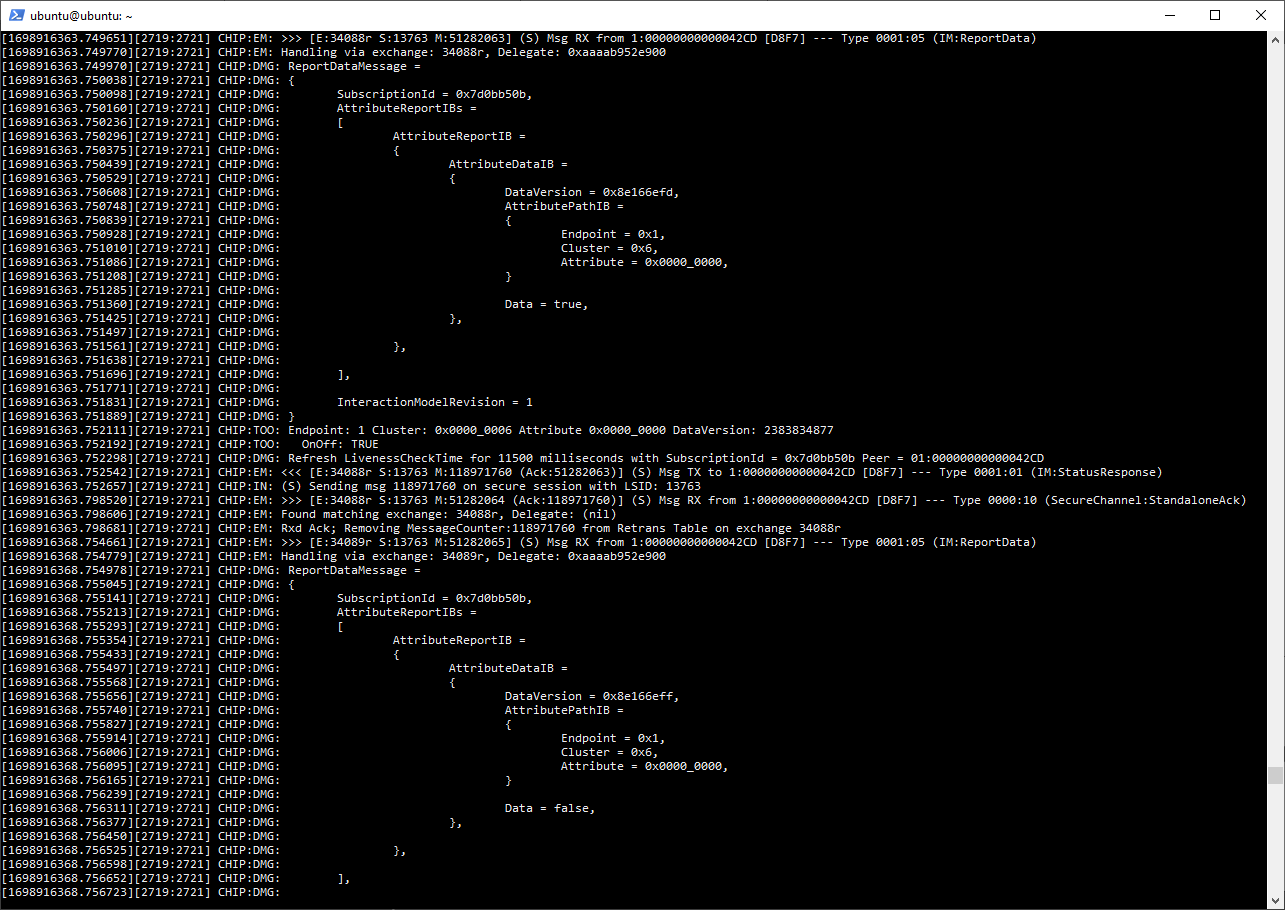
<event-name> is the name of the chosen event.

<min-interval> specifies the minimum number of seconds that must elapse since the last report for the server to send a new report.

<max-interval> specifies the number of seconds that must elapse since the last report for the server to send a new report.

<node\_id> is the user-defined ID of the commissioned node.

<endpoint\_id> is the ID of the endpoint where the chosen cluster is implemented.



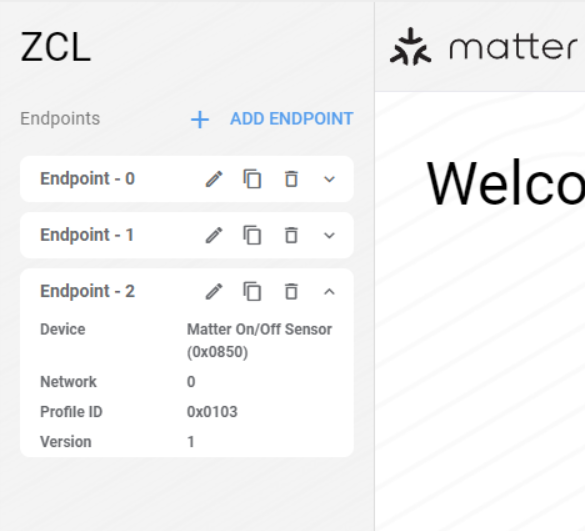
So now every time I push the user button and the led changes, I get an update.

## Level control test

**Before continuing this section please read** [10.1 Accelero sensor problem](#_Accelero_sensor_problem)

To test level control I started with the MatterSensorOverThread example and modify the endpoints in the Zigbee Cluster Configuration tool.

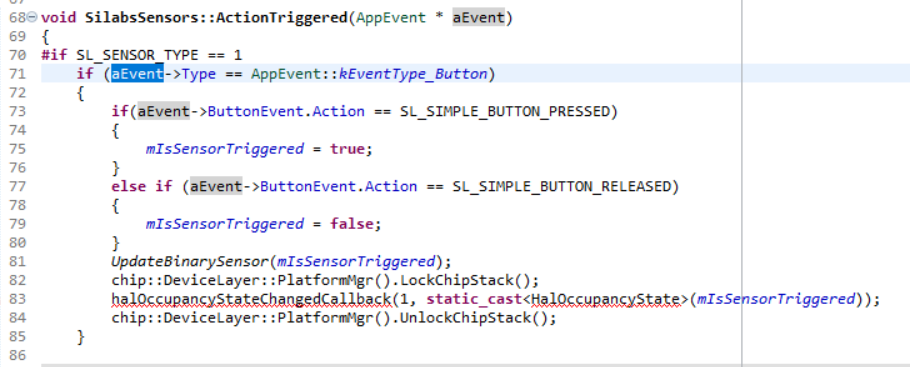
First step was the Zigbee Cluster Configuration tool in Simplicity Studio v5. I added a new endpoint with the device type “Matter On/Off Sensor”. I want to use this cluster to store the data in the future.



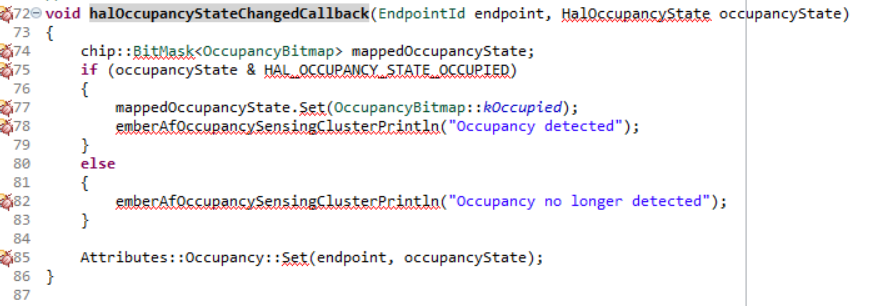
After that I enabled the level control cluster as a Client & Server in the General tab, actually we need only the Server cluster but I select both to test this.



In the example project I found how they used a sensor en stored the data into an Attribute of a Cluster. In the SensorsCallback file we can see what they do when a button is pressed to activate the sensor.



We can see that they call another callbackfunction “halOccupancyStateChangedCallback”. This function is located in the “occupancy-server.h” file in the clusters folder.



On line 85 they set the state by using Attributes::Occupancy::Set(). This is what we need for our sensor. We are going to do the same for the level-control. The only thing that needs to be done is to set the current level attribute, which can be done with the following function.

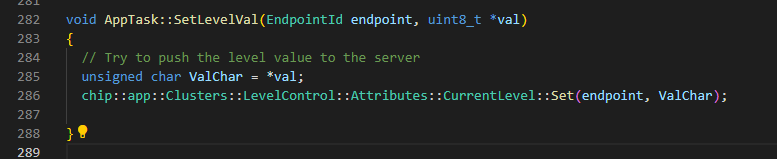
chip::app::Clusters::LevelControl::Attributes::CurrentLevel::Set(endpoint, \*val);

To use the set function to change the value of the level control attribute we have to add:

#include *<app-common/zap-generated/attributes/Accessors.h>*

To test this properly the intention is to create a virtual level that increments every x times from 0 to 254 and back. With this virtual level, I can do the monitoring from the command line. To do this I had to make some modifications in the AppTask.c file.

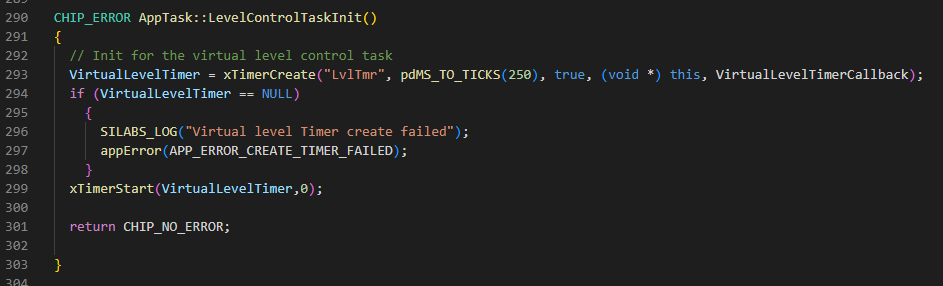
First I created a function that sets the level of a specific endpoint:



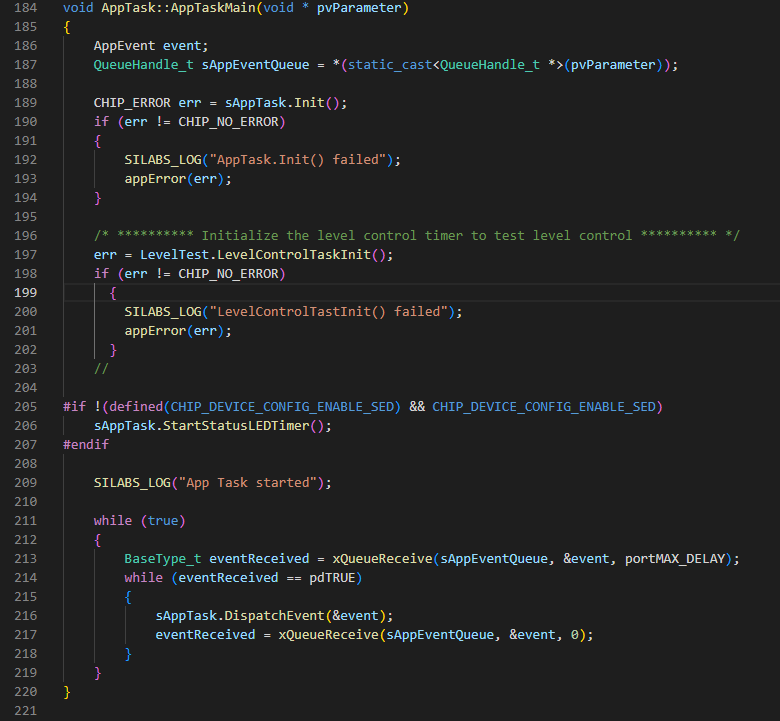
Then I created a callback function that can be called every x times to set the value. This will increment or deincrement the value from 0 to 254 and after that it will lock the chip stack and call the SetLevelVal function.



To make sure that the callback function is called every x times I had to create a function that initialize, sets the callback and starts the timer.



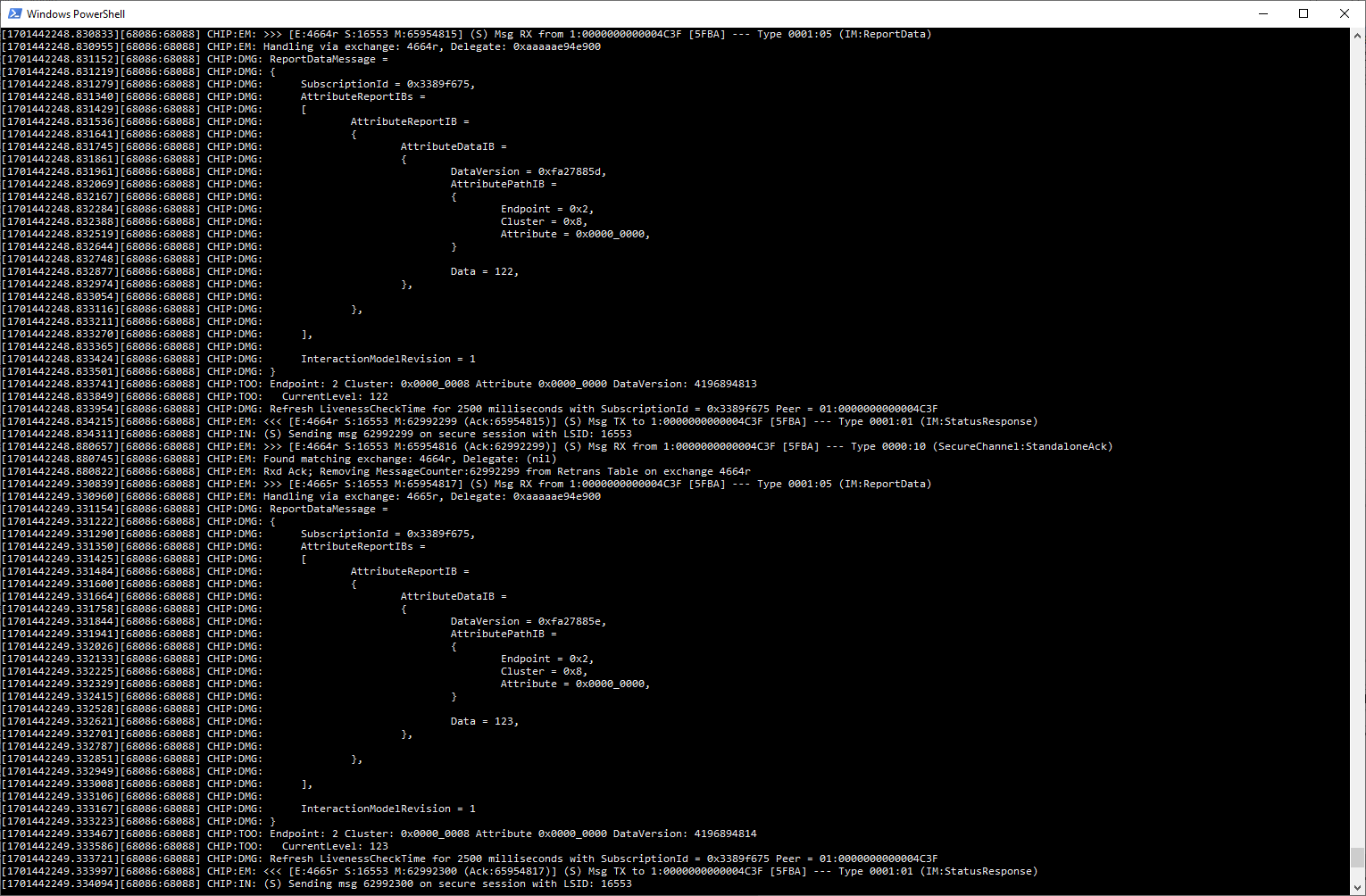
The only thing that needs to be done is to implement this in the main task.



### Capture data using chip-tool

To capture the data we can use the interactive function in the chip-tool and subcribe to the endpoint. Every time the data changes it will be displayed. The following commands is being used:

1. Mattertool interfactive start
2. Levelcontrol subscribe current-level (min-interval: I used 0) (max-interval: I used 1) (nodeId) (endpointId)



With this output I can see that the level control is working.

## Terminal output test

In the end we want to use the terminal output to control the game. In the previous tests the data is successful shown in the terminal but it shows a lot of information I don’t need. I tried to use grep but because the chip-tool interactive mode is a process, this was very difficult.

**HOE KUNNEN WE DEZE DATA FILTEREN -> in pybreakout v1.1 test.py werken met subprocess???**

# ICM-20689 Inertial sensor

The ICM-20689 is a 6-axis inertial sensor that combines a 3-axis accelerometer and a 3-axis gyroscope. This sensor is built in the development board. There is a software component available in the Gecko SDK that contains the driver for this sensor. The datasheet of the sensor contains all the information and registers needed to configure and use the sensor.

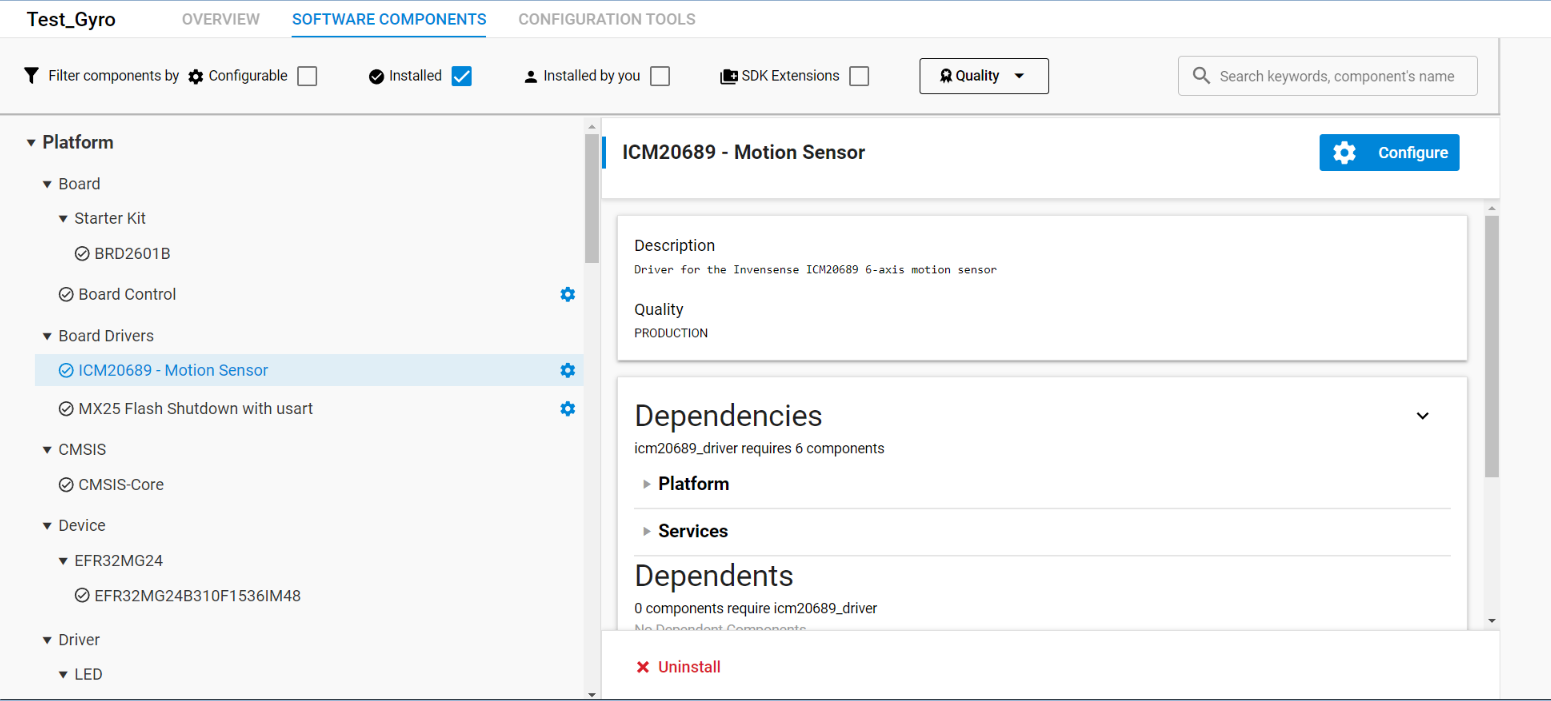
For this project, the gyroscope, as well as the accelerometer, can be used, but there are two differences. The gyroscope measures the speed (°/s), so when moving the sensor give’s a value how fast the sensor moves. When the sensor stops moving we get a “0” value.

The accelerometer measures the gravity (g). The sensor holds the value at its position even if the sensor stops moving. It is easier to use the accelerometer because we can assign the position to a certain value.

## Using the ICM-20689 driver

The driver contains all the functions needed to use the sensor.

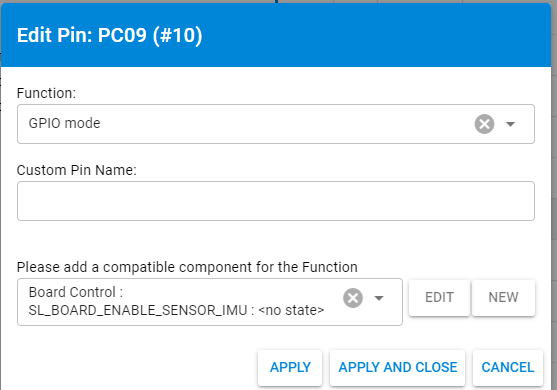
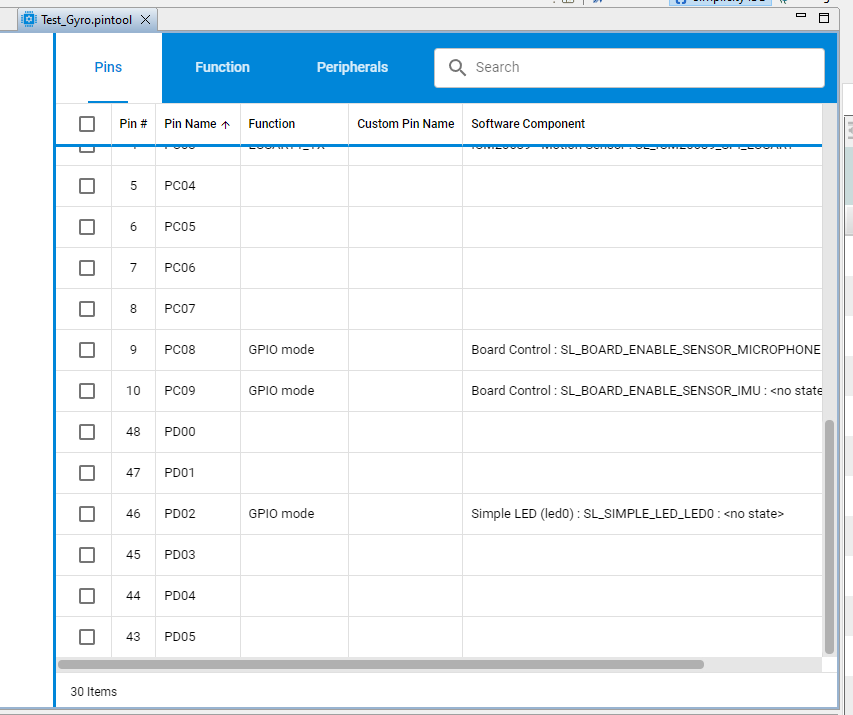
Silicon Labs has a driver for this sensor that contains all the functions needed to use the sensor. The driver can be used by installing the software component in simplicity studio. Open the project file (.slcp) and select software components. Search for ICM20689 and install the component.



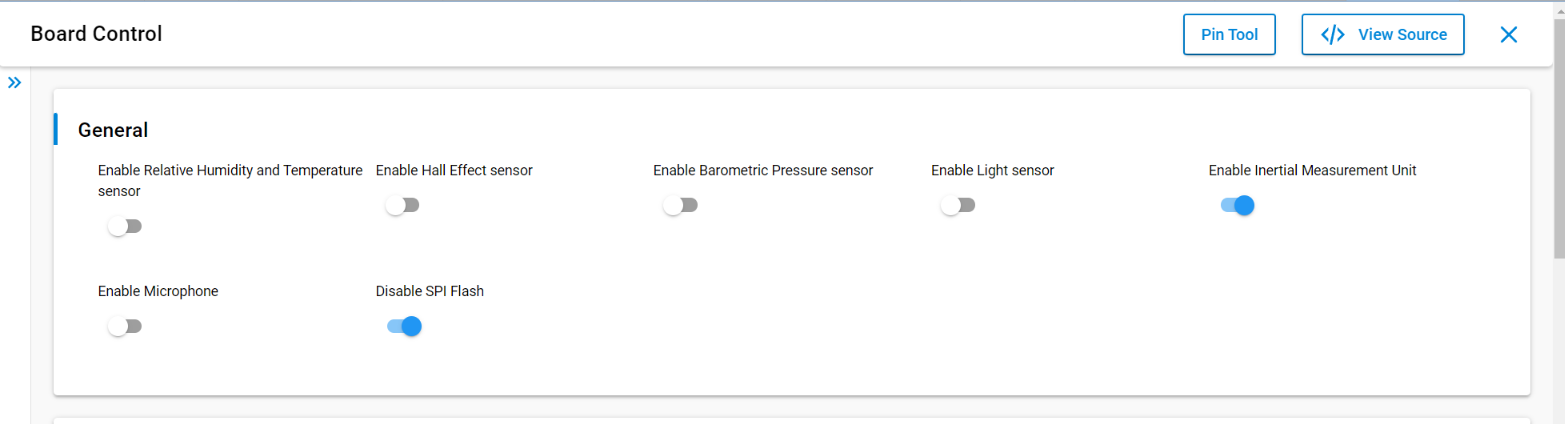
After installed the driver set PC09 to high to enable the sensor, refer to [2.3 ICM-20689 6-Axis sensor](#_ICM-20689_6-Axis_sensor). There are two options to do this, by using the pin tool or manually.

* By using the pintool:

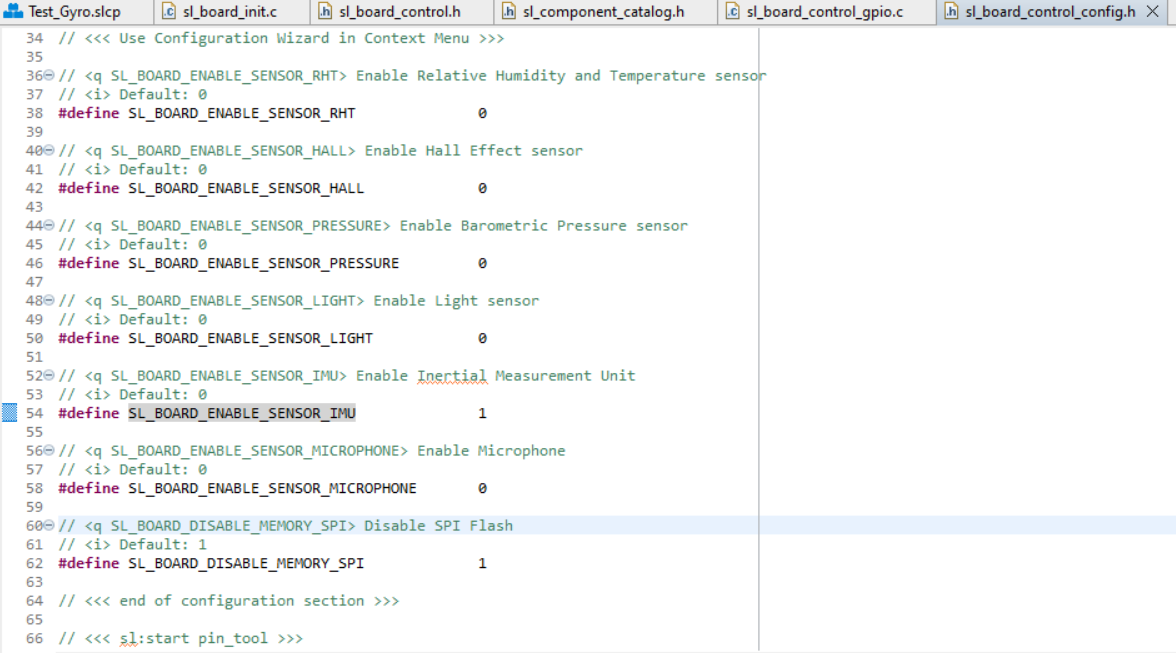
Select PC09, then click on edit



After that click on Enable Inertial Measurement Unit.



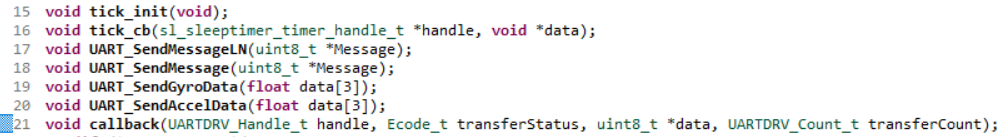
* Manually:  
  To enable the sensor go to the folder config in the project and open sl\_board\_control\_config.h and set the define SL\_BOARD\_SENSOR\_IMU to 1.



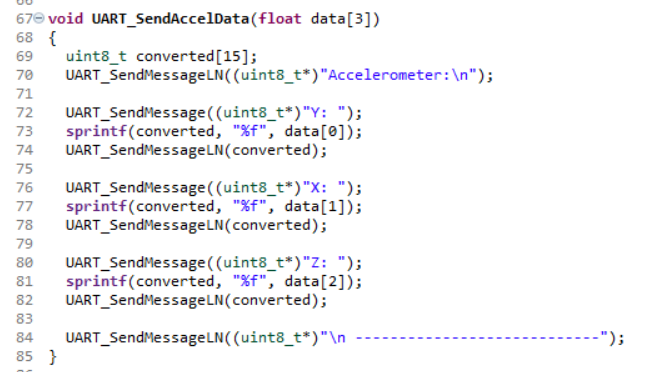
## Tests

### Raw data

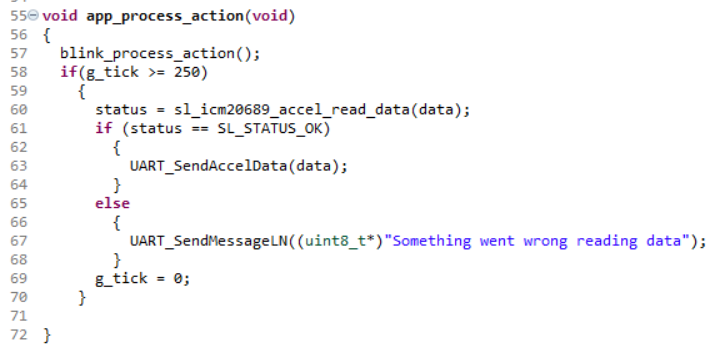
To test the motion sensor I have used a simple bare-metal project using the Gecko SDK. I added UART to send the sensor data to a terminal. To use UART I had to install the software component for this. I also created a tick timer to send the data with an interval of x ms.I created a file called message.c with some useful functions.



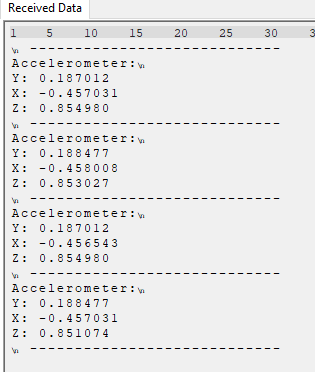
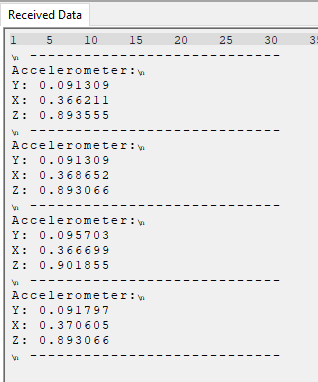
Tick\_init starts a periodic timer and the tick\_cb gets called every time the timer period times out. The UART\_SendMessageLN sends a string with linefeed ad the end while UART\_SendMessage does not. The functions UART\_SendGyroData and UART\_SendAccelData will send the data coming from the sensor. This function expects a array so we can send the 3-axis at once and uses also the SendMessage functions to send the data. It also uses sprintf to convert the float value to a character.



In the application function “app\_process\_action” the tick timer increments every 1ms the variable g\_tick and every 250 milliseconds this block of code gets executed. So every time this gets executed we read the sensor and store the data in a float array that will be send with the UART\_SendAccelData function.



When I run this code and tilt the board first to left and then right we get this data.

So the most interesting value to read is the X axis. We can see that when the board is tilt to the left we get a negative value and to the right we get an positive value. So that is what we going to use.

### Converted data

The floating point value that the sensor gives is not usable in the Matter application. In the Matter application I use the attribute Level Control that stores a uint8 datatype. So we need to convert the floating point variable to a uint8 variable.

MEER UITLEG EN SCREENSHOTS

# Issues

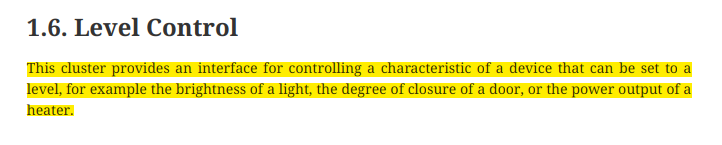
In this section I’m going to tell all about the problems I faced during the research and the possible solutions I found.

## Accelero sensor problem

When doing the research of the Matter protocol I was facing a big potential problem for the project. The protocol has pre-build “device type’s” these are officially defined and they are not customizable. The only thing we can adjust is a custom cluster. A cluster contains a set of functionalities. This looks like a major problem because we want to read a sensor that is not included into the officially device type’s and we DON’T want to manipulate the protocol.

### Possible solution

* Digging deeper into the Device Library, I noticed that there's actually one device type we can use, On/Off Sensor. The On/Off sensor includes an optional cluster Level control. Level control can be anything and is not something specific.



This can solve the problem on the side of server so we can manipulate the current-level attribute at the server side, our sensor.

* Another solution that I found was to create a custom app, combine this with the device type and the Level control cluster.

### Solution

There is no good solution, I want to do something unintended with the protocol. First we had to decide whether to continue with this project or not. We decided to continue because we want to know what the protocol is capable of. Then I decided to use a device type that best fit my needs and add the level control to it. The On/Off sensor fits best and the level control cluster is optional so I decided to use this.

## Connection between matter device (using chip-tool) and the “game”

First I thought that the interaction between the matter protocol and the game can be done by the read commands or to subscribing to an attribute of a endpoint (for example the on/off attribute that holds the on/off state) using the chip-tool. While doing some [tests](#_Simple_on/off_device) it seems that we get a lot of information and not only the state of a device, for example a ‘True’ value as the button is pressed. So getting that values from the chip-tool in the command line isn’t that simple. I don’t even know it is possible to filter everything so I get only the information I want. For controlling the game I need only the value of the attribute, whether it is two on/off “devices” one for left one for right or it is a level control.

### Possible solution

Maybe to make this work I can built another device that can be controlled by the game controller and then use gpio’s (with or without pwm) that can be connected to the raspberry PI and then read these pin’s so we can control the breakout game with this pins?

### Solution

WAT IS UITEINDELIJK DE OPLOSSING GEWORDEN?

# Application game controller

All the components are individually tested and it needs to bring together.

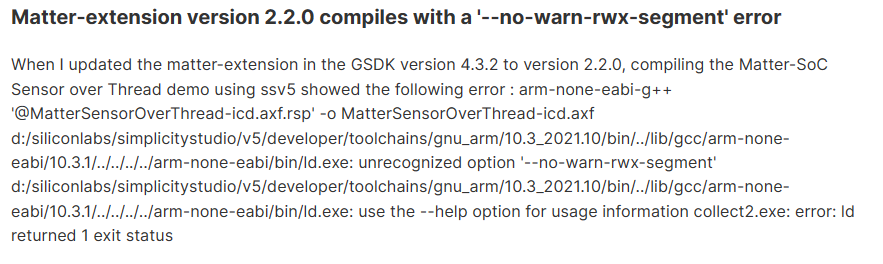
# Breakout game

# Notes

* **READ THE DETAILED DEVELOPMENT TOPICS** [Overview Guides - latest - Silicon Labs Matter Silicon Labs (silabs.com)](https://docs.silabs.com/matter/latest/matter-overview-guides/matter-provisioning)
* **Importand to read** [Developing with Silicon Labs Matter - latest - Silicon Labs Matter Silicon Labs (silabs.com)](https://docs.silabs.com/matter/latest/matter-start/)
* **How to use the On/Off server with level control to react to the gyroscope**
  + Use a callback function when the gyro reaches a sertain degrees
  + In the callback function include the level-control.h
  + Add app-common/zap-generated/attributes/Accessors.h  
    This is necessary for using the Set function to set the value to the attribute.
  + Add using namespace chip::app::Clusters::LevelControl
  + Add the following line to set the value Attributes::LevelControl::CurrentLevel::Set(endpoint, value) OR when not adding using namespace ….. add chip::app::Clusters::OnOff::Attributes::LevelControl::CurrentLevel::Set(endpoint, value)

**Problem with SSv5**

[Matter-extension version 2.2.0 compiles with a '--no-warn-rwx-segment' error (silabs.com)](https://community.silabs.com/s/question/0D5Vm000000kGczKAE/matterextension-version-220-compiles-with-a-nowarnrwxsegment-error?language=en_US)





# Document Information

## Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Description** |
| 22/09/2023 | BGY | 0.1 | Project overview and wireless controller description |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Related Documents

|  |  |
| --- | --- |
| **Ref** | **Description** |
| Matter-1.2-Core-Specification.pdf | Full specification document of the Matter protocol. |
|  |  |
| Matter-1.2-Application-Cluster-Specification.pdf | Supported application cluster specification. |
|  |  |
| Matter-1.2-Device-Library-Specification.pdf | All the device types are defined in this document. |
| DS-000143-ICM-20689-TYP-v1.1.pdf | The ICM-20689 motion tracking sensor datasheet. |