

Silicon Labs Matter

[Developing with Silicon Labs Matter](#)

[New Features](#)

[Quick-Start Guides](#)

[Overview and Setup](#)

[Matter Light and Switch Example](#)

[Wi-Fi](#)

[Thread](#)

[Next Steps](#)

[Fundamentals](#)

[Matter Fundamentals](#)

[Introduction to Matter](#)

[Matter Data Model](#)

[Matter Interactions Model](#)

[Matter Security](#)

[Matter Developer's Guide](#)

[Introduction](#)

[Matter Prerequisites](#)

[Hardware Requirements](#)

[Software Requirements](#)

[Artifacts](#)

[Matter Over Thread Example](#)

[Using the Matter Hub](#)

[Setting up the RCP](#)

[Creating an End Device](#)

[Using the Chip-Tool](#)

[Matter Over Wi-Fi Example](#)

[Overview](#)

[Getting Started](#)

[Software Installation](#)

[Get Started with SoC](#)

[Get Started with NCP](#)

[Set up Chip-Tool](#)

[Running the Matter Demo](#)

[Flash Firmware](#)

[Flash Bootloader](#)

[Build an SoC Application Using Studio](#)

[Build an NCP Application Using Studio](#)

[Flash a Binary](#)

- [Set Up the Raspberry Pi](#)
- [Run an Application](#)
- [Debug an Application](#)
- [Supported Features](#)
 - [Intermittently Connected Devices \(ICD\)](#)
 - [Direct Internet Connectivity](#)
 - [Interoperability with Ecosystems](#)
 - [Optimizing Memory Usage](#)
 - [Optimizing ICD Power Consumption](#)
 - [JLink RTT Support with SOC](#)
- [Direct Internet Connectivity](#)
 - [AWS Configuration Registration](#)
 - [OpenSSL Certificate Creation](#)
 - [Mosquitto Installation](#)
 - [MQTT Explorer Setup](#)
 - [Build DIC Application](#)
- [Matter Ecosystems](#)
 - [Single Controller Configuration](#)
 - [Google Ecosystem Setup](#)
 - [Apple Ecosystem Setup](#)
 - [Amazon Ecosystem Setup](#)
 - [Samsung Ecosystem Setup](#)
 - [Multi-Controller Configuration](#)
- [Matter Bridge](#)
 - [Matter Bridge Overview](#)
 - [Building The Matter Bridge](#)
 - [Running The Matter Bridge](#)
- [Detailed Development Topics](#)
 - [Overview Guides](#)
 - [Matter Provisioning](#)
 - [Test Matter Certificates for Development](#)
 - [Matter Commissioning](#)
 - [Matter Intermittently Connected Devices \(ICD\)](#)
 - [Matter OpenThread ICD Device](#)
 - [Matter Serial Port Communication \(Matter Shell\)](#)
 - [Matter SLC CLI Setup and Build Instructions](#)
 - [Matter Solutions](#)
 - [Matter OTA](#)
 - [Matter OTA Bootloader](#)
 - [Matter OTA Software Update](#)
 - [Matter 917 SOC OTA Software Update](#)
 - [Matter OTA WiFi Project](#)
 - [Matter Production Guide](#)
 - [Introduction](#)

[Device Development Prerequisites](#)
[Custom Part Manufacturing Services](#)
[Kudelski Security](#)
[Device Attestation](#)
[Matter API Reference](#)
[DataModel](#)
[Attributes](#)
[Clusters](#)
[Commands](#)
[Events](#)
[Cluster Implementation](#)
[Resources](#)
[Reference Guides](#)
[Matter Commit Hashes](#)
[How to Flash a Silicon Labs Device](#)
[How to Find Your Raspberry Pi](#)
[Using Development Tools in Simplicity Studio](#)
[Building a Custom Matter Device](#)
[Building a Multi-Endpoint Device](#)
[Using ZAP, the ZCL Advanced Platform](#)
[Using Wireshark with Matter](#)
[Matter EFR32 Flash Savings Guide](#)
[Matter FAQ](#)
[Thread FAQ](#)
[Wi-Fi FAQ](#)

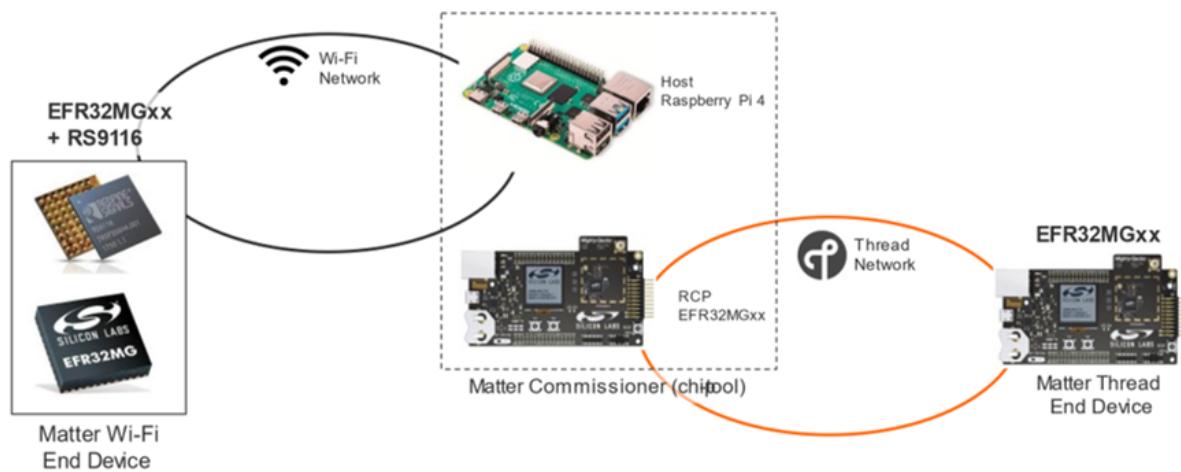
Developing with Silicon Labs Matter

Silicon Labs Matter

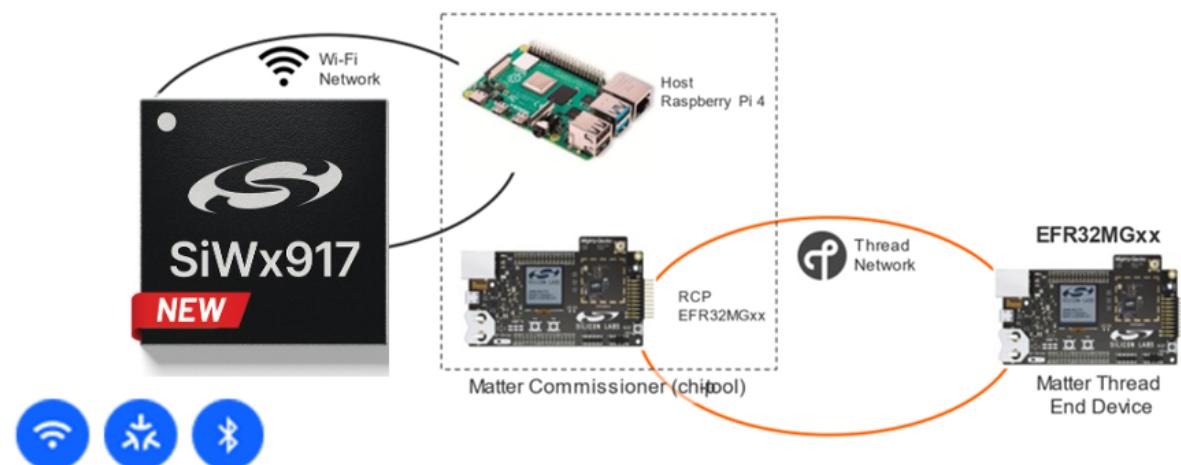
The Matter protocol leverages existing IP technologies, including Wi-Fi and Thread, to build a unified wireless connectivity ecosystem for smart homes. Internet Protocol (IP)-based networking provides manufacturers with simplified development while improving device compatibility for consumers.

Silicon Labs supports Matter on both 802.15.4 (Thread) and 802.11 (Wi-Fi) transport protocols. The Thread development use case differs from Wi-Fi because the Thread protocol requires an OpenThread Border Router (OTBR).

The Unify Matter Bridge is an application that makes legacy devices, such as Z-Wave and Zigbee devices, accessible on a Matter fabric. It does so by acting as an *IoT Service* in a Unify Framework.



Device: SiWx917 SoC



Two Paths for Development

These pages are for users who want to develop Matter applications in Simplicity Studio. The Simplicity Studio development path is the preferred path if you are looking for a GUI-based development experience in which you can create production-ready projects from a well-tested library. The Simplicity Studio development path also natively supports development on the Windows operating system. As a result, Windows users should use Simplicity Studio for their development environment.

Alternatively you can develop applications directly out of the [Silicon Labs Matter GitHub repo](#). Complete documentation for the GitHub development use case is provided in the [Silicon Labs Matter GitHub Documentation](#). This path is best for those who are experienced working with Matter and Silicon Labs products, prefer working with GitHub and a workflow that's not IDE-driven, or need access to newer features sooner at the cost of lesser test coverage.

Other Resources

To see release notes containing list of features and knowns issues, go to [Matter Release Notes on Silicon Labs Matter Extension](#).

If you are new to Matter or would like more information about Silicon Labs Matter-based products, see the [Matter content on silabs.com](#).

For background information on the Matter standard, see the [Connectivity Standard Alliance page](#).

To quickly make a simple Matter network, see the [quick-start guides](#).

To develop your own customized applications with Matter over Thread and Matter over Wi-Fi, see the [Matter Developer's Guide](#).

New Features

New Features

New Features for v2.2.0-1.2

- GA support for Intermittently Connected Devices
- Introduction of a third LCD screen to display application information
- Introduction of the Dishwasher Demo Application
- Adds support of Matter 1.2 on all devices
- Adds Matter support on SiWx917 SoC Common flash variants - BRD4338A
- Adds LCD display support on SiWx917 SoC for all the Wi-Fi Matter Apps
- Adds support for Direct Internet Connectivity on the SiWx917 SoC & NCP
- Support for Visual Studio Code integration
- Adds support for certificate provisioning on SiWx917 SOC
- Adds support for Firmware Upgrade on SiWx917 SOC

Self-Provisioning Mode

Silicon Labs' Matter examples now include a self-provision mode, which enables the application to be used as Generator Firmware with the provisioning script:

- e.g.: `python3 provision.py -c config/silabs.json -gf ../out/light/BRD4187C/matter-silabs-lighting-example.s37`

To enter the self-provisioning mode, factory reset the device pressing buttons BTN0 and BTN1 for six seconds. Using this method, the device application only needs to be flashed once, provisioned multiple times, and be ready for commissioning after each provisioning.

Support for Intermittently Connected Devices (ICD)

With the official introduction of Matter Short Idle Time (SIT) ICDs, both the door-lock sample app and the light-switch sample app are configured as SIT ICDs by default (with the exception of SiWx917 SoC examples). The default configurations can be found in their respective `sl_matter_icd_config.h` configuration files.

- Full support for ICD Short Idle Time (SIT) Devices in support of the Matter 1.2 specification
 - In this release, Silicon Labs has provided full support for Short Idle Time intermittently connected devices
 - These are ICDs (formerly called Sleepy End Devices) which must remain responsive to user input such as Door Locks and Window Coverings
- ICD Management cluster server implementation
 - Silicon Labs has provided an implementation of the ICD cluster server and the configuration of the ICD
- ICD Manager and ICD Event Manager has been implemented to manage the Idle and Active mode of the ICD
- NEW DNS advertisement Text Key SAI: indicates the SLEEPY_ACTIVE_INTERVAL (default to 4000 ms when ICD is not enabled)
- NEW Matter ICD configuration defines:
 - `CHIP_CONFIG_ICD_IDLE_MODE_INTERVAL` sets the value for the ICD IdleInterval attribute
 - `CHIP_CONFIG_ICD_ACTIVE_MODE_INTERVAL` sets the value for the ICD ActiveInterval attribute
 - `CHIP_CONFIG_ICD_ACTIVE_MODE_THRESHOLD` sets the value for the ICD ActiveThreshold attribute
 - `CHIP_CONFIG_ICD_CLIENTS_SUPPORTED_PER_FABRIC` sets the value for the ICD ClientsSupportedPerFabric attribute
 - All of these defines can be configured within `sl_matter_icd_config.h` inside the config directory (default values listed here):

```
#define SL_IDLE_MODE_INTERVAL = 600 // 10min Idle Mode Interval
#define SL_ACTIVE_MODE_INTERVAL = 1000 // 1s Active Mode Interval
#define SL_ACTIVE_MODE_THRESHOLD = 500 // 500ms Active Mode Threshold
#define SL_ICD_SUPPORTED_CLIENTS_PER_FABRIC = 2 // 2 registration slots per fabric

// The OpenThread polling rates used in either ICD mode
#define SL_OT_IDLE_INTERVAL = 15000 // 15s Idle Intervals
#define SL_OT_ACTIVE_INTERVAL = 200 // 200ms Active Intervals
```

- CHANGES:

- Optimized the subscription reports by synchronizing all client's subscriptions with the ICD idle mode interval. This ensures the minimal amount of wake ups possible due to subscription reports. This component is introduced as `matter_subscription_synchronization`.
- The previous `matter_sed` components has been replaced by `matter_icd`. This goes in line with previous sleepy end device behavior being deprecated and replaced by the ICD behavior.
- Silicon Labs' Light Switch and Door Lock apps support the ICD implementation and have the ICD cluster enabled.

Overview and Setup

Quick-Start Guides for Matter over Thread and Matter over Wi-Fi

Overview

The procedures here describe how to make a simple network of a light, a switch, and a Matter hub, and to use the switch to control the light. First, set up your hardware and software as described below. Then you will follow a step by step procedure to:

- Create a Matter hub on a Raspberry Pi.
- Compile and load a light and a switch example application onto two Silicon Labs development boards to make light and switch Matter Accessory Devices (MADs).
- Create a Matter network with the MADs and the Matter hub.
- Test the light through the Matter hub.
- Bind the switch MAD to the light MAD, so that the switch can control the light.

Initial Setup

Both the Matter over Wi-Fi and Matter over Thread demos require that you have set up a simple development environment with Simplicity Studio, two EFR32MG24-based development boards, and a Raspberry Pi used as a Matter hub. The following requirements are common to both demos. The Thread demo also requires a radio co-processor (RCP) as part of the Matter Hub. The requirements for this are provided in the [introduction to the Thread demo](#).

Hardware Requirements

Matter Hub

- 1 Raspberry Pi 4B
- 1x high speed, 64 GB SD card

Matter Devices

Matter Over Wi-Fi Accessory Device Requirements for NCP Mode

The Silicon Labs Matter over Wi-Fi NCP mode demo and development requires two boards: the Silicon Labs EFR32 Radio board to run the Matter code and either the RS9116, SiWx917, or WF200 to run the Wi-Fi protocol stack.

The following boards are supported for the Matter over Wi-Fi demos and development:

- MG24 Boards:
 - BRD4186C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@10dBm
 - [XG24-RB4186C](#)
 - MG24 with WSTK: [xG24-PK6009A](#)
 - BRD4187C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@20dBm
 - [XG24-RB4187C](#)
 - MG24 with WSTK: [xG24-PK6010A](#)
- Wi-Fi NCP Dev Kits & boards
 - RS9116
 - SB-EVK1 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK1](#)
 - SB-EVK2 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK2](#)

Overview and Setup

DB-EVK1 / Dual Band Wi-Fi Development Kit / 2.4GHz & 5GHz

- [RS9116X-DB-EVK1](#)

- SiWx917 NCP

- SiWx917 NCP Mode / Wi-Fi Expansion Board / 2.4GHz
 - BRD8045A (B0 Expansion v2.0)

- WF200

- WF200 / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8022A](#)
 - WF200 / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8023A](#)

- Interconnect board (included in the Wi-Fi kits)

- SPI Cable (included in the RS9116 kit)

- Jumper Cables (included in the RS9116 kit)

Note: For more information, refer to [Hardware Requirements](#).

Matter Over Wi-Fi Accessory Device Requirements for SoC Mode

The Silicon Labs Matter over Wi-Fi demo and development for SoC mode requires the SiWx917 SoC board that supports Matter over Wi-Fi in a single-chip package. The integrated MCU is dedicated for peripheral and application-related processing (Matter), while the ThreadArch® runs the wireless and networking protocol stacks.

Pre-built images for the SiWx917 connectivity firmware are available as per the instructions on the [Matter Artifacts page](#). The following boards are supported for the Matter over Wi-Fi demos and development:

- **Wi-Fi SoC Boards:**

- SiWx917 / BRD4002A / Wireless Starter Kit
- SiWx917 Soc Mode
 - SiWx917 SoC / Common Flash Radio Board / 2.4GHz
 - BRD4338A - B0 common flash v2.0

Software Requirements

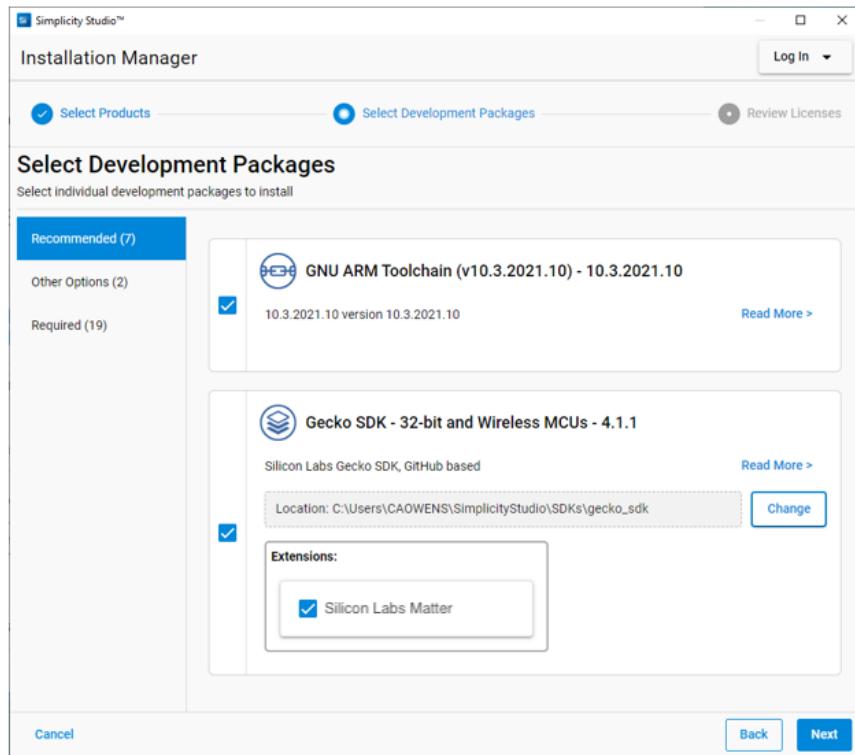
Simplicity Studio 5: Download and install Simplicity Studio 5 for your operating system from the Silicon Labs [Simplicity Studio page](#). While the installation process is easy to follow, instructions are provided in the Simplicity Studio v5 [Getting Started section](#).

Ozone - The J-Link Debugger: [Ozone](#) is a full-featured graphical debugger for embedded applications. With Ozone, it is possible to debug any embedded application on C/C++ source and assembly level.

Simplicity Commander: [Simplicity Commander](#) is a utility that provides GUI and command line access to the debug features of an EFM32 device. It allows you to flash firmware, update the kit firmware, and lock or unlock debug access.

Tera Term: [Tera Term](#) is the terminal emulator for Microsoft Windows that supports serial port, telnet, and SSH connections.

Silicon Labs Matter GSDK Extension: Once Simplicity Studio 5 is installed, you will be prompted to install the Gecko SDK Suite (GSDK). Here you should also install the Matter Enablement Package by making sure the extension is checked, as shown.



Installation of Wi-Fi SDK and Wiseconnect Packages The following packages will be installed during the installation of Simplicity Studio. Refer to [Package Installation](#)

Matter Hub Raspberry Pi Image: A copy of the pre-built image from the Silicon Labs web services can be downloaded in this [zipfile](#). Note this is a large file and will take some time to download.

Note: The Matter hub for Matter over Thread requires an additional device, a radio co-processor. See [the introduction to the Matter over Thread demo](#) for more information.

Matter Bootloader Image: The EFR32MG24 devices must be programmed with a bootloader. Obtain those here: [Silicon Labs Matter Artifacts](#).

SSH Client: Managing the Matter hub often requires connecting to it remotely. An SSH client is needed to follow the step-by-step example in this document (PuTTY is used). Install software such as [PuTTY](#), Terminal, or a similar application for access to the Raspberry Pi-based Matter hub.

SD Card-Flashing Software: Many different applications can be used to prepare an SD card for the Raspberry Pi, such as the [Raspberry Pi Imager](#), [balenaEtcher](#), and [Rufus](#). The step-by-step example in this document uses the Raspberry Pi Imager.

Visual Studio Code Development

In addition to creating and building your Matter project within Simplicity Studio, Silicon Labs also provides Visual Studio Code (VSCode) IDE integration. Matter projects support VSCode integration **with the exception of Matter Solutions projects**.

For more information on development within Visual Studio Code, please visit [Visual Studio Code Enablement](#).

Next Steps

Now that you have your environment, you can create a [Matter over Wi-Fi network](#) or a [Matter over Thread network](#).

Matter Light and Switch Example

Matter Light and Switch Example

These pages describe how to create a simple Matter network in which you can use a switch to control a light.

You should have obtained hardware and installed software as described in the [Overview](#). Once you have done so, you can begin the step-by-step instructions for Matter over Wi-Fi or Matter over Thread.

- [Wi-Fi](#)
- [Thread](#)
- [Next Steps](#)

Wi-Fi

Light and Switch Step-by-Step Example

Setting up the Matter Hub/Chip-Tool

This procedure prepares the Raspberry Pi 4B (RPi4B) to become a Matter Hub. You should have downloaded the Matter Hub Raspberry Pi image and Raspberry Pi Imager as described in the [Overview](#). The Raspberry Pi image contains software called chip-tool, which provides a command-line interface into the Matter protocol.

1. Install the Raspberry Pi Imager and insert the SD card into the PC to flash the image.
 1. Open the Imager, select the Operating System as 'Custom OS' and browse for the Raspberry Pi Image.
 2. Select the storage as an SD card.
 3. Click the settings icon to configure the access point (AP) credentials, Hostname and user credentials. Make sure the 5 GHz Wi-Fi credentials of the dual-band AP are entered.
 4. Click the 'write' option. **Note** this will erase all existing content on that SD card.
2. Insert the SD card into Raspberry Pi 4B (RPi4B).
3. Power-up the RPi4B. Once it is booted up, check the Raspberry Pi's IP address. Refer to [Finding Raspberry Pi IP address](#) in the References chapter to get the IP address or enter the Hostname directly in PuTTY.
4. Use PuTTY to connect to RPi4B.
 1. The first time connecting to RPi4B, PuTTY will warn about a new host key or key fingerprint. Accept the key.
 2. The credentials (username: password) are the same given Step 1.
 3. Switch to root mode and navigate to path "/home/ubuntu/connectedhomeip/out/standalone" to find the chip-tool.

Matter hub/chip-tool are ready and working. Keep the PuTTY session open for the following steps.

Creating the Matter Accessory Devices (MADs)

Hardware Requirements

- SiWx917 / BRD4002A / Wireless Starter Kit
 - SiWx917 SoC Mode
 - SiWx917 SoC / Common Flash Radio Board / 2.4GHz
 - BRD4338A - B0 common flash v2.0
- Note:** Refer to [SiWx917 SoC](#) for more details.

Software Requirements

In order to run the Light and Switch Example on SiWx917 SOC, software must be installed. Refer to [Software Requirements](#).

Note: Switch application is not supported for NCP devices

1. In Simplicity Studio 5, create the Light MAD:
 1. Switch to the Launcher view (if not already in it).
 2. Connect one compatible dev board to the development computer.
 3. Once it shows up in the Debug Adapters view, select it.
 4. Open the Example Projects and Demos tab, select the **Matter** filter and enter "Wi-Fi" in **Filter on keywords**.
 5. Select the *Matter - Lighting over Wi-Fi* example, click **Create**, rename the project if you wish, and click **Finish**.
 6. Once the project is created, the perspective changes to the Simplicity IDE perspective. In the Project Explorer view, right-click the project and select *Build Project*.
 7. Once the project has compiled, in the Debug Adapters view right-click the board and select *Upload application*.
 8. Select the *Application image path* (Select the path for `.rps` or `_isp.bin` file for soc in the path '`<workspace>\project_name\GNU ARM v12.x.x - Default`').
 9. Disconnect the dev board from development computer.

10. **Optional:** Label this device (eg: my_light or my_switch) to make it easier to identify later.
2. Repeat the process with the second Dev board, but select the *Matter - Light Switch over Wi-Fi* example instead.

Creating the Matter Network

This procedure uses the chip-tool installed on the Matter Hub. Chip-tool includes many commands. The following are some that are used in this example:

- chip-tool ble-wifi pairing
- chip-tool onoff
- chip-tool toggle
- chip-tool accesscontrol
- chip-tool binding

In a PuTTY session to the Matter hub, use the chip-tool to commission the Matter light device. The various compatible boards will have different setups for their LED(s). Typically one LED (or a color channel, if RGB) will be used to indicate the network status of the device:

- A short flash indicates the MAD is advertising to join a network
- A rapid flash indicates the commissioning is in progress
- Solid ON: the MAD is now in the network

1. Power up the Matter Light device.
2. Once it is powered up and booted, use the PuTTY session to commission the device using
`./chip-tool pairing ble-wifi nodeID SSID PSK 20202021 3840`
 where `nodeID` is replaced with the desired ID (for example `./chip-tool pairing ble-wifi 1122 Silabs PSK 20202021 3840`).
3. Make sure the SSID and PSK given here are of 2.4 GHz of the Dual Band AP.
 Be sure to note which nodeIDs are used for Matter light and Matter light_switch devices. These will be needed later for modifying the Matter light's ACL & the Matter light switch's binding table.
4. Power up the Matter light switch device and commission it too, using a different `nodeID`.

Now two Matter accessory devices (MADs) are on the network and ready to be used.

Controlling the Light MAD

1. In a PuTTY session to the Matter hub, use the chip-tool to test the Matter light device.
 1. Control the light status of the light MAD Using `./chip-tool onoff` on `nodeID 1`. You can also use `chip-tool onoff off` and `chip-tool toggle`.
 2. For dev board with buttons available, you can use BTN1 to toggle the light status locally.
2. In a PuTTY session to the Matter hub, use the chip-tool to bind the light_switch MAD to the light MAD, thus allowing the switch to control the light.
 1. First, modify the Access Control List (ACL) of the Matter light device. This list determines which device in the network the Matter light device will react to. Use: `./chip-tool accesscontrol write acl '[{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [{"nodeID": 112233}], "targets": null}, {"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": [{"nodeID": "nodeID-switch"}], "targets": null}]'` **nodeID-light 0**, where the highlighted parameters are:
 - **112233**: The node ID of the controller. This is always 112233.
 - **nodeID-switch**: The node ID of the Matter light switch device.
 - **nodeID-light**: The node ID of the Matter light device.
 - **0**: The endpoint in the Matter light device that holds the ACL. This is always 0.
 To read the ACL for a Matter device use: `./chip-tool accesscontrol read acl nodeID 0`, where the highlighted parameters are:
 - **nodeID**: The nodeID of the Matter device (Light or Light_switch) to read the ACL contents from.
 - **0**: The endpoint in the Matter device that holds the ACL. This is always 0.

For Example:- `./chip-tool accesscontrol write acl '[{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [{"nodeID": 112233}], "targets": null}, {"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": [{"nodeID": "nodeID-switch"}]}]'` 1111 0

1. Second, bind the switch's write command to the light. This is done by updating the binding table of the Matter light_switch device. This can be done using the command: `./chip-tool binding write binding '[{"fabricIndex": 1, "nodeID": nodeID-light, "endpoint": 1, "cluster": 6}]'` **nodeID-switch 1**, where the highlighted parameters are:
 - **nodeID-light**: The node ID of the Matter light device.
 - **1**: The application endpoint in the light. This is always 1.

- **6**: The on/off cluster in the light. This is always 6.
- **nodeID-switch**: The node ID of the switch.
- **1**: This is the application endpoint in the switch that holds the binding table. This is always 1.

The binding table from a Matter device can be read using: `./chip-tool binding read binding nodeID-switch 1`, where the highlighted parameters are:

- **nodeID-switch**: The node ID of the Matter switch.
- **1**: The application endpoint in the switch that holds the binding table. This is always 1.

For Example:- `./chip-tool binding write binding '[{"fabricIndex": 1, "node": 1, "endpoint": 1, "cluster": 6}]' 2222 1`

3. With the binding complete, a button press (BTN1) on Matter light_switch device should now toggle the light status of Matter light device.

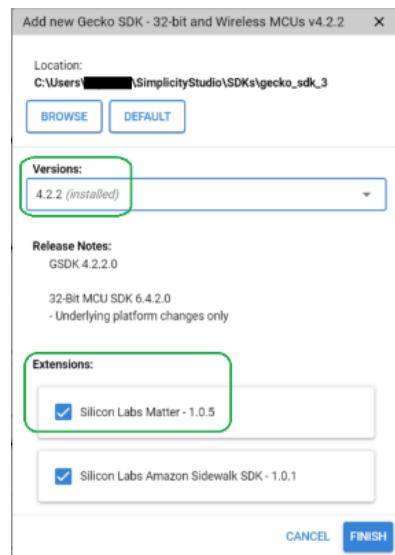
Thread

Matter over Thread Light and Switch Step-by-Step Example

Setting up the Matter hub

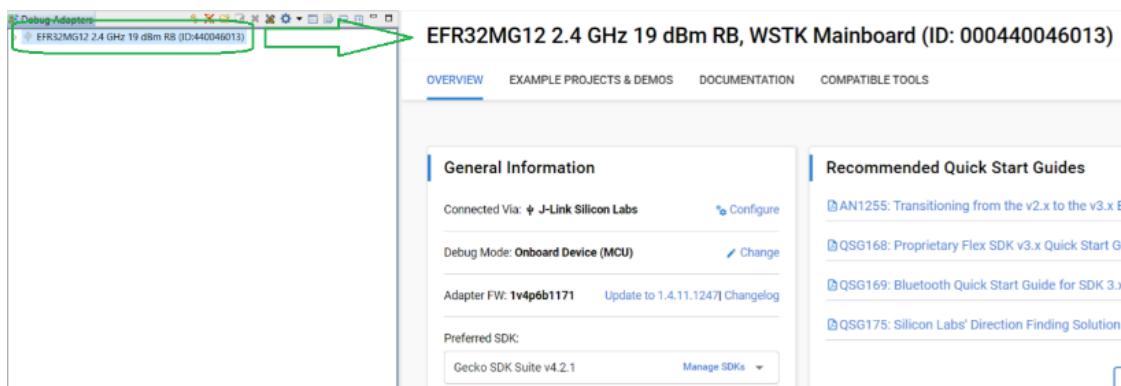
1. Prepare a compatible dev board to become your Matter hub's ot-rcp (see details in the [introduction](#)):

1. Start Simplicity Studio 5 with the latest GSDK and Silicon Labs Matter GSDK Extension installed (see details in the [Overview](#)).

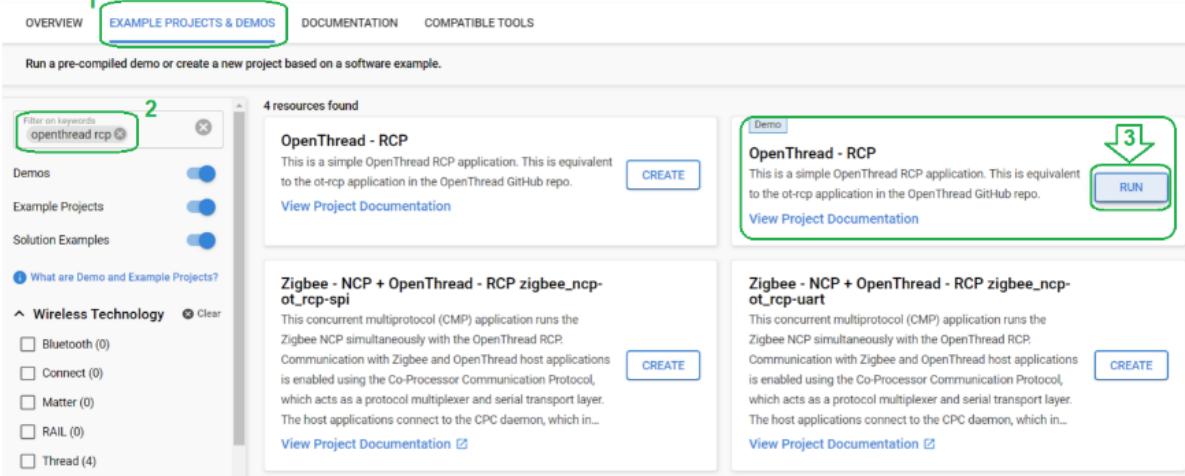


2. Connect the dev board to your development computer.

3. Once it shows up in the Debug Adapters view, select it.



4. Go to the Example Projects and Demos tab. Select the *OpenThread* filter and enter "openthread rcp" in the **Filter on keywords** box. Select the *OpenThread - RCP* example and click **Run**.

EFR32MG12 2.4 GHz 19 dBm RB, WSTK Mainboard (ID: 000440046013)


The screenshot shows the Thread developer portal interface. At the top, there are tabs: OVERVIEW (highlighted with a green box), EXAMPLE PROJECTS & DEMOS (also highlighted with a green box), DOCUMENTATION, and COMPATIBLE TOOLS. Below the tabs, a message says "Run a pre-compiled demo or create a new project based on a software example." A search bar contains the text "Filter on keywords openthread rcp". On the left, there are filters for Demos (selected), Example Projects (selected), and Solution Examples (selected). Under "What are Demo and Example Projects?", there is a section for "Wireless Technology" with options for Bluetooth, Connect, Matter, RAIL, and Thread, none of which are selected. The main area displays four resources found:

- OpenThread - RCP**: This is a simple OpenThread RCP application. This is equivalent to the ot-rcp application in the OpenThread GitHub repo. [View Project Documentation](#) [CREATE](#)
- OpenThread - RCP**: This is a simple OpenThread RCP application. This is equivalent to the ot-rcp application in the OpenThread GitHub repo. [View Project Documentation](#) [RUN](#) (highlighted with a green box and a green arrow labeled 3)
- Zigbee - NCP + OpenThread - RCP zigbee_ncp-ot_rcp-spi**: This concurrent multiprotocol (CMP) application runs the Zigbee NCP simultaneously with the OpenThread RCP. Communication with Zigbee and OpenThread host applications is enabled using the Co-Processor Communication Protocol, which acts as a protocol multiplexer and serial transport layer. The host applications connect to the CPC daemon, which in... [View Project Documentation](#) [CREATE](#)
- Zigbee - NCP + OpenThread - RCP zigbee_ncp-ot_rcp-uart**: This concurrent multiprotocol (CMP) application runs the Zigbee NCP simultaneously with the OpenThread RCP. Communication with Zigbee and OpenThread host applications is enabled using the Co-Processor Communication Protocol, which acts as a protocol multiplexer and serial transport layer. The host applications connect to the CPC daemon, which in... [View Project Documentation](#) [CREATE](#)

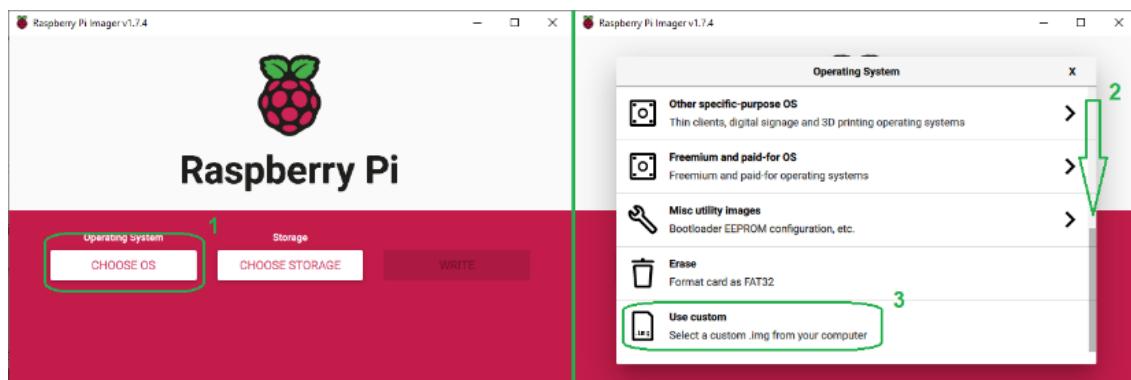
5. Disconnect the dev board and connect it to your RPi4B.

2. Prepare the Raspberry Pi 4B (RPi4B) to become a Matter hub:

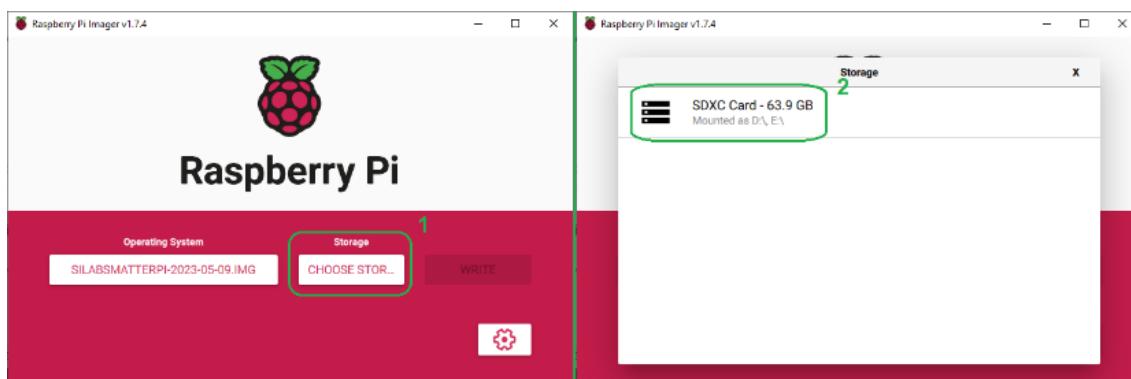
1. Download and extract the Matter hub Raspberry Pi image (see details under [Software requirements](#)).

2. Flash the image to the desired SD card. Please note this will erase all existing content on that SD card:

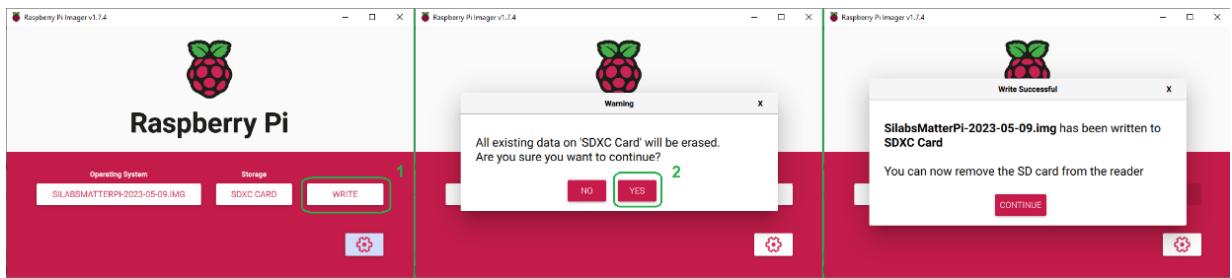
1. Under **Operating System**, click **CHOOSE OS**. Scroll down and choose the last option, **Use custom**. Browse to your extracted **SilabsMatterPi.img** file and select it.



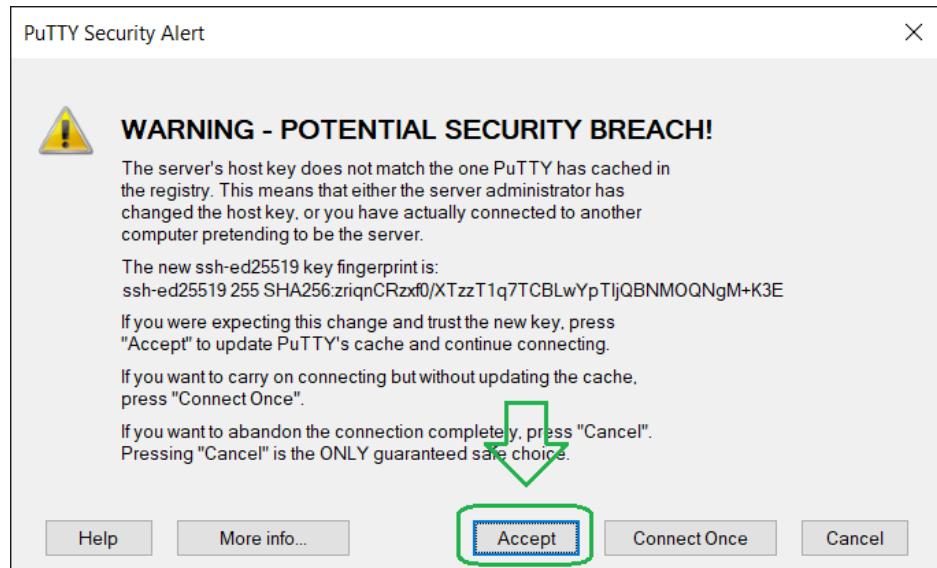
2. Under **Storage**, click **CHOOSE STORAGE**. Select your target SD card. If it does not show up, make sure it is inserted properly and not in use by other software and retry this step.



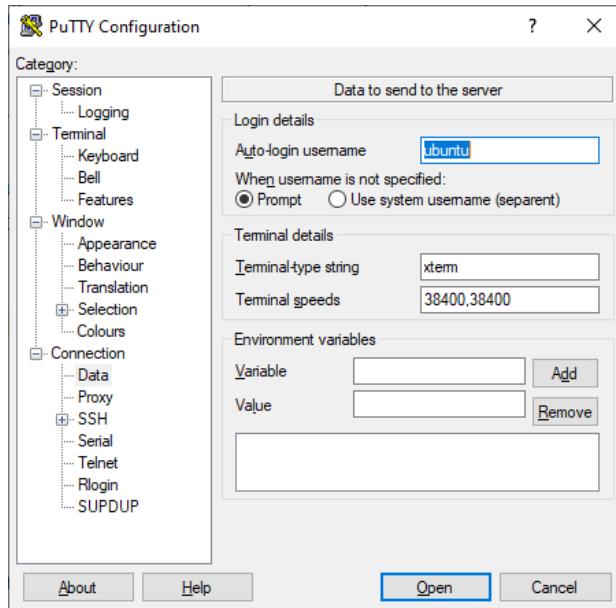
3. Click **WRITE** and confirm the action by clicking **YES**. Wait for the process to finish writing to the SD card and verifying the results.



3. Insert the SD card in your Raspberry Pi 4B (RPi4B) and connect it to your network by Ethernet (Wi-Fi cannot be used for this example).
 4. Power-up the RPi4B. Once it is booted up, check your local network DHCP IP address allocation rules to determine the RPi4B's assigned IP address. You may also use networking tools such as *nmap* to find your RPi4B's IP assigned address.
 5. Use PuTTY to connect to your RPi4B.
1. The first time you connect to your RPi4B, PuTTY will warn you about a new host key or key fingerprint. Accept the key.



2. The default credentials are **ubuntu:ubuntu** (username:password). **Note:** In PuTTY, you can set the default username to **ubuntu** under *Connection > Data > Auto-login username*.



3. You may be prompted to change the password.
6. Verify that your *ot-rcp* board is visible in your RPi4B. It should show up as a *ttyUSBn* or a *ttyACMn* in */dev/*, where *n* indicates the port's number (ex: */dev/ttyACM0*).

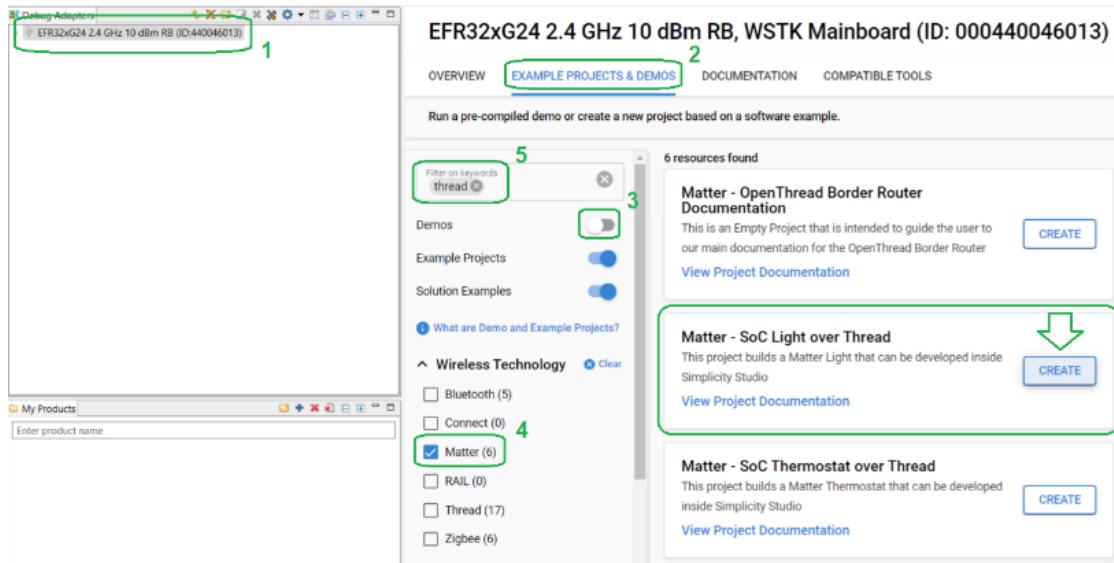
```
ubuntu@ubuntu:~$ ls /dev/ttyA*
/dev/ttyACM0 /dev/ttyAMA0
ubuntu@ubuntu:~$
```

Note: If your RCP shows up with a number other than 0, the *otbr-agent* file will need to be updated.

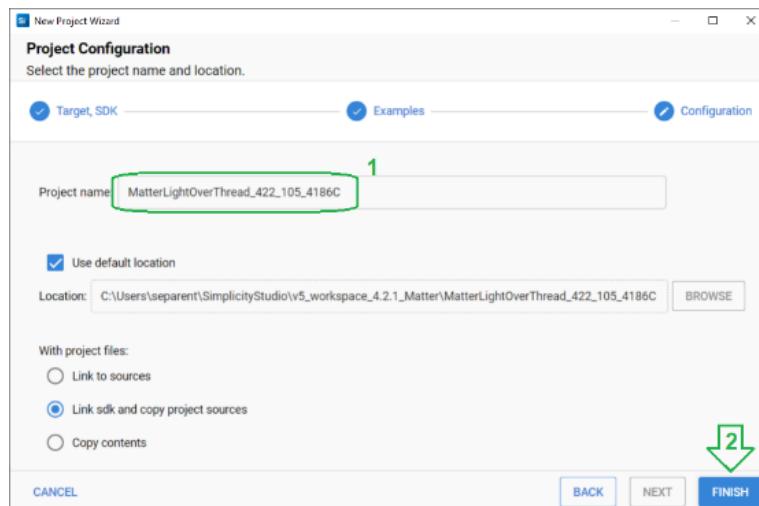
You now have a working Matter hub. Keep the PuTTY session open for the following steps.

Creating the Matter Accessory Devices (MADs)

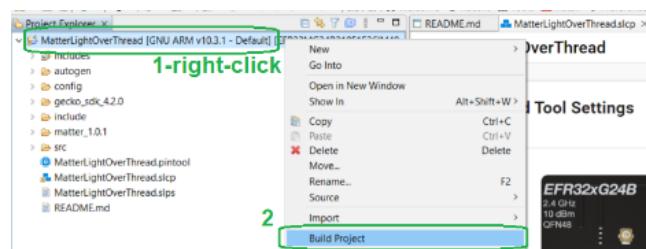
1. In Simplicity Studio 5, create the light MAD:
 1. Switch to the Launcher view (if not already in it).
 2. Connect one compatible dev board to your development computer. This example uses a BRD4186C.
 3. Once it shows up in the Debug Adapters view, select it.
 4. Go to the Example Projects and Demos tab, turn off *Demos*, check the *Matter* filter under *Wireless Technology* and enter "*thread*" in the *Filter on keywords* box.



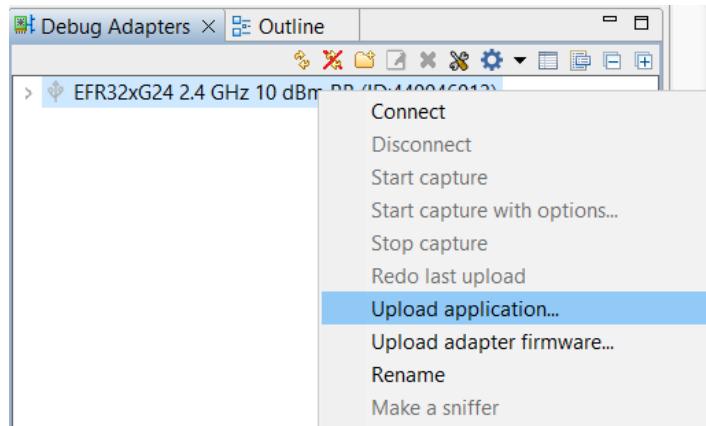
5. Select the *Matter - SoC Light over Thread* example and click **Create**. Name your project and click **Finish** (no other changes are required at this time).



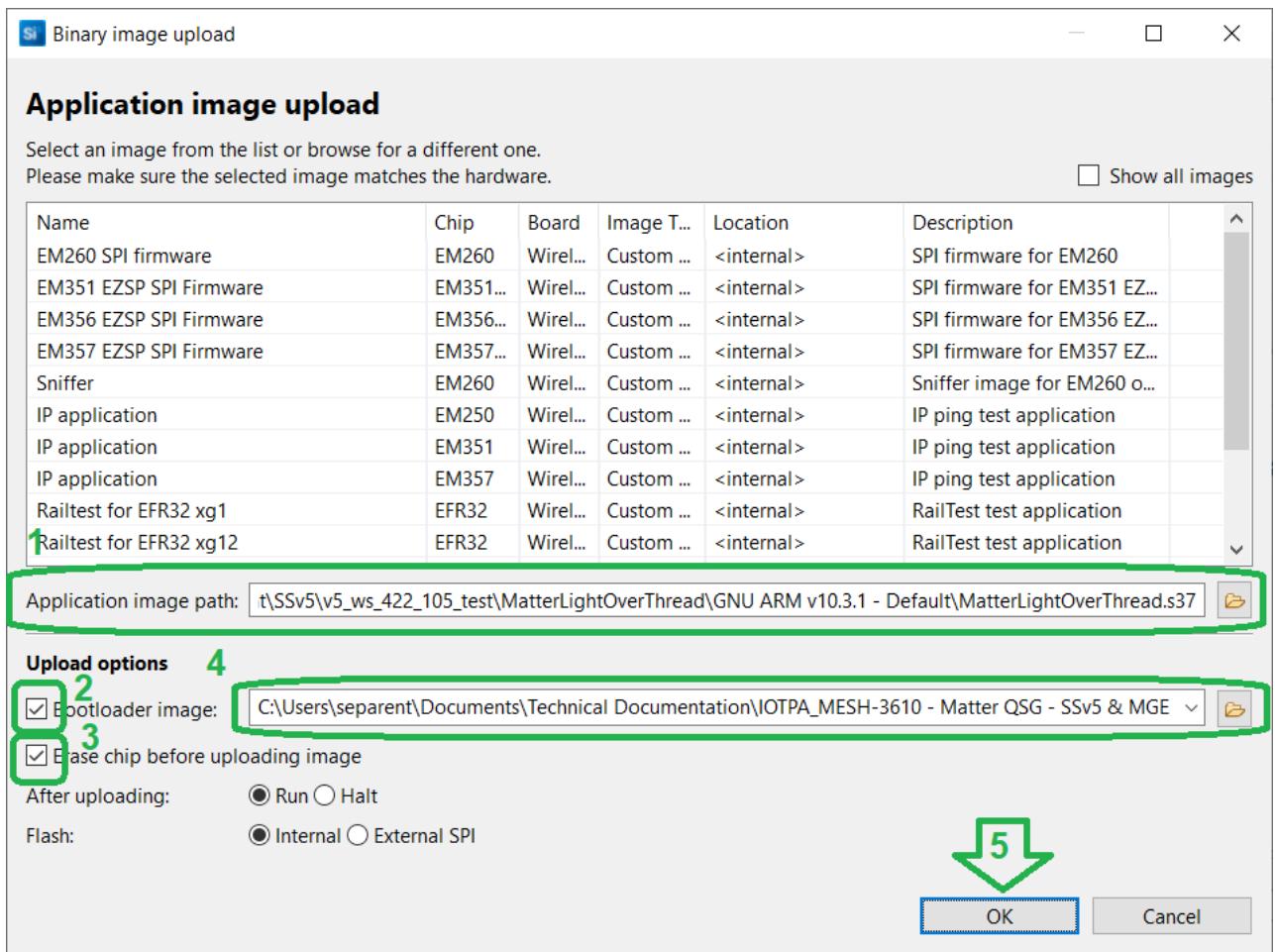
6. Once the project is created, right-click it in the Project Explorer view and select *Build Project*.



7. Once the compilation is done, right-click the dev board in the Debug Adapters view and select *Upload application...*

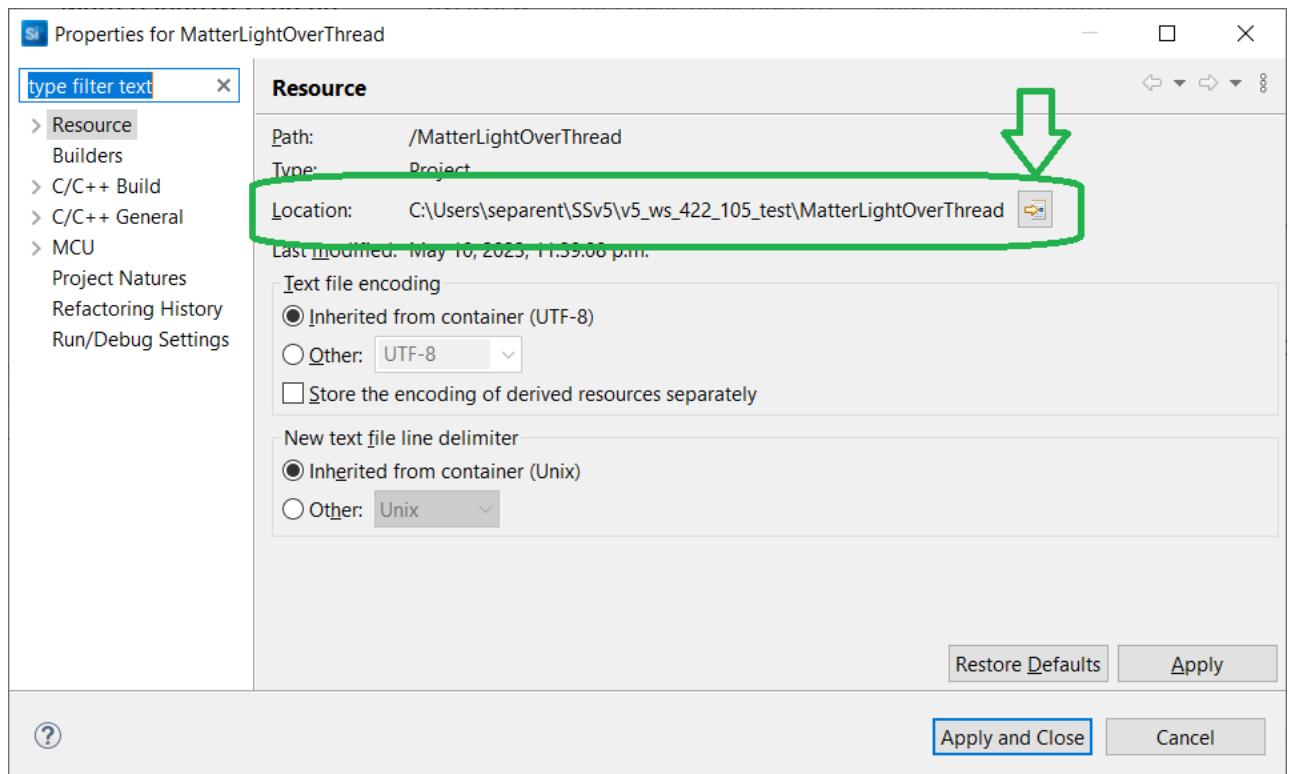


8. Select the *Application image path* for your newly compiled project and a *Bootloader image*. Bootloader images are provided in the zip file referenced on the [Silicon Labs Matter Artifacts page](#). Unzip and reference the extracted location. Check "Erase chip before uploading image".



Note: If you are unsure of the path for the newly created binary, you can find the project's path in the project's *Properties* window under *Resource*. The binary is typically located inside the *{workspace folder}/{project name folder}/{{GNU ARM v?? folder}}/{project name binary}.s37*. For example, you would find the binary for a Matter light over Thread project with the default name here: *{workspace folder}/MatterLightOverThread/GNU ARM v10.3.1 - Default/MatterLightOverThread.s37*

Note: You should only need to upload a bootloader image and erase the chip once. Subsequent application uploads do not need the bootloader image or chip erasure.



9. If you are using a dev board with a WSTK, you should now see a QR code displayed on the WSTK's LCD.
10. Disconnect the dev board from your development computer.
11. **Optional:** you may want to label this device as your light or switch, as appropriate, to make it easier to identify later.
2. Repeat the process above with the second dev board but selecting the *Matter - SoC Light Switch over Thread* example instead.

Creating the Matter Network

The *mattertool* is a convenient wrapper script that allows you to easily perform common steps using *chip-tool*. Chip-tool provides a command-line interface into the Matter protocol. mattertool and chip-tool are installed in the Matter Hub image file.

mattertool provides many commands at your disposal. These are some important ones used in this example:

- *mattertool startThread*
- *mattertool bleThread*
- *mattertool on*
- *mattertool off*
- *mattertool toggle*

You can use the command *mattertool help* to display the available commands. With some commands (such as on, off, and toggle), you may also want to specify which node ID to interact with. You can do this by adding the *-n nodeId* parameter after a command, such as *mattertool on -n 5678*.

The *mattertool* also allows you to perform any of the commands you can normally use directly with the *chip-tool*.

1. In a PuTTY session to the Matter hub, use the *mattertool* to create your network.
 1. Start the Thread network with: *mattertool startThread*. This will output the Thread dataset.

```
ubuntu@ubuntu:~$ mattertool startThread
WELCOME TO...

ACTIVATOR! This sets your shell environment variables.

Activating environment (setting environment variables):

Setting environment variables for CIPD package manager...done
Setting environment variables for Python environment.....done
Setting environment variables for pw packages.....skipped
Setting environment variables for Host tools.....done

Checking the environment:

20230511 05:52:56 WRN Current uname (5.15.0-1012-raspi #14-Ubuntu SMP PREEMPT Fri Jun 24 13:10:28 UTC 2022 aarch64) does not match Bootstrap uname (5.15.0-52-generic #58-Ubuntu SMP Thu Oct 13 08:03:55 UTC 2022 aarch64), you may need to rerun bootstrap on this system
20230511 05:52:56 INF Environment passes all checks!

Environment looks good, you are ready to go!

Starting a new thread network
Done
Done
Done
Done
Done
Done
Done
dead00beef00cafe
Done
New ThreadDataset: 0e080000000000000000000000000000b35060004001ffffe00208dead00beef00cafe0708fddead00beef000005105b4e7458
86cdf87de6e27516d42723a6030a4f70656e54687265616401025a54041042dcba5df17d3d9ff41fba7ceaad3b840c0402a0f7f8
ubuntu@ubuntu:~$
```

2. If you need the Thread dataset in the future (such as after a power cycle), you can use `mattertool getThreadDataset`.

Connecting MADs to the Matter hub

1. In a PuTTY session to the Matter hub, use the `mattertool` to commission your Matter light switch device.

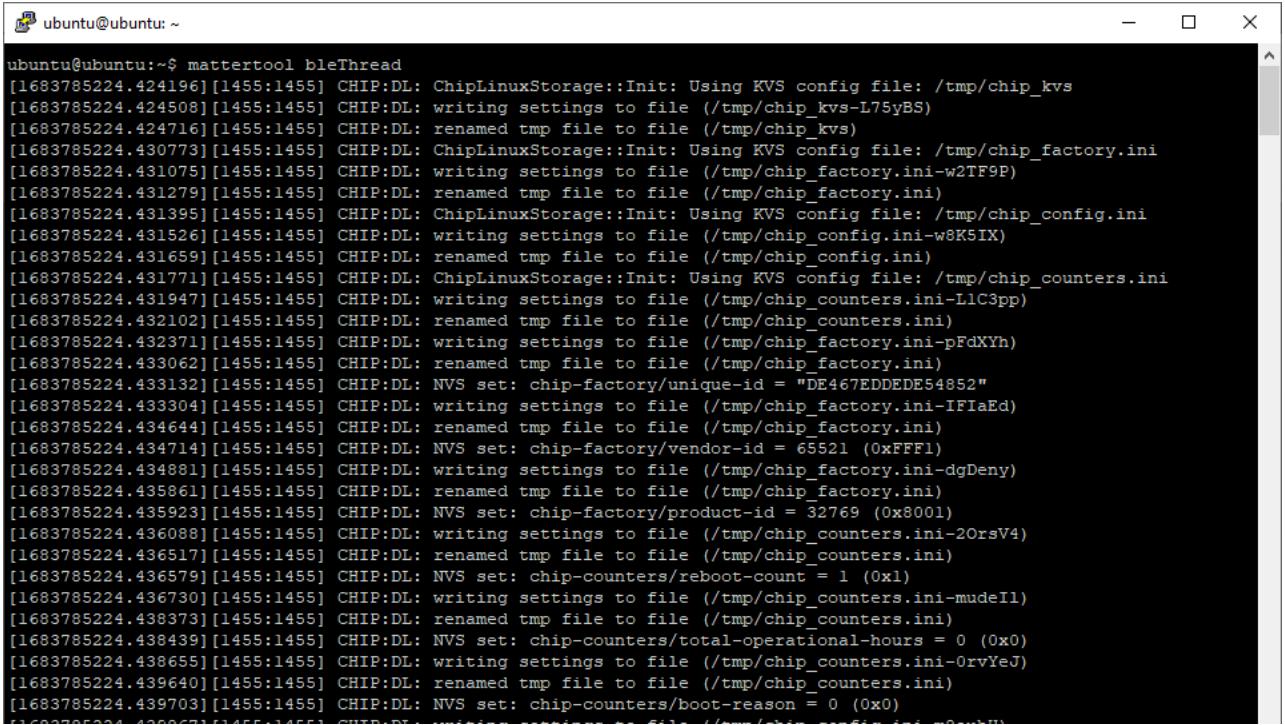
1. Connect to a power source your Matter light switch device.

Notes: Only power-up one device at a time to ensure proper commissioning since no discriminator is used to keep this example simple.

If this device previously failed commissioning or was on a Thread network, you may need to perform a factory reset before doing this process.

The Matter devices will stop advertising after ~15 minutes. Make sure to complete the commissioning before the advertising has expired. If unsure, power cycle the MAD.

2. Once the MAD is powered and booted up, use the PuTTY session to commission the device using 'mattertool bleThread -n nodeID', where 'nodeID' is replaced with the desired ID. In the resulting log, you should see a line similar to: [1683785224.525598][1455:1461] CHIP:CTL: Commission called for node ID 0x000000000000637E , indicating the nodeID (in this example, 0x637E) of the newly commissioned device.



```
ubuntu@ubuntu:~$ mattertool bleThread
[1683785224.424196][1455:1455] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1683785224.424508][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_kvs-L75yBS)
[1683785224.424716][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_kvs)
[1683785224.430773][1455:1455] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1683785224.431075][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_factory.ini-w2TF9P)
[1683785224.431279][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_factory.ini)
[1683785224.431395][1455:1455] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1683785224.431526][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_config.ini-w8K5IX)
[1683785224.431659][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_config.ini)
[1683785224.431771][1455:1455] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1683785224.431947][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-LlC3pp)
[1683785224.432102][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)
[1683785224.432371][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_factory.ini-pFdXYh)
[1683785224.433062][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_factory.ini)
[1683785224.433132][1455:1455] CHIP:DL: NVS set: chip-factory/unique-id = "DE467EDDE54852"
[1683785224.433304][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_factory.ini-IFIaEd)
[1683785224.434644][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_factory.ini)
[1683785224.434714][1455:1455] CHIP:DL: NVS set: chip-factory/vendor-id = 65521 (0xFFFF)
[1683785224.434881][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_factory.ini-dgDeny)
[1683785224.435861][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_factory.ini)
[1683785224.435923][1455:1455] CHIP:DL: NVS set: chip-factory/product-id = 32769 (0x8001)
[1683785224.436088][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-2OrsV4)
[1683785224.436517][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)
[1683785224.436579][1455:1455] CHIP:DL: NVS set: chip-counters/reboot-count = 1 (0x1)
[1683785224.436730][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-mudeII)
[1683785224.438373][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)
[1683785224.438439][1455:1455] CHIP:DL: NVS set: chip-counters/total-operational-hours = 0 (0x0)
[1683785224.438655][1455:1455] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-0rvYeJ)
[1683785224.439640][1455:1455] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)
[1683785224.439703][1455:1455] CHIP:DL: NVS set: chip-counters/boot-reason = 0 (0x0)
```

Note: If you do not specify a *nodeID*, one will be assigned automatically. Make sure to take note of what nodeID assigned to your Matter light switch & Matter light devices. These will be needed later for modifying the Matter light's ACL & the Matter light switch's binding table.

3. Power up the Matter light device and commission it as well, by following the previous steps for this MAD but using a different *nodeID*.

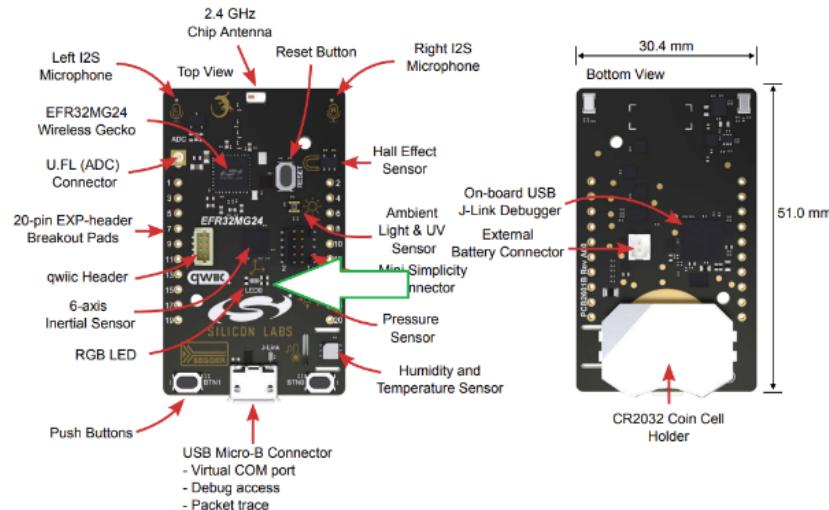
You now have two Matter devices on your network ready to be used

Controlling the Light MAD

The various compatible boards will have different setups for their LED(s). Typically, one LED (or a color channel, if RGB) is used to indicate the network status of the device:

- Short blink: indicates the MAD is advertising to join a network.
- Half/half blink: indicates the BLE steps are in progress.
- Long blink: indicates joining the Thread network.
- Solid on: indicates the MAD is now in the network.

For example, on the dev board *xG24-DK2601B EFR32xG24* (also known as *BRD2601*), the red channel of *LEDO* is used to indicate the network status. The green channel of *LEDO* is used to indicate the light status.



1. In a PuTTY session to your Matter hub, use the `mattertool` to test your Matter light device.

1. Using `mattertool on -n nodeID` (similarly, `mattertool off` and `mattertool toggle` can also be used) you can control the light status of your Matter light device.

```
ubuntu@ubuntu:~$ mattertool on -n 0xBBBB
[1684249271.557600][1606:1606] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1684249271.563328][1606:1606] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1684249271.563588][1606:1606] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1684249271.563641][1606:1606] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1684249271.564218][1606:1606] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini-51WzbY
[1684249271.564740][1606:1606] CHIP:DL: reading chip counters to file (/tmp/chip_counters.ini)
[1684249271.564791][1606:1606] CHIP:DL: NVS set: chip-counters/reboot-count = 8 (0x8)
[1684249271.565400][1606:1606] CHIP:DL: Got Ethernet interface: eth0
[1684249271.565888][1606:1606] CHIP:DL: Found the primary Ethernet interface:eth0
[1684249271.566169][1606:1606] CHIP:DL: Got WiFi interface: wlan0
[1684249271.566546][1606:1606] CHIP:DL: Failed to reset WiFi statistic counts
[1684249271.566706][1606:1606] CHIP:IN: UDP::Init bindlisted port=0
[1684249271.566824][1606:1606] CHIP:IN: UDP::Init bound to port=45069
[1684249271.566850][1606:1606] CHIP:IN: UDP::Init bindlisted port=0
[1684249271.566945][1606:1606] CHIP:IN: UDP::Init bound to port=37794
[1684249271.566971][1606:1606] CHIP:IN: BLEBase::Init - setting/overriding transport
[1684249271.566993][1606:1606] CHIP:IN: TransportMgr initialized
[1684249271.567000][1606:1606] CHIP:FFI: Initializing fabricTable from persistent storage
[1684249271.567111][1606:1606] CHIP:FFI: Last Known FabricTable: 2022-11-21T16:49:03
[1684249271.568785][1606:1606] CHIP:FFI: Index OAI has been retrieved from storage. Compressed FabricId 0xCDBDA0EB95F24134, FabricId 0x0000000000000001, NodeId 0x0000000000000001B669, VendorId 0xFFFF
[1684249271.571409][1606:1606] CHIP:ZCL: Using ZAP configuration...
[1684249271.578507][1606:1606] CHIP:DLS: MDNS failed to join multicast group on wpan0 for address type IPV4: ../../examples/chip-tool/third_party/connectedhomeip/src/inet/UDPEndPointImplSockets.cpp

[1684249271.613889][1606:1611] CHIP:CSM: FindOrEstablishSession: PeerId = 1:00000000000000BBB
[1684249271.613918][1606:1611] CHIP:CSM: FindOrEstablishSession: No existing OperationalSessionSetup instance found
[1684249271.613952][1606:1611] CHIP:CTL: OperationalSessionSetup[1:00000000000000BBB]: State change 1 --> 2
[1684249271.613986][1606:1611] CHIP:DLS: Lookup listening interface for new LL address
[1684249271.614243][1606:1611] CHIP:DLS: UDP:[fd11:22:2aef:fe1d:2ff7:ff20weth0]:5540: new best score: 6
[1684249271.615441][1606:1611] CHIP:DLS: Checking node lookup status after 1 ms

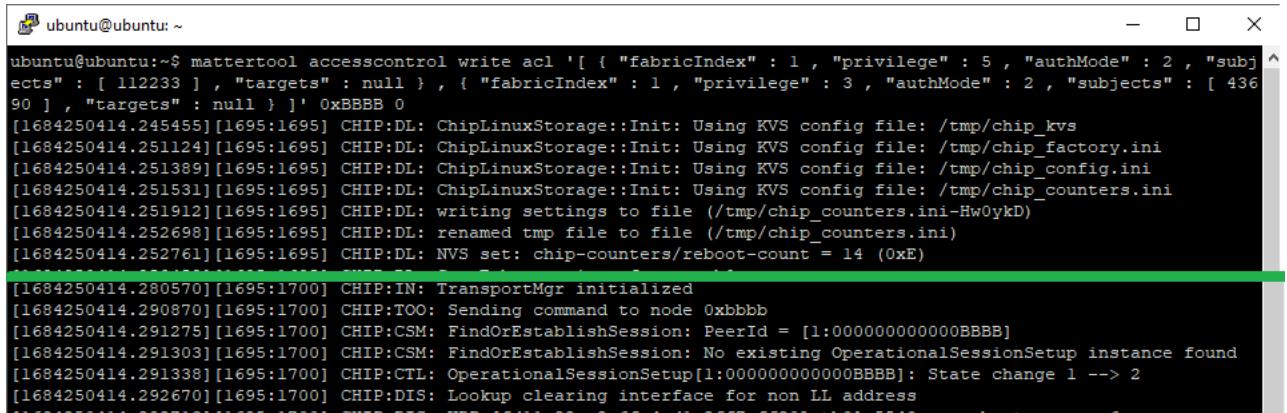
[1684249272.207730][1606:1611] CHIP:IN: SecureSession[0xfffffa000b030]: Activated - Type:2 LSID:3316
[1684249272.207791][1606:1611] CHIP:IN: New secure session activated for device <0x00000000000000BBB, 1>, LSID:3316 PSID:34900
[1684249272.280093][1606:1611] CHIP:CTL: OperationalSessionSetup[1:00000000000000BBB]: State change 4 --> 5
[1684249272.288072][1606:1611] CHIP:TOO: Sending cluster (0x00000006) command (0x00000001) on endpoint 1
[1684249272.290300][1606:1611] CHIP:DMG: ICR moving to [AddInC0mm]
[1684249272.290379][1606:1611] CHIP:DMG: ICR moving to [AddInC0mm]
[1684249272.298604][1606:1611] CHIP:EM: << [E:614371 M:165550076] (S) Msg TX to 1:00000000000000BBB [4134] --- Type 0001:08 (IM:InvokeCommandRequest)
[1684249272.298702][1606:1611] CHIP:IN: (S) Sending msg 165550076 on secure session with LSID: 3316
[1684249272.298902][1606:1611] CHIP:EM: ICR moving to [CommandSend]
[1684249272.298917][1606:1611] CHIP:DMG: ICR moving to [CommandSend]
[1684249272.298917][1606:1611] CHIP:EM: << [E:614361 M:236736867 (Ack:258557202)] (U) Msg TX to 0:00000000000000 [0000] --- Type 0000:10 (SecureChannel:StandaloneAck)
[1684249272.298945][1606:1611] CHIP:IN: (U) Sending msg 236736867 to IP address 'UDP:[fd11:22:2aef:fe1d:2ff7:ff20wpan0]:5540'
[1684249272.299030][1606:1611] CHIP:EM: Flushed pending ack for MessageCounter:258557202 on exchange 614361
[1684249272.391564][1606:1611] CHIP:EM: >> [E:614371 M:210431289 (Ack:165550076)] (S) Msg RX from 1:00000000000000BBB [4134] --- Type 0001:09 (IM:InvokeCommandResponse)
[1684249272.391600][1606:1611] CHIP:IN: Found matching exchange: 614371, Delegate: 0xfffffa000a78
[1684249272.391762][1606:1611] CHIP:EM: Rxd Ack: Removing MessageCounter:165550076 from Retrans Table on exchange 614371
[1684249272.391856][1606:1611] CHIP:DMG: ICR moving to [ResponseRe]
[1684249272.391931][1606:1611] CHIP:DMG: InvokeResponseMessage =
[1684249272.392003][1606:1611] CHIP:DMG: {
[1684249272.392013][1606:1611] CHIP:DMG: suppressResponse = false,
[1684249272.392013][1606:1611] CHIP:DMG: InvokeResponseIBs =
[1684249272.392166][1606:1611] CHIP:DMG: [
[1684249272.392335][1606:1611] CHIP:DMG: InvokeResponseIB =
[1684249272.392411][1606:1611] CHIP:DMG: {
[1684249272.392468][1606:1611] CHIP:DMG: CommandStatusIB =
[1684249272.392552][1606:1611] CHIP:DMG: {
[1684249272.392620][1606:1611] CHIP:DMG: CommandPathIB =
[1684249272.392703][1606:1611] CHIP:DMG: EndpointId = 0x1,
[1684249272.392789][1606:1611] CHIP:DMG: ClusterId = 0x6,
[1684249272.392880][1606:1611] CHIP:DMG: CommandId = 0x1,
[1684249272.392967][1606:1611] CHIP:DMG: },
[1684249272.393052][1606:1611] CHIP:DMG: ),
[1684249272.394141][1606:1611] CHIP:DMG: },
[1684249272.394260][1606:1611] CHIP:DMG: Received Command Response Status for Endpoint=1 Cluster=0x0000_0006 Command=0x0000_0001 Status=0x0
[1684249272.394102][1606:1611] CHIP:IN: ICR moving to [WaitingRe]
[1684249272.394407][1606:1611] CHIP:EM: << [E:614371 M:165550077 (Ack:210431289)] (S) Msg TX to 1:00000000000000BBB [4134] --- Type 0000:10 (SecureChannel:StandaloneAck)
[1684249272.394405][1606:1611] CHIP:IN: (S) Sending msg 165550077 on secure session with LSID: 3316
[1684249272.394464][1606:1611] CHIP:EM: Flushed pending ack for MessageCounter:210431289 on exchange 614371
[1684249272.395075][1606:1606] CHIP:CTL: Shutting down the commissioner
[1684249272.395142][1606:1606] CHIP:CTL: Stopping commissioning discovery over DNS-SD
[1684249272.395375][1606:1606] CHIP:CTL: Shutting down the controller
[1684249272.395428][1606:1606] CHIP:IN: Expiring all sessions for fabric Ox1!!
[1684249272.395428][1606:1606] CHIP:IN: SecureSession[0xfffffa000b030]: MarkForEviction Type:2 LSID:3316
[1684249272.395516][1606:1606] CHIP:SC: SecureSession[0xfffffa000b030]: Moving from state 'kActive' --> 'kPendingEviction'
[1684249272.395558][1606:1606] CHIP:IN: SecureSession[0xfffffa000b030]: Released - Type:2 LSID:3316
```

2. For dev board with buttons available, you can use BTN1 to toggle the light status locally.

2. In a PuTTY session to your Matter hub, use the `mattertool` to bind your Matter light switch device to your Matter light device, thus allowing the switch to control the light.

First, you will need to modify the Access Control List (ACL) of your Matter light device. This list determines which device in the network your Matter light device will react to. Modify your Matter light device's ACL using: `mattertool accesscontrol write acl '[{ "fabricIndex" : 1 , "privilege" : 5 , "authMode" : 2 , "subjects" : [112233] , "targets" : null } , { "fabricIndex" : 1 , "privilege" : 3 , "authMode" : 2 , "subjects" : [nodeID-switch] , "targets" : null }]' nodeID-light 0`, where the highlighted parameters are:

- **112233**: The node ID of the controller (OTBR). This is always 112233.
- **nodeID-switch**: The node ID of the Matter light switch device in base 10 (ex: 43690 for 0xAAAA).
- **nodeID-light**: The node ID of the Matter light device in hex (ex: 0xBBB).
- **0**: The endpoint in the Matter light device that holds the ACL. This is always 0.



```
ubuntu@ubuntu:~$ mattertool accesscontrol write acl '[ { "fabricIndex" : 1 , "privilege" : 5 , "authMode" : 2 , "subjects" : [ 112233 ] , "targets" : null } , { "fabricIndex" : 1 , "privilege" : 3 , "authMode" : 2 , "subjects" : [ 43690 ] , "targets" : null } ]' 0xBBB
[1684250414.245455] [1695:1695] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1684250414.251124] [1695:1695] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1684250414.251389] [1695:1695] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1684250414.251531] [1695:1695] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1684250414.251912] [1695:1695] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-HwOykD)
[1684250414.252698] [1695:1695] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)
[1684250414.252761] [1695:1695] CHIP:DL: NVS set: chip-counters/reboot-count = 14 (0xE)

[1684250414.280570] [1695:1700] CHIP:IN: TransportMgr initialized
[1684250414.290870] [1695:1700] CHIP:TOO: Sending command to node 0xbbbb
[1684250414.291275] [1695:1700] CHIP:CSM: FindOrEstablishSession: PeerId = [1:000000000000BBBB]
[1684250414.291303] [1695:1700] CHIP:CSM: FindOrEstablishSession: No existing OperationalSessionSetup instance found
[1684250414.291338] [1695:1700] CHIP:CTL: OperationalSessionSetup[1:000000000000BBBB]: State change 1 --> 2
[1684250414.292670] [1695:1700] CHIP:DIS: Lookup clearing interface for non LL address
[1684250414.292713] [1695:1700] CHIP:DIS: UDP::fd11::2::2::ff:4ad1:2ff7:ff20&eth01:5540: new best score: 6
```

Note: The ACL table action is a read/modify/write step. If you accidentally remove the chip-tool entry, this will prevent further control of the device, requiring a factory reset.

You can read the ACL for a Matter device using: `mattertool accesscontrol read acl nodeID 0`, where the highlighted parameters are:

- **nodeID**: The nodeID of the Matter device you wish to read the ACL contents from.
- **0**: The endpoint in the Matter device that holds the ACL. This is always 0.

```
ubuntu@ubuntu:~$ mattertool accesscontrol read acl 0xBBBB 0
[1684250640.139382][1709:1709] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1684250640.144840][1709:1709] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1684250640.145105][1709:1709] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1684250640.145217][1709:1709] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1684250640.145573][1709:1709] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-YwEytb)

[1684250640.170001][1709:1714] CHIP:IN: Transporting initialized
[1684250640.189917][1709:1714] CHIP:TOO: Sending command to node 0xbbbb
[1684250640.190382][1709:1714] CHIP:CSM: FindOrEstablishSession: PeerId = [1:000000000000BBBB]
[1684250640.190412][1709:1714] CHIP:CSM: FindOrEstablishSession: No existing OperationalSessionSetup instance found
[1684250640.190446][1709:1714] CHIP:CTL: OperationalSessionSetup[1:000000000000BBBB]: State change 1 --> 2
[1684250640.191829][1709:1714] CHIP:DIS: Lookup clearing interface for non LL address
[1684250640.191874][1709:1714] CHIP:DIS: UDP:[fd11:22::2a6f:4ed1:2ff7:ff20%eth0]:5540: new best score: 6
[1684250640.191903][1709:1714] CHIP:DIS: Checking node lookup status after 2 ms

[1684250640.931348][1709:1714] CHIP:DMG: AttributeReportIB =
[1684250640.931418][1709:1714] CHIP:DMG: {
[1684250640.931471][1709:1714] CHIP:DMG: AttributeDataIB =
[1684250640.931537][1709:1714] CHIP:DMG: {
[1684250640.931601][1709:1714] CHIP:DMG: DataVersion = 0x5125b161,
[1684250640.931662][1709:1714] CHIP:DMG: AttributePathIB =
[1684250640.931725][1709:1714] CHIP:DMG: {
[1684250640.931791][1709:1714] CHIP:DMG: Endpoint = 0x0,
[1684250640.931858][1709:1714] CHIP:DMG: Cluster = 0x1f,
[1684250640.931927][1709:1714] CHIP:DMG: Attribute = 0x0000_0000,
[1684250640.931992][1709:1714] CHIP:DMG: ListIndex = Null,
[1684250640.932055][1709:1714] CHIP:DMG: }
[1684250640.932122][1709:1714] CHIP:DMG: Data =
[1684250640.932183][1709:1714] CHIP:DMG: {
[1684250640.932246][1709:1714] CHIP:DMG: {
[1684250640.932310][1709:1714] CHIP:DMG: 0x1 = 3,
[1684250640.932379][1709:1714] CHIP:DMG: 0x2 = 2,
[1684250640.932443][1709:1714] CHIP:DMG: 0x3 = [
[1684250640.932513][1709:1714] CHIP:DMG: 43690,
[1684250640.932587][1709:1714] CHIP:DMG: ],
[1684250640.932655][1709:1714] CHIP:DMG: 0x4 = NULL
[1684250640.932722][1709:1714] CHIP:DMG: 0xfe = 1,
[1684250640.932790][1709:1714] CHIP:DMG: },
[1684250640.932850][1709:1714] CHIP:DMG: },
```

2. Second, you need to bind the switch's write command to the light. This is done by updating the binding table of your Matter light switch device. You do this with: mattertool binding write binding '[{ "fabricIndex": 1, "node": : **nodeID-light**, "endpoint": 1, "cluster": 6 }]' **nodeID-switch 1**, where the highlighted parameters are:

- **nodeID-light**: The node ID of the Matter light device.
- 1: The application endpoint in the light. In the Silabs examples, this is always 1.
- 6: The on/off cluster in the light. This is always 6.
- **nodeID-switch**: The node ID of the switch.
- 1: This is the application endpoint in the switch that holds the binding table. In the Silabs examples, this is always 1.

```
ubuntu@ubuntu:~$ mattertool binding write binding '[{"fabricIndex": 1, "node": 48059, "endpoint": 1, "cluster": 6}]' 0xAAAAA1
[1684251010.012247][1788:1788] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1684251010.017800][1788:1788] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1684251010.018054][1788:1788] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1684251010.018168][1788:1788] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1684251010.018514][1788:1788] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-VWw07X)
[1684251010.062428][1788:1793] CHIP:IN: TransportMgr initialized
[1684251010.070971][1788:1793] CHIP:TOO: Sending command to node 0xaaaa
[1684251010.071380][1788:1793] CHIP:CSM: FindOrEstablishSession: PeerId = [1:000000000000AAAA]
[1684251010.071415][1788:1793] CHIP:CSM: FindOrEstablishSession: No existing OperationalSessionSetup instance found
[1684251010.071449][1788:1793] CHIP:CTL: OperationalSessionSetup[1:000000000000AAAA]: State change 1 --> 2
[1684251010.073948][1788:1793] CHIP:DIS: Lookup clearing interface for non LL address
[1684251010.073991][1788:1793] CHIP:DIS: UDP:[fd11:22::3c4f:did=5067:16bd&eth0].5540: new best score: 6
[1684251010.0848056][1788:1793] CHIP:DMG: WriteClient moving to [ResponseRe]
[1684251010.0848103][1788:1793] CHIP:DMG: WriteResponseMessage =
[1684251010.0848122][1788:1793] CHIP:DMG: {
[1684251010.0848137][1788:1793] CHIP:DMG:     AttributeStatusIBs =
[1684251010.0848160][1788:1793] CHIP:DMG:     [
[1684251010.0848177][1788:1793] CHIP:DMG:         AttributeStatusIB =
[1684251010.0848198][1788:1793] CHIP:DMG:         {
[1684251010.0848216][1788:1793] CHIP:DMG:             AttributePathIB =
[1684251010.0848238][1788:1793] CHIP:DMG:             {
[1684251010.0848261][1788:1793] CHIP:DMG:                 Endpoint = 0x1,
[1684251010.0848284][1788:1793] CHIP:DMG:                 Cluster = 0xle,
[1684251010.0848306][1788:1793] CHIP:DMG:                 Attribute = 0x0000_0000,
[1684251010.0848327][1788:1793] CHIP:DMG:             }
[1684251010.0848351][1788:1793] CHIP:DMG:
[1684251010.0848372][1788:1793] CHIP:DMG:         StatusIB =
[1684251010.0848394][1788:1793] CHIP:DMG:         {
[1684251010.0848415][1788:1793] CHIP:DMG:             status = 0x00 (SUCCESS),
[1684251010.0848436][1788:1793] CHIP:DMG:         },
[1684251010.0848457][1788:1793] CHIP:DMG:
[1684251010.0848476][1788:1793] CHIP:DMG:     },
[1684251010.0848502][1788:1793] CHIP:DMG:
[1684251010.0848519][1788:1793] CHIP:DMG:     AttributeStatusIB =
[1684251010.0848538][1788:1793] CHIP:DMG:     {
[1684251010.0848557][1788:1793] CHIP:DMG:         AttributePathIB =
[1684251010.0848577][1788:1793] CHIP:DMG:         {
[1684251010.0848597][1788:1793] CHIP:DMG:             Endpoint = 0x1,
[1684251010.0848619][1788:1793] CHIP:DMG:             Cluster = 0xle,
[1684251010.0848641][1788:1793] CHIP:DMG:             Attribute = 0x0000_0000,
[1684251010.0848664][1788:1793] CHIP:DMG:             ListIndex = Null,
[1684251010.0848685][1788:1793] CHIP:DMG:         }
[1684251010.0848708][1788:1793] CHIP:DMG:
[1684251010.0848728][1788:1793] CHIP:DMG:     StatusIB =
[1684251010.0848748][1788:1793] CHIP:DMG:     {
[1684251010.0848769][1788:1793] CHIP:DMG:         status = 0x00 (SUCCESS),
[1684251010.0848790][1788:1793] CHIP:DMG:     },
[1684251010.0848811][1788:1793] CHIP:DMG:
[1684251010.0848829][1788:1793] CHIP:DMG:     },
[1684251010.0848850][1788:1793] CHIP:DMG:
[1684251010.0848867][1788:1793] CHIP:DMG:     ],
```

Note: As with the ACL table step, the binding table action is a read/modify/write step. If you accidentally remove the chip-tool entry, this will prevent further control of the device, requiring a factory reset.

You can read the binding table from a Matter device using: `mattertool binding read binding nodeID-switch 1`, where the highlighted parameters are:

- **nodeID-switch**: The node ID of the Matter switch.
 - **1**: The application endpoint in the switch that holds the binding table. In the Silabs examples, this is always 1.

```
ubuntu@ubuntu:~$ mattertool binding read binding 0xAAAA 1
[1684251152.333501][1800:1800] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1684251152.339025][1800:1800] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1684251152.339286][1800:1800] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1684251152.339402][1800:1800] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1684251152.339818][1800:1800] CHIP:DL: writing settings to file (/tmp/chip_counters.ini-EN8097)
[1684251152.340505][1800:1800] CHIP:DL: renamed tmp file to file (/tmp/chip_counters.ini)

[1684251152.363210][1800:1805] CHIP:IN: TransportMgr initialized
[1684251152.373261][1800:1805] CHIP:TOO: Sending command to node Oxaaaa
[1684251152.373756][1800:1805] CHIP:CSM: FindOrEstablishSession: PeerId = [1:000000000000AAAA]
[1684251152.373793][1800:1805] CHIP:CSM: FindOrEstablishSession: No existing OperationalSessionSetup instance found
[1684251152.373829][1800:1805] CHIP:CTL: OperationalSessionSetup[1:000000000000AAAA]: State change 1 --> 2
[1684251152.376289][1800:1805] CHIP:DIS: Lookup clearing interface for non LL address

[1684251153.097364][1800:1805] CHIP:DMG: },
[1684251153.097407][1800:1805] CHIP:DMG: AttributeReportIB =
[1684251153.097429][1800:1805] CHIP:DMG: {
[1684251153.097469][1800:1805] CHIP:DMG: AttributeDataIB =
[1684251153.097492][1800:1805] CHIP:DMG: {
[1684251153.097524][1800:1805] CHIP:DMG: DataVersion = 0xe1374bbe,
[1684251153.097550][1800:1805] CHIP:DMG: AttributePathIB =
[1684251153.097575][1800:1805] CHIP:DMG: {
[1684251153.097610][1800:1805] CHIP:DMG: Endpoint = 0x1,
[1684251153.097663][1800:1805] CHIP:DMG: Cluster = 0x1e,
[1684251153.097702][1800:1805] CHIP:DMG: Attribute = 0x0000_0000,
[1684251153.097731][1800:1805] CHIP:DMG: ListIndex = Null,
[1684251153.097767][1800:1805] CHIP:DMG: }
[1684251153.097794][1800:1805] CHIP:DMG: Data =
[1684251153.097823][1800:1805] CHIP:DMG: {
[1684251153.097860][1800:1805] CHIP:DMG: 0x1 = 48059,
[1684251153.097885][1800:1805] CHIP:DMG: 0x3 = 1,
[1684251153.097946][1800:1805] CHIP:DMG: 0x4 = 6,
[1684251153.097984][1800:1805] CHIP:DMG: 0xfe = 1,
[1684251153.098013][1800:1805] CHIP:DMG: },
[1684251153.098051][1800:1805] CHIP:DMG: },
[1684251153.098079][1800:1805] CHIP:DMG: },
```

With the binding complete, a button press on the Matter light switch device should now toggle the light status in the Matter light device.

Next Steps

Next Steps

Now that you have gotten a sense of what goes into making a Matter network, you can begin to customize MADs and other features for your own purposes. The [Developers' Guide](#) contains more detail, and also contains information on a number of special development topics. Your Matter Extension package contains a number of other Matter examples that you can use as a starting point, the first four of which were used in this example.

- Matter SoC Light over Wi-Fi
- Matter SoC Light Switch over Wi-Fi
- Matter SoC Light over Thread
- Matter SoC Light Switch over Thread
- Matter SoC Window Cover over Wi-Fi
- Matter SoC Lock over Wi-Fi
- Matter SoC Window Cover over Thread
- Matter SoC Lock over Thread
- Matter SoC Thermostat over Thread

If you need support, you can contact Silicon Labs through the [Silicon Labs Community](#). Our engineers and community will be happy to help! You may also find answers here:

- [Silicon Labs Matter articles](#)

Matter Fundamentals

Matter Fundamentals

This section contains information for those not yet familiar with Matter.

- [Introduction to Matter](#): Offers an overview for those new to Matter.
- [The Matter Data Model](#): Reviews the components of the Matter Data Model including nodes, endpoints, clusters, and device types.
- [The Matter Interaction Model](#): Describes how the model defines the methods of communication between nodes, serving as the common language for node-to-node information transmission.
- [Matter Security](#): Provides an overview of security for Matter as promoted by the Connectivity Standards Alliance (CSA).

Introduction to Matter

Introduction to Matter

Why Matter?

The Connectivity Standards Alliance (CSA) seeks to enable smart home devices to be secure, reliable, and interoperable with other Internet of Things (IoT) devices, regardless of manufacturer. One of the biggest pain points of smart home devices is the various application-layer implementations that limit the compatibility of different smart home devices. Therefore, Matter was adopted so that matter-enabled devices, under the same standard, provide a better experience for both the manufacturers and users.

While Matter was created to make the IoT seamless and effortless for users, it is important to acknowledge the complexity involved under the surface. Matter leverages a range of tools to enable seamless and secure connectivity within an IoT system. Understanding these tools is crucial for a comprehensive overview of this technology.

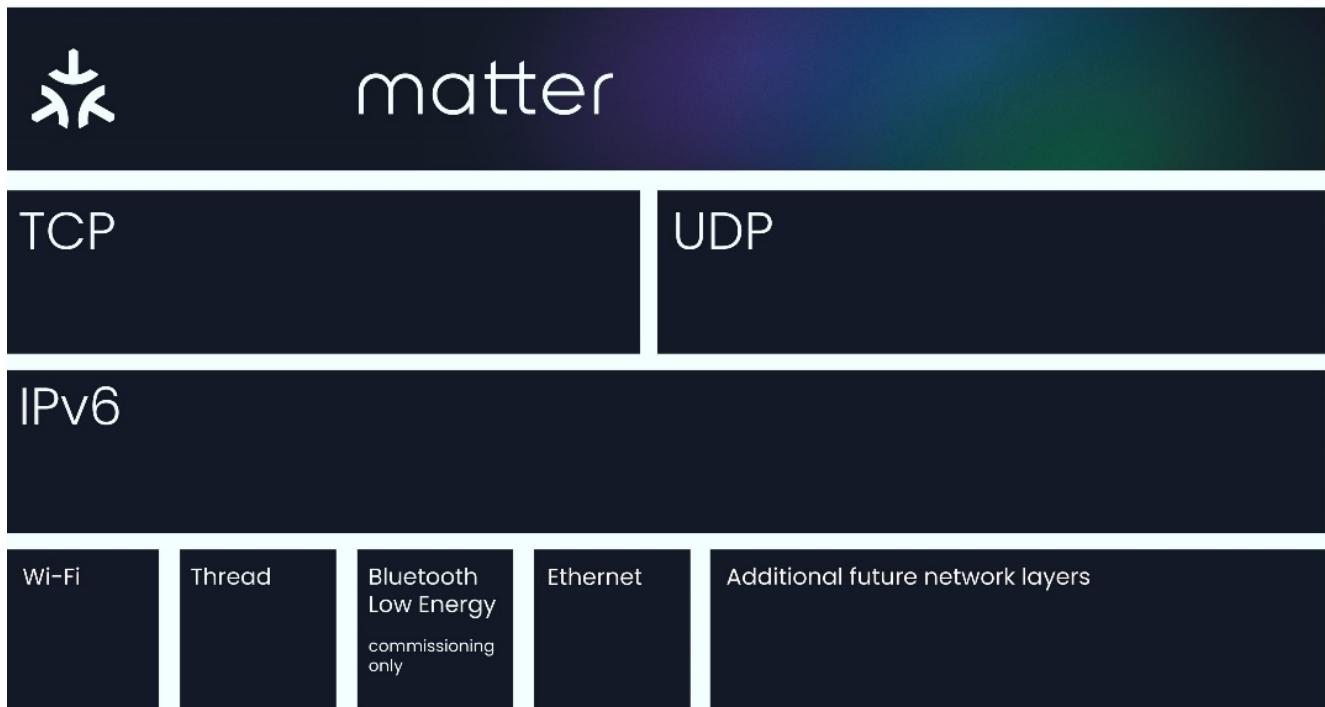
Matter Enablement

Matter implements the Application Layer of the Open Systems Interconnection (OSI) model. It builds on the lower layers, such as the transport and network layers, to enable reliable communication between nodes.

Matter may sit on top of two prominent connectivity technologies: Thread and Wi-Fi. Thread is a low-power wireless mesh networking protocol that facilitates reliable communication between nodes. It enables extended coverage and reliable connections to improve the overall performance of IoT ecosystems.

Matter is an IPv6-based protocol that utilizes transport layer protocols like TCP/UDP to facilitate network addressing and reliable transmission of data packets, respectively. Due to this, Matter is compatible with multiple connectivity options (such as Thread and Wi-Fi). This flexibility allows Matter-enabled devices to communicate over various network protocols, ensuring broad compatibility and integration with different networking technologies.

The true power of Matter lies in its commitment to interoperability. Industry leaders such as Google, Apple, Amazon, and Samsung Smart Things have implemented Matter in their IoT devices, fostering wider adoption from other manufacturers. Matter also supports bridging from other existing technologies, such as Zigbee, Bluetooth® Mesh and Z-Wave. This allows already existing IoT technology to be integrated into an interoperable environment. The compatibility and seamless integration across brands create a unified ecosystem where your smart devices work harmoniously.



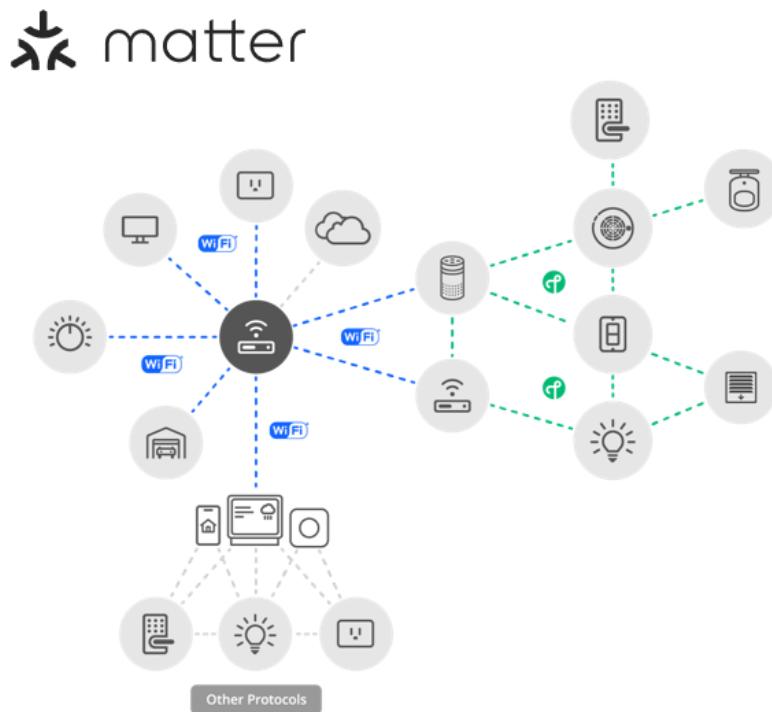
Matter Network Architecture

In theory, Matter can sit on top of any IPv6-bearing network. However in practice, the Matter specifications solely focus on three link layer technologies, enabling Matter to run on Ethernet, Wi-Fi, and Thread 802.15.4 networks. As mentioned above, one of the great benefits of Matter is its flexibility, especially when it comes to network configuration. The Matter protocol can operate without a globally-routable IPv6 infrastructure and allows the flexibility of having multiple Matter networks run over the same set of constituent IP Networks.

Two common underlying network topologies are commonly used in Matter. The first is known as a Single Network topology, where Matter runs solely over one Network. This means the Matter Network could run over one 802.15.4 Thread network or over a Wi-Fi network. In this scenario all Matter devices are connected to the same single logical network.



The other, more common, network topology is the star network, which consists of multiple peripheral networks joined together by a central hub network. If a peripheral network is used it must be directly joined to a hub via one or more border routers. A border router (or an edge router) is a special router that can provide routing services between two IP Subnets, effectively acting as a bridge between the two different networks. This enables a lot of flexibility and interoperability between various home networks that can all be interconnected.



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.

Below is a table of some basic Identifiers that are commonly used in Matter to identify and communicate with nodes on the fabric.

Identifiers	Definition
Fabric	64 bit number that uniquely identifies the Matter fabric.
Vendor	16 bit number that uniquely identifies a particular product manufacturer, vendor, or group thereof.
Product	16 bit number that uniquely identifies the product from a specific vendor.
Group	16 bit number ID set of nodes across a Matter Fabric
Universal Group	16 bit subrange of the Group ID reserved for groups across Matter Standard. Specifically a UID for all nodes, all non-ICD nodes, and all proxies.
Operational Node	64 bit number that uniquely identifies an individual node on the fabric
PAKE Key Identifiers	This is a subrange of Node ID used to assign an Access Control subject to a particular PAKE key. This creates an ACL (Access Control List) entry to provide admin access via a PASE session.

Matter Layered Architecture

Matter is split up into a layered architecture to help separate the different responsibilities and encapsulate various pieces of the protocol stack. The following diagram shows the various interactions between the Matter application stack layers as defined by the Matter CSA specification. For implementation purposes the last four layers are handled as a Messaging Layer and a Transport layer.



- The **Application Layer** of the Matter stack is the highest layer, and corresponds to the high-level logic of the device. The user application is built on the unified data model, which helps improve interoperability.
- The **Data Model** layer corresponds to how the data and action elements support the functionality of the application, such as the defines of the elements, namespaces for endpoints, clusters, and attributes in the application.
- The **Interaction Model** handles, as the name suggests, the interactions between the nodes and is responsible for how data is transferred between nodes. Both the Data Model and Interaction Model are inherited from the well-known dotdot standard used by Zigbee.
- The **Action Framing** layer is where the interactions are transformed into a message payload.
- The **Security Layer** takes the payload and encrypts and appends the packet with a MAC (Message authentication code).
- From the security layer, the packet transfers down to the Transport layer to what the CSA refers to as the **Message Framing and Routing** layer. This updates the payload with the necessary routing information such as fabric and Node ID.
- Finally the packet is sent to the **Transport and IP Framing** layer, from which the payload is sent through the IP network either through TCP (which is not yet currently supported) or Matter's Message Reliability Protocol, a software layer on top of UDP, for IP management of the data. Note that the IP Framing in Matter is handled by the Networking Protocol stack to handle this.

Once the data is received on a peer device, it travels up the Matter Protocol stack in reverse and delivers the message to the **Application** layer.

Matter Security

As mentioned before, one of the main benefits of using Matter is the enhanced security that it offers. Matter offers security at many layers throughout the network. Many security features are integrated into commissioning a Matter device onto an existing network. Matter requires all devices to have a device-specific passcode that delivers "proof of ownership" to commissioning devices and requires all devices to have immutable credentials that can be cryptographically verified to indicate that the joining devices are Matter-certified devices. In fact, the network credentials are only given to the Matter device after the device has been authenticated and verified. This helps keep the Matter network secure from foreign and unsecure devices, significantly improving the overall network security. Furthermore, Matter enables encryption and authentication to all unicast messages, as well as providing relay protection. Below is a list of commonly-used Matter security terms.

Concepts	Definition
DAC (Device Attestation Certificate)	This is an immutable certificate of credentials that can be cryptographically verified to confirm that the device is a certified Matter device.
PASE (Passcode Authentication Session Establishment)	This process at the Commissioning stage uses a passcode provided out-of-band (like a device's QR code) to commission a Matter device on the network.
CASE (Certificate Authentication Session Establishment)	This process at the Operational stage establishes and provides an authentication key exchanged between two devices.

More detailed information on Matter security can be found in [Matter Security Documentation](#) and [Matter Commissioning Documentation](#).

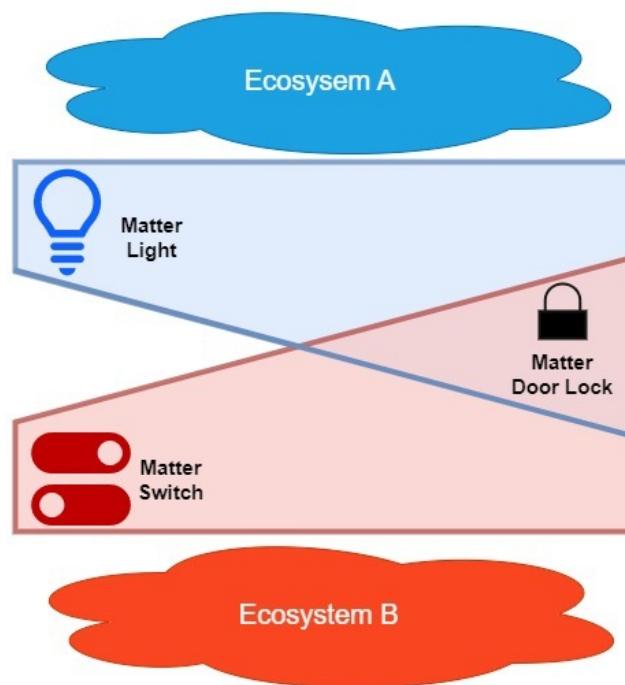
Matter Data Model

The Matter Data Model

The Data Model in Matter describes a hierarchical encapsulation of data elements in the Matter network, including, but not limited to, nodes, endpoints, clusters, and device types, where the node is the highest level data element.

A single physical Matter device, such as a light, switch, or door lock, can be represented by one or more nodes. An environment where multiple Matter nodes interoperate is referred to as a Matter fabric. These nodes share a common root of trust. On each separate fabric, a physical Matter device is represented by a node. Every node has a unique network address (Operational Node ID) that makes it uniquely identifiable in the fabric it is on. For example, in the following figure a Matter Lighting device on an Apple HomePod fabric (blue) has an operational node ID unique to the HomePod fabric and a Matter Switch device on a separate Samsung SmartThings Station fabric (red) in the same home has an operational node ID unique to the SmartThings fabric. These IDs may be the same or different; they are independent of each other because they are on two different fabrics.

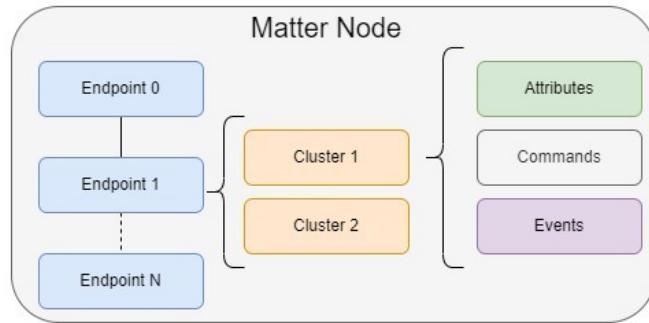
A Matter device may also be part of more than one fabric and thus is represented on each different fabric by a different node. In following figure, a Matter Door Lock device is on both the HomePod and SmartThings fabrics. Thus, the Door Lock is represented by two different nodes; one for the HomePod fabric and another for the SmartThings fabric. The operational node IDs for the two nodes representing the device may be the same or different; the IDs are independent of each other because the nodes are on two different fabrics.



The following section describes the Matter data model, including Nodes and Device Types.

Nodes

In the following figure, the hierarchical structure of endpoint, cluster, attributes/commands/events is shown from left to right.



Each node contains the complete application functionality for its device on a single stack. Because of this, nodes can communicate directly with other nodes on the network without the need for an intermediary.

Nodes have a set of related behaviors, known as a role. There are a few main node roles:

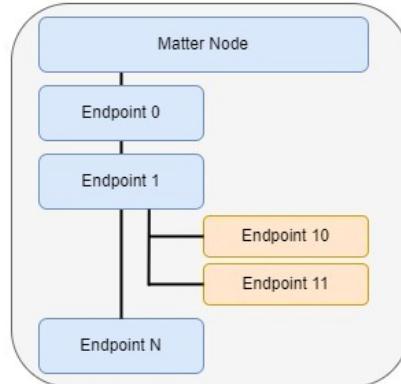
- **Commissioner** - Commissions/adds new devices to a Matter network
- **Controller** - Controls one or more nodes
- **Controlee** - Can be controlled by one or more nodes
- **Over the Air (OTA) Provider** - Provides OTA software updates to the OTA Requestor
- **OTA Requestor** - Requests OTA software updates from the OTA Provider

Endpoints

Endpoints enclose one component of a node's complete functionality; together, they encapsulate all the node's capabilities needed for functional wholeness. A smart thermostat, for example, could have two endpoints where one would implement the temperature control functionalities and another would implement the temperature monitoring functionalities.

Each of these endpoints are known as a feature set, which is made of clusters that define the attributes, events, and commands of a single endpoint's functionality. There are two distinct types of endpoints in Matter: leaf endpoints and composed endpoints.

- Leaf endpoints, such as Endpoints 0, 10, and 11 in the following figure, do not require other endpoints to function.
- Composed endpoints, such as Endpoint 1 in the following figure, require other endpoints to function. Connector lines indicate the endpoints that a certain endpoint has access to.



Nodes have numbered endpoints starting from 0 that contain their own feature set. Endpoints with the same number but on different nodes may enclose different feature sets. The exception is Endpoint 0, which is reserved exclusively for Utility

Clusters. These special clusters are specifically used for enclosing a node's servicing functionality: the discovery process, addressing, diagnostics, and software updates.

Endpoints are individually addressable to easily modify feature sets separately.

Clusters

Clusters are collections of data that group the attributes, events, and commands of a specific functionality, representing a single feature in an endpoint's feature set. A cluster may be thought of as an interface, service, or object class and is the lowest independent functional element in the device data model. Endpoints have multiple clusters to create individual instances of the same functionality for easier unit control. For example, each light on a light strip may have its own designated cluster for independent access.

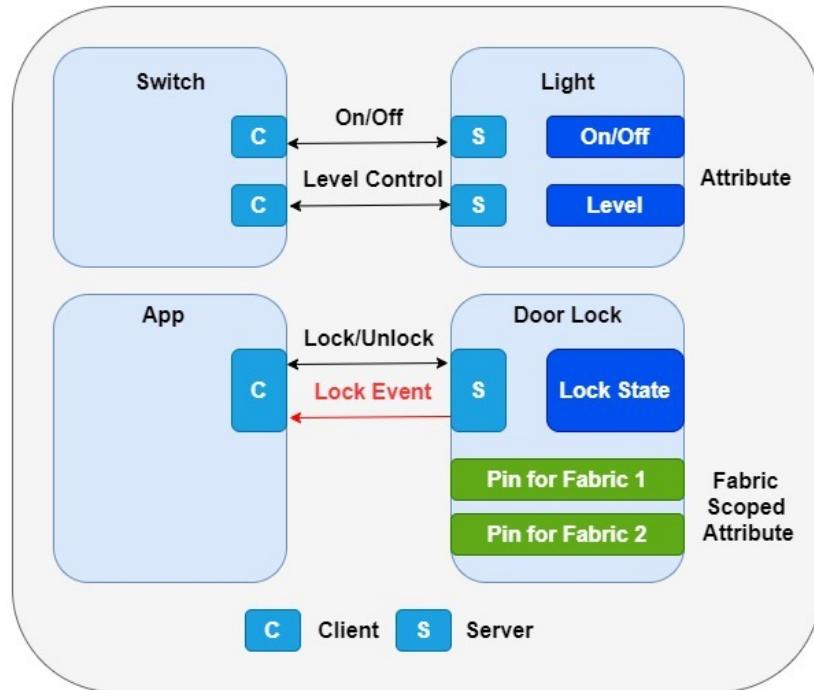
Attributes, events, and commands make up clusters:

- **Attributes** represent the current state, configuration, or capability of a node, for example whether a light is on or off, or if a switch is up or down.
 - Fabric-scoped attributes are only accessible to devices in the same fabric.
 - Attributes can be of uint8, string, array, etc. data types.
- **Commands** are actions that a cluster can perform, analogous to verbs in the English language.
 - Commands always have a direction, either from client to server or vice versa.
 - The target can reply to the command in one of two ways:
 - **Request** - such as toggling the ON/OFF attribute of a server cluster representing a light.
 - **Response** - such as sending a success status, or an error/unsupported status.
- **Events** are a record of past transitions between states of the node.
 - Events include data for a timestamp and priority of each change, as well as a monotonically increasing counter to track the number of state changes.
 - They are useful in capturing state transitions and modeling past data that attributes do not store.

Clusters have two main types:

- **Server cluster** - Stateful, holds the data for the attributes, events, and commands.
- **Client cluster** - Stateless, interacts with other server clusters by reading and writing attributes, reading remote events, and/or invoking methods.

Any cluster can be a server or a client, giving nodes the ability to both store information and horizontally communicate with other nodes. For a light and switch example, a client cluster in the light would send a command to a server cluster in the light to toggle the on/off feature of the light. The following figure illustrates examples of cluster communication. On the top is a light and switch example and on the bottom is an app controlling a door lock.



Device Types

Device types are a collection of clusters on their respective endpoints that define top-level attributes of the physical device they represent. Device types can require other types for operational purposes; these are known as composed device types and require composed endpoints.

A device can be made up of any combination of clusters. Therefore, to ensure the interoperability of devices from different manufacturers on a single network, Matter defined sets of requirements for official device types in the Matter Device Library for users to implement and extend. Every definition contains the device type ID, type revision number, and mandatory cluster(s) with their minimum revision number. Device Types are constantly updating, with each iteration tracked using the revision number starting from 1. However, changes to a device type's definition do not change its functionality, but only serve to improve operation. It is also important to note that newer versions of a device will continue to interoperate with older revision levels.

The device types that Matter supports are an unyielding requirement for Matter nodes. Nodes that implement certain device types are required to include feature sets of clusters on one or more endpoints for the said device type. A node cannot implement a device type if it does not have all the required feature sets. Official documentation lists the application device types that Matter supports in the Device Library Specification, while the respective supported application clusters are defined in the Application Cluster Library. These two documents, along with Chapter 7: Data Model Specification of the Matter specification document, can be found on the website for CSA members: [Specifications Download Request - CSA-IOT](https://groups.csa-iot.org/wg/matter-tsg/document/18649).

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 found here: <https://groups.csa-iot.org/wg/matter-tsg/document/18649> and Chapter 2 of the Zigbee Cluster Library Specification found here: <https://groups.csa-iot.org/wg/members-all/document/23019>. The Matter Data Model better defines the architecture in the Zigbee Cluster Library while keeping the certifiable cluster specifications.

Matter Interactions Model

The Matter Interaction Model

The Matter Device Interaction Model (IM) defines the methods of communication between nodes, and serves as the common language for node-to-node information transmission.

Nodes communicate with each other through interactions. Interactions are a sequence of transaction(s), which in turn are a sequence of actions.

For example, in a Read Interaction, a client cluster can initiate a Read Transaction, where the client can request to read an attribute and a server cluster can respond by reporting the attribute. Both the client request and the server response are separate actions, but they are part of the same Read Transaction, which the Read Interaction encompasses.



The Interaction Model supports four types of interactions:

- Read
- Write
- Invoke
- Subscribe

All interaction types except Subscribe 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.

The following concepts are important for understanding transactions.

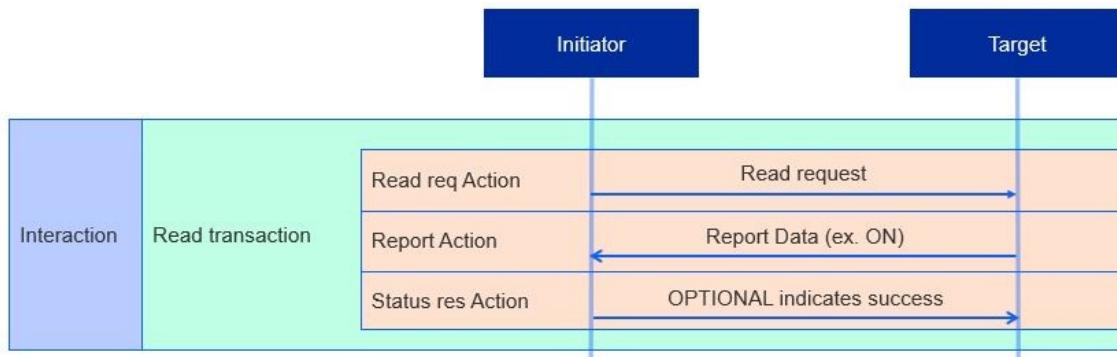
- **Initiators and Targets** - Interactions happen between an initiator node and target node(s). The initiator starts the transaction, and the target responds to the initiator's action. More specifically, the transaction is usually between a client cluster on the initiator node and a server cluster on 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 of devices allow an initiator to send an action to multiple targets. This group-level 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. Examples of path assembly:
 - <path> = <node> <endpoint> <cluster> <attribute / event / command>
 - <path> = <group ID> <attribute / event / command>

When groupcasting, a path may include the group or a wildcard operator to address several nodes simultaneously, decreasing the number of actions required and thus decreasing the response time of an interaction. Without groupcasting, humans may perceive latency between multiple devices reacting to an interaction. For example, when turning off a strip of lights, a path would include the group containing all the lights instead of turning off each light individually.

The following sections review each of the four interaction types and their constituent transactions and actions.

The Read Interaction

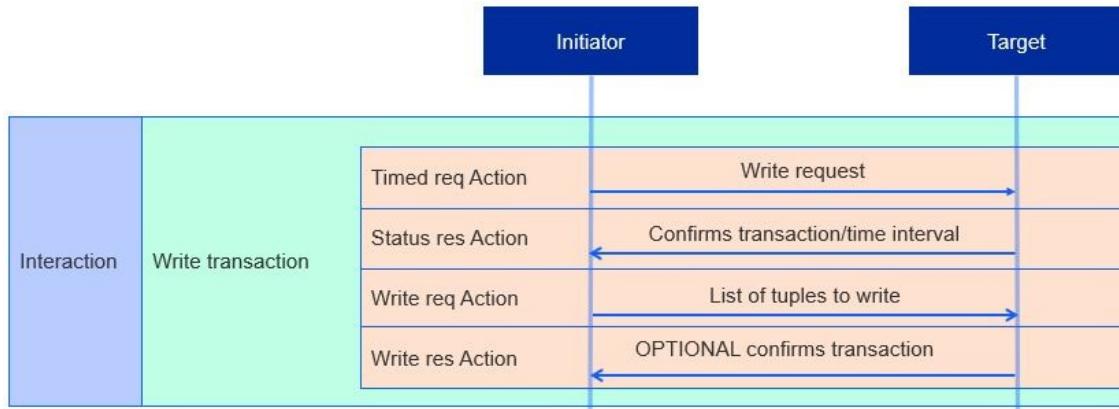


An initiator starts a Read Interaction when it wants to determine the value of one or more of a target node's attributes or events. The following steps occur:

1. **Read Request Action** - Requests a list of the target's attributes and/or events, along with paths to each
2. **Report Data Action** - Generated in response to the Read Request Action. Target sends back the requested list of attributes and/or events, a suppress response, and a subscription ID.
 1. Suppress response: Flag that indicates whether the status response should be sent or withheld.
 2. Subscription ID: Integer that identifies the subscription transaction, only included if the report is part of a Subscription Transaction.
3. **Status Response Action (OPTIONAL)** - Generates a Status Response by default; however, not sent if the suppress response flag is set. Ends transaction once the initiator sends the Status Response or receives a Report Data with the suppress flag set.

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

The Write Interaction



An initiator modifies a target's attributes through a Write Interaction, which consists of either a Timed or Untimed Write Transaction.

An untimed transaction remains open to the receiver for an indefinite period, whereas a timed transaction establishes a maximum period (usually a few seconds) to receive a return action.

Timed Transactions and Security

Timed transactions are mainly used for devices such as doors or locks because they protect assets and thus are a greater target for intercept attacks. To understand why timed transactions are effective it is important to understand the nature of intercept attacks:

1. The initiator node sends an initial message directed to the target node.
2. An attacker intercepts the message and holds it, preventing the message from reaching the target.
3. Since the initiator did not receive a message back from the target, the initiator sends another message.
4. The attacker intercepts this second message and sends the first message to the target, keeping the second message for later use.
5. The target receives the first message as if it were arriving from the initiator node, sending a confirmation response to the initiator node and, unknowingly, the attacker.

The problem lies in the second message; since the target never received the second message, the attacker now has a valid message to use at its convenience. The message may elicit a response from the target node such as "unlock" or "open door," which means that the network now has a breach in security. By establishing a maximum period to receive a message back, a timed transaction effectively guards against intercept attacks. The attacker can no longer hold a message to use at its convenience, as the message will expire after a set time.

Although timed transactions are important in guarding against attacks, they increase the complexity of a network since they need more actions. Therefore, they are only recommended for use on transactions that give access to valuable information.

Timed Write Transactions

A Timed Write Transaction consists of the following sequence of actions:

1. **Timed Request Action** - Sets the time interval to send a Write Request Action.
 2. **Status Response Action** - Confirms the transaction and time interval.
 3. **Write Request Action** - Requests three items:
 1. List of tuples (each tuple is called a write request) containing the path and data to be modified.
 2. Timed request flag indicating if the transaction is timed.
 3. Suppress response flag.
- If the transaction is timed and a timed request flag is set, the initiator must also send a timeout: the number of milliseconds the transaction remains open, during which the next action to be received is still valid.
4. **Write Response Action (OPTIONAL)** - A list of paths or error codes for every write request. Like a Read Transaction Status Response, a Write Response is not sent if the suppress response flag is set.

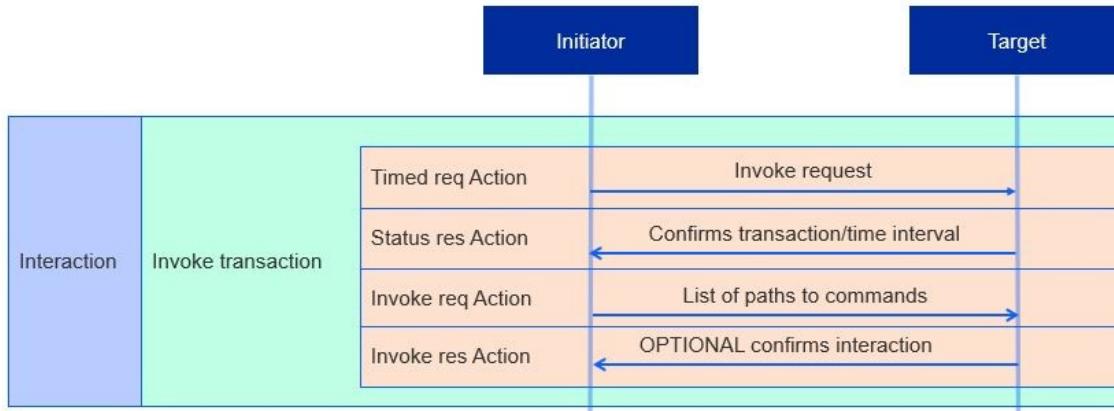
Untimed Write Transactions

An Untimed Write Transaction requires only the Write Request Action and the Write Response Action Timed Write Transaction since there is no time interval that needs to be set or confirmed.

Write Transaction Restrictions

Untimed and timed Write Transactions differ in their restrictions. All actions in timed transactions are unicast-only, whereas Untimed Write Request Actions may be multicast but require the Suppress Response flag to be set to prevent the network from flooding with status responses.

Invoke Interaction



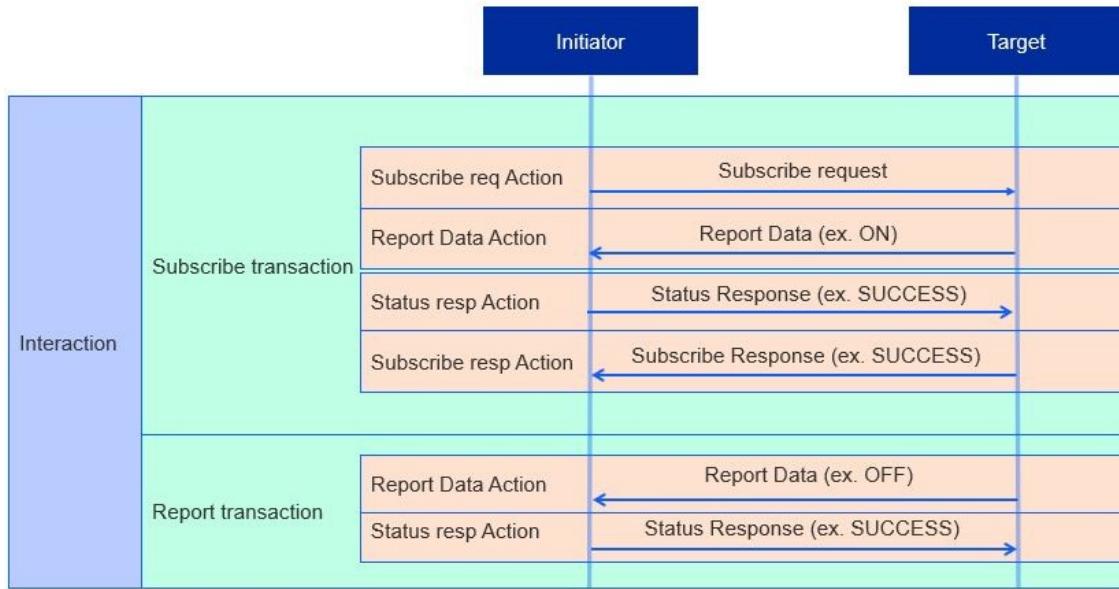
An initiator invokes command(s) on a target's cluster(s) through Invoke Interactions. An Invoke Interaction consists of either a Timed or Untimed Invoke Transaction, just like a Write Interaction consists of a Timed or Untimed Write Transaction.

Just like a Timed Write Transaction, a Timed Invoke Transaction consists of the following steps:

1. **Timed Request Action** - Sets the time interval to send a Write Request Action.
 2. **Status Response Action** - Confirms the transaction and time interval.
 3. **Invoke Request Action** - Requests four items:
 1. List of paths to cluster commands (each item in the list is an invoke command which may optionally contain argument(s) for the command).
 2. Timed request flag.
 3. Suppress response flag.
 4. Interaction ID: Integer to match the Invoke Request to its corresponding Invoke Response.
- An Invoke Request initiating a timed Invoke Transaction must also send a timeout just like a timed Write Transaction.
4. **Invoke Response (OPTIONAL)** - Target responds by sending back the interaction ID and a list of invoke responses: command responses and statuses for each invoke request. Like a Write Response, an Invoke Response is not sent if the suppress response flag is set

Untimed and timed Invoke Transactions differ in the same way that untimed and timed Write Transactions differ, both in their actions and restrictions on unicast or multicast.

Subscription Interaction



An initiator uses a Subscription Interaction to automatically receive periodic Report Data Transactions from the target. This creates a relationship between the initiator and target, which are referred to respectively as the subscriber and publisher after the subscription has been made.

Subscription Interactions include two transactions types: A Subscribe Transaction and Report Transaction.

Subscribe Transaction

The Subscribe Transaction is as follows:

- 1. Subscribe Request Action** - Requests three items:
 - Min interval floor (minimum interval between Data Reports).
 - Max interval ceiling (maximum interval between Data Reports).
 - Request for attributes and/or events to be reported.
- 2. Subscribe Request Action** - A Report Data Action containing the first batch of data, known as the Primed Published Data.
- 3. Status Response Action** - Acknowledges the Report Data Action.
- 4. Subscribe Response Action** - Finalizes the subscription ID (an integer that acts as an identifier for the subscription) and the min interval floor and max interval ceiling. Indicates a successful subscription between the subscriber and publisher.

Report Transaction

After a successful subscription, Report Transactions are sent to the subscriber. There are two types of Report Transactions: non-empty and empty.

- 1. Non-empty**
 - Report Data Action - Reports data and/or events with the SuppressResponse flag set to FALSE
 - Status Response - Indicates a successful report or an error, the latter of which ends the interaction
- 2. Empty**
 - Report Data Action - A report that has no data or events with the SuppressResponse flag set to TRUE, meaning no Status Response.

Subscription Interaction Restrictions

Subscription Interactions have a few restrictions.

- First, the Subscribe Request and Subscribe Response actions are unicast-only, meaning an initiator cannot subscribe to more than one target simultaneously.
- Second, Report Data Actions in the same Subscription Interaction must have the same subscription ID.
- Third, a subscription may be ended if the subscriber responds to a Report Data Action with an "INACTIVE_SUBSCRIPTION" status or if the subscriber does not receive a Report Data Action within the max interval ceiling. The latter connotes that the publisher may end a subscription by not sending Report Data Actions.

Relating Matter to Zigbee

The Matter Interaction Model originates from Chapter 2 in the Zigbee Cluster Library found here: <https://groups.csaiot.org/wg/members-all/document/23019>; the Interaction model extends this by abstracting interactions from other layers (security, transport, message format, encoding). The Interaction Model fills in gaps in the Zigbee Cluster Library identified by the Matter Data Model Tiger Team, adding Multi-Element Message support, Synchronized Reporting, reduce message types in commands and actions, complex data type support in all messages, Events, and interception attack.

Matter Specifications

More information on the Matter Interaction Model can be found in Chapter 8 of Matter Core Specifications [Specifications Download Request - CSA-IOT](#).

Matter Security

Matter Security

Matter raises the bar on security to a new level beyond simply guaranteeing the communication pipe is secure. Now, the end device must be proven to be authentic. The Matter Node Security will likely raise over time. As threats evolve, the SHOULDs will become SHALLs. Creating Secure Identities and injecting them securely in your manufacturing process is not trivial and can be costly. Silicon Labs has the hardware, software, and services to get your secure Matter products to market quickly and cost effectively.

Register at [Silicon Labs Tech Talks](#) to watch a detailed on-demand discussion of Matter Security, along with other tech talks as part of the Interactive Matter Training Series.

Note: All graphics were extracted from the Tech Talk, *Future-Proofing Matter Security with Secure Vault*, created by the Connectivity Standards Alliance (CSA) and used with permission.

Principles

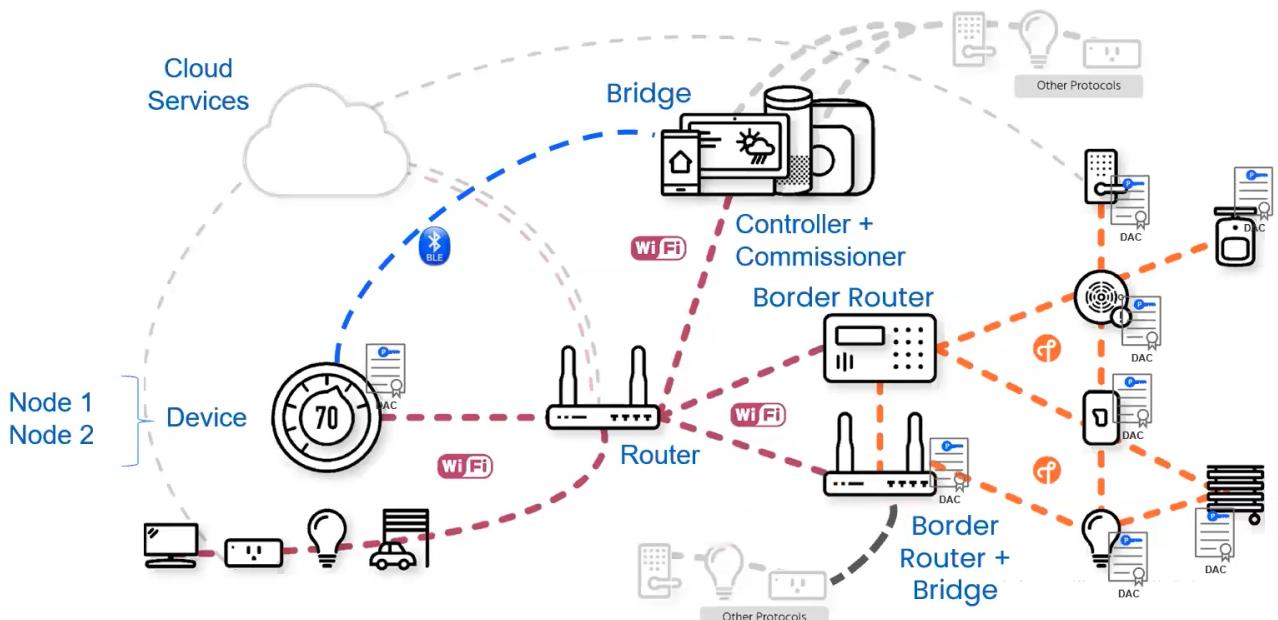
The following are the guiding principles for the Matter security design:

1. **No anonymous joining:** Always requires “proof of ownership” (that is, a device-specific passcode).
2. **Device Attestation:** Every device has unique identity that is authenticated by the manufacturer and verified through the CSA as a certified device.
3. **Operational Credentials:** When commissioned onto a Matter network, every device is given unique operational credentials after verifying their manufacturer credentials.
4. **Network Credentials:** The Wi-Fi network key or Thread Master key are not given until the device’s certificate is verified and authenticated properly.
5. **Open standard:** The open-source software is open to third parties vetting the claims by examining the standard and auditing the source code.

Security Tenets Promoted by the Connectivity Standards Alliance (CSA)

1. Easy, secure, and flexible device commissioning
2. Validation that each device is authentic and certified
3. Up-to-date information via Distributed Compliance Ledger
4. Strong device identity so only your devices can join
5. Secured communications protecting confidentiality, etc.
6. Even group communications secured
7. Multiple administrators and controllers, maximizing choice
8. Verified access controls to prevent unauthorized actions
9. Secured, standard software updates
10. Remote monitoring of software integrity

Matter Security Relevant Nomenclature



- **Node:** An independently addressable entity on a Matter network, which must be running an approved protocol (eg, Wi-Fi and Thread).
- **Device:** Anything that you take out of a box, such as a thermostat. Devices can have multiple nodes.
- **Router:** A standard router found in homes.
- **Controller:** Controls multiple nodes on a network.
- **Commissioner:** Can commission multiple nodes on a network in three ways:
 - BLE (most common)
 - WiFi AP protocol
 - Ethernet

Once commissioned, devices can start communicating on the Matter network via an approved protocol, usually WiFi.

A **Bridge** bridges to other protocols, such as Zigbee or ZWave to allow other devices communicate in the network.

A **Border Router** is intended to perform the communication protocol translation between approved protocols. It does not bridge between other protocols.

Device Identity starts with an identity within the device called a device attestation certificate (DAC). Any device that needs to be commissioned needs a DAC. If the device is inherently trusted within the ecosystem that it's trying to join, it does not need a DAC. If not, it does need a DAC.

End devices are what need to be trusted.

Matter Security Provisioning

Certificates and Process Overview

Each Matter device gets two certificates. The first, the **device certificate**, is programmed by the manufacturer before the device is shipped. This will be used later for device attestation when trying to join the network. The other, the **operational certificate**, is assigned by the commissioner in the commissioning stage. Certificates natively use a CHIP TLV format but can convert to/from X.509 format. All devices are given an operational certificate to prove their authorization on the Matter network (fabric) and securely identify them.

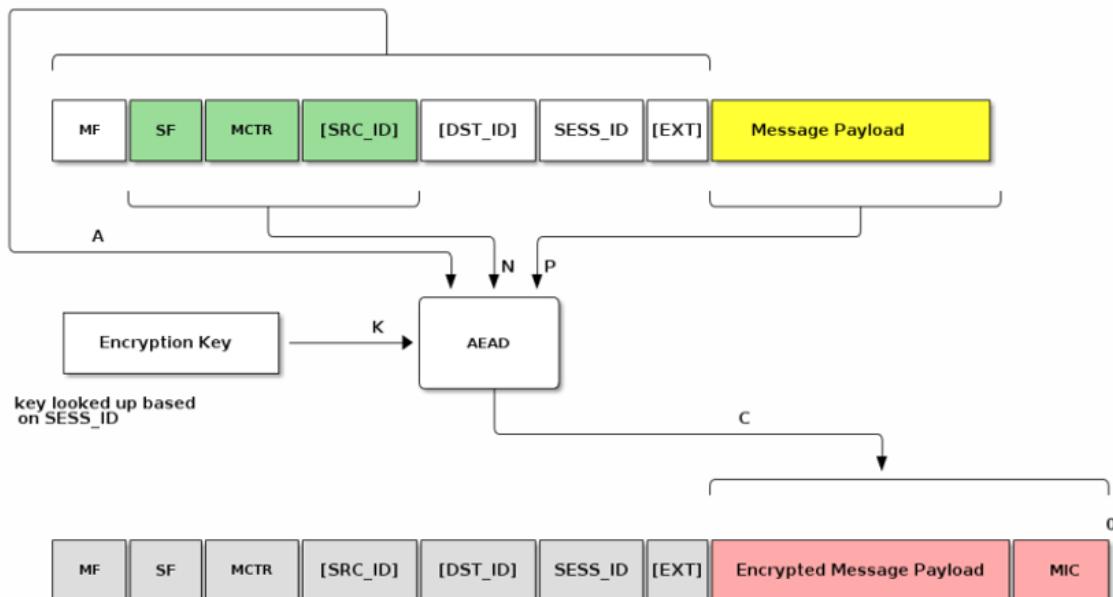
Communication between Matter devices is protected with different keys in different stages. At the commissioning stage, the key is a result of the Password Authenticated Session Establishment (**PASE**) process over the commissioning channel

using the passcode from the device's QR code as the input. During this initial setup, verification of possession of the passcode by both commissioner and joining device is confirmed. At the operational stage, the key is a result of the Certificate Authenticated Session Establishment (**CASE**) process over the operational channel using the operational certificate as the input. These sessions are used during normal operation between controller and device to validate that both are part of the Matter network.

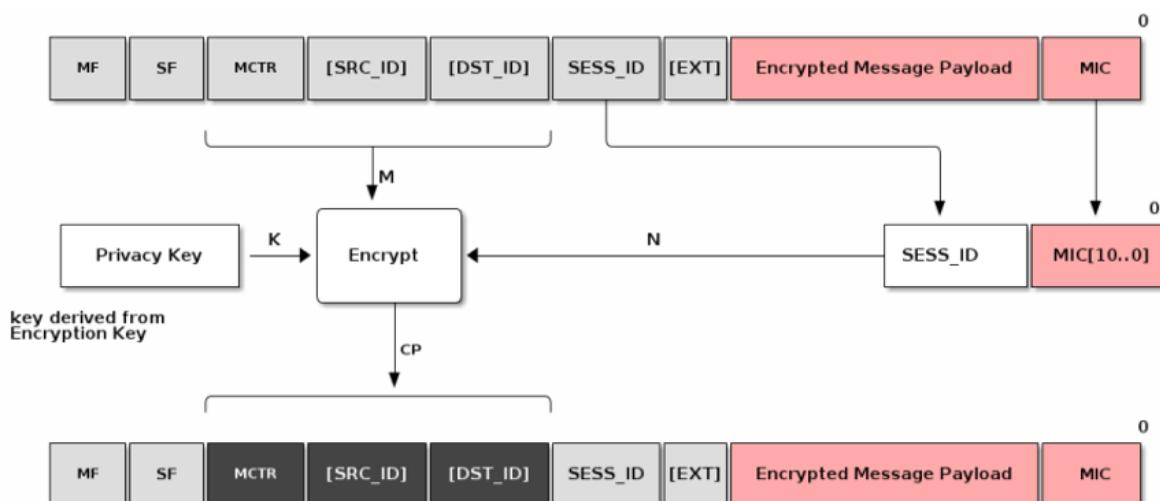
Message Protection

Various cryptographic algorithms are used to ensure communication security and integrity. These include:

- **Hashing Algorithm:** SHA-256
- **Message Authentication:** HMAC-SHA-256
- **Public Key:** ECC Curve NIST P-256
- **Message Encryption:** AES-CCM (128 bit keys)



Confidentiality: Message payload is encrypted by the encryption key (AES)



Privacy: Addresses are encrypted by the privacy key

Onboarding Payload

The Onboarding Payload is the information used by the Commissioner to ensure interoperability between commissioners and devices. It can be encoded in different formats:

- Human-readable (numeric string)
- Machine-readable (QR code and NFC tag)



Onboarding Payload Element	Description
Version	Provides versioning of the payload.
Vendor ID	Assigned by CSA. Allows a way to identify the maker of the device.
Product ID	Vendor specified. Unique for each certified product within a Vendor ID.
Custom Flow	<p>Indicates to the Commissioner the steps needed before commissioning can take place.</p> <ul style="list-style-type: none"> • Standard commissioning flow: A device, when uncommissioned, always enters commissioning mode upon power-up. • User-intent commissioning flow: Device requires user action (pressing a button, for example) to enter commissioning mode. • Custom commissioning flow: Interaction with a service provided by the manufacturer is required for initial device setup.
Discovery Capabilities Bitmask	<p>Indicate device's available technologies for device discovery:</p> <ul style="list-style-type: none"> • Soft-AP • BLE • On IP Network (device is already on the IP network)

Onboarding Payload Element	Description
Discriminator	Helps to further identify potential devices during the setup process.
Passcode	Establishes proof of possession and is also used as the shared secret for the initial secure channel before further onboarding steps.
TLV Data	(Optional) TLV (Tag-length-value) data. Indicates manufacturer-specific information elements and/or elements common to Matter. For instance, kTag_NumberOfDevices: Number of devices that are expected to be onboarded using this payload when using the Enhanced Commissioning Method.

Commissioning Steps

There are four steps involved in commissioning devices to start communicating on a Matter network:

1. Device Discovery
2. Secure Channel (PASE)
3. Device Attestation
4. Configuration

1. Device Discovery

The focus of this phase is getting the onboarding payload. The steps are:

1. The Device announces its availability for commissioning over initial network.
2. The Commissioner finds the Device.
3. The Commissioner connects to the Device, using:
 - Discriminator
 - Vendor ID (optional)
 - Product ID (optional)

2. Secure Channel (PASE)

The focus of this phase is establishing a secure link. The steps are:

1. The Commissioner establishes a secure unicast channel to the Device.
2. Protocol PASE = Password Authenticated Session Establishment
3. Based on SPAKE2+ protocol
4. Uses:
 - Passcode
 - Verifier

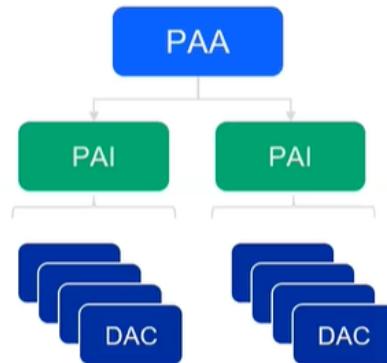
Several key constructs are important to understand prior to understanding Device Attestation.

Public Key Infrastructure (PKI)

A PKI is a set of roles, policies, and procedures used to create, manage, distribute, and revoke digital certificates and manage public-key encryption. The Matter Certificate Policy defines the rules and methods by which the Matter PKI Policy Authority (PKI-PA) is governed.

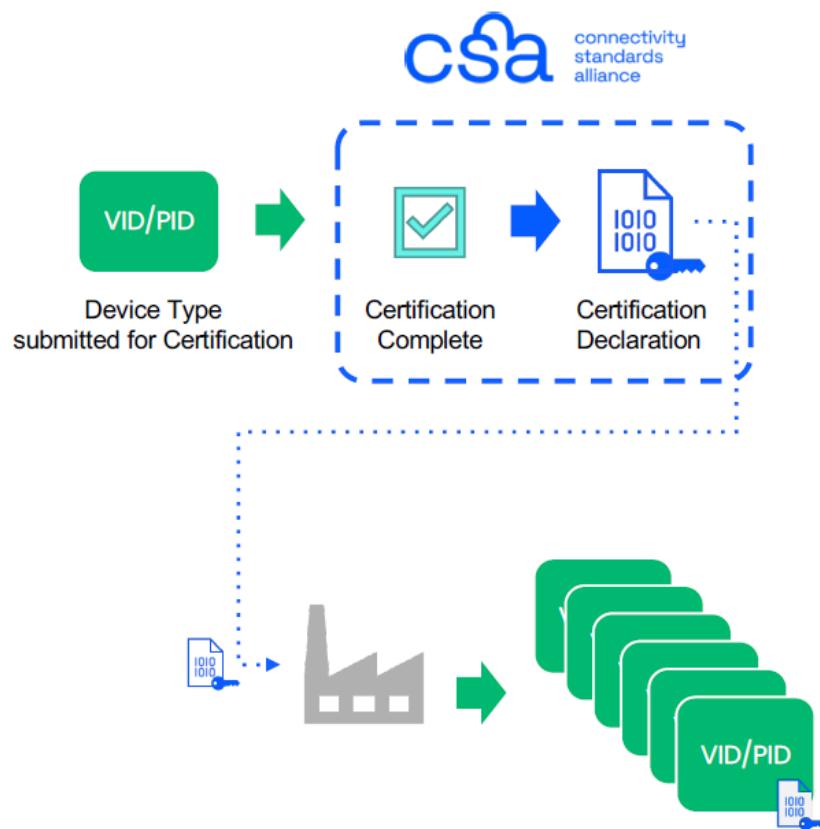
The Matter PKI for Device Attestation is comprised of:

- Certificate authorities:
 - PAA (Product Attestation Authority)
 - PAI (Product Attestation Intermediate)
- Authorized entities:
 - DAC (Device Attestation Certificate)



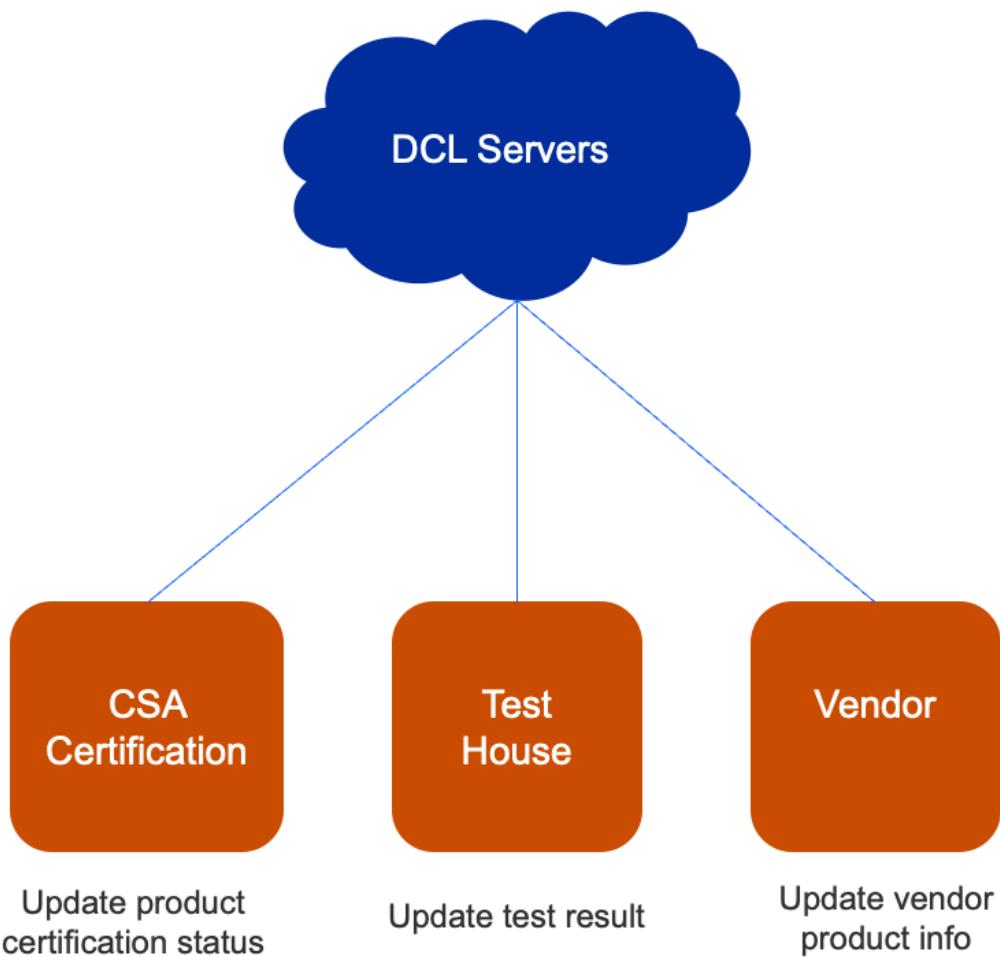
Certification Declaration

Another data construct that is necessary for Device Attestation is the Certification Declaration (CD), which is cryptographically signed by CSA and contains the Vendor and Device information as well as the PAA of the device. The CD must be put into the Device during manufacturing to be used during the Device Attestation process. The Commissioner will ask for the stored CD during the commissioning of the Node.



Distributed Compliance Ledger

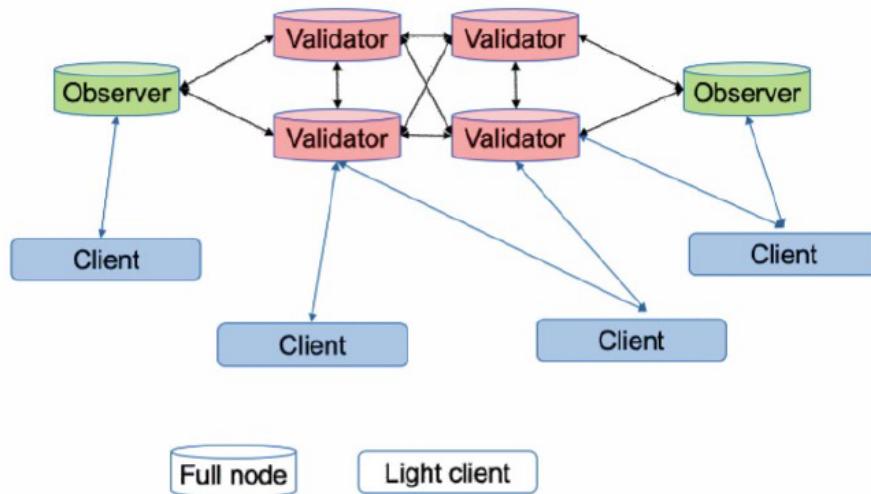
The Distributed Compliance Ledger (DCL) is the immutable single source of truth. It is a private blockchain-based distributed ledger of data records. Reading from the DCL is open to public, but writing to the DCL is restricted to various parties/roles. These roles typically include CSA certification, test house, and vendor roles.



It contains records about all certified devices, such as:

- Certification status
- Vendor ID (VID)
- Product ID (PID)
- Product name
- Part number and version
- Software and firmware versions
- Special commissioning instructions
- URLs to product pages and user manual

It also contains the Root PAA certificate for that Device, which is needed to complete the Device certificate chain verification. The DCL also contains the Certification Declaration ID number that will be compared with the CD pulled from the Device.

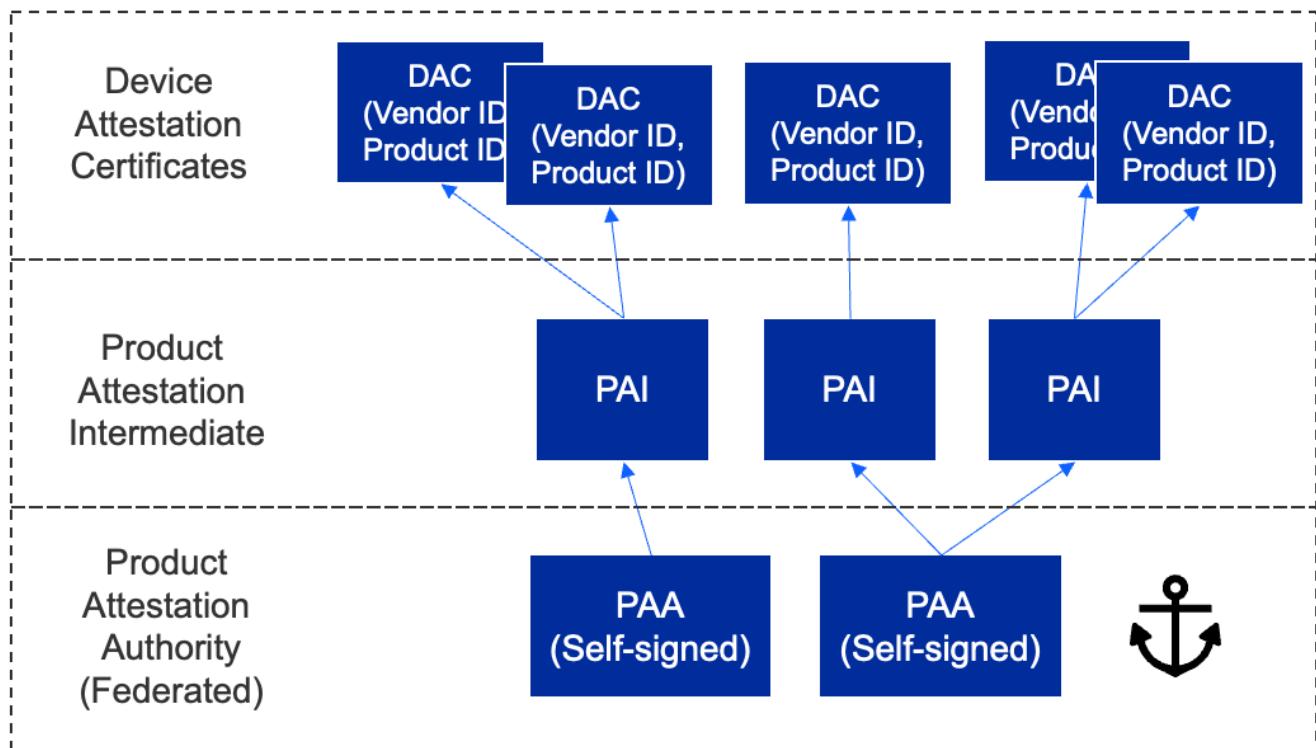


3. Device Attestation

Every device has a unique certificate that is signed by the manufacturer. There is no single root CA across all devices. During commissioning, the device is challenged to prove possession of the associated private key. The certificate can be validated against the Distributed Compliance Ledger (DCL) to verify device certification status.

The hierarchy allows for a 3-level tier:

- The first level is the Product Attestation Authority (PAA).
- The PAA will be used to sign the Product Attestation Intermediate (PAI).
- The PAI will be used to sign the Device Attestation Certificate (DAC). The DAC will be transferred to the commissioner and verified against the DCL.



The focus of this phase is to verify the authenticity of the Device. The high-level steps are:

1. The Commissioner verifies the Device's:
 - VID
 - PID
 - Certification status
2. To do so, it uses:
 - Device Attestation Credentials
 - Distributed Compliance Ledger (DCL) or
 - Certification Declaration (CD)

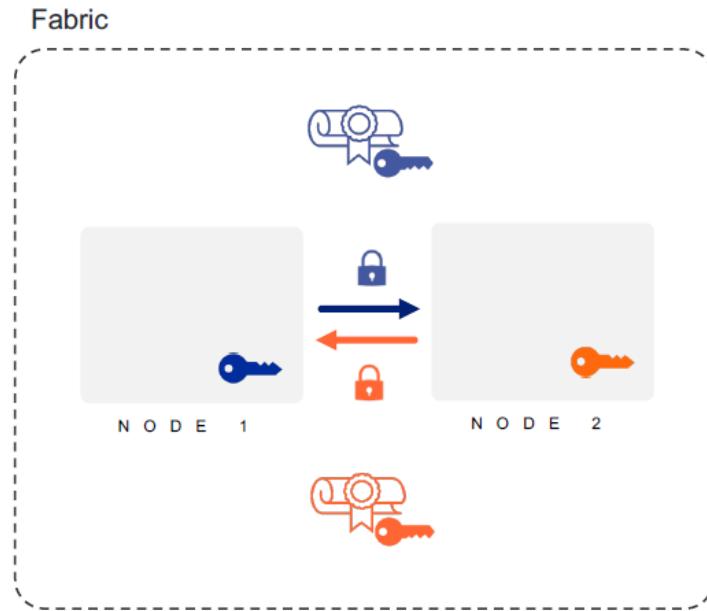


DAC is retrieved and verified before the device joins the Thread or Wi-Fi network. The Commissioner issues a challenge to the device to prove it possesses the associated Private Key.

First, the Commissioner asks the Node for the CD, the PAI Certificate, and the DAC. It then pulls the Certificate ID, the PAA Certificate, and the Device VID/PID from the immutable root of trust DCL. At that point, it has all the information needed to perform the device attestation. The Commissioner then runs a certification chain check from the DAC to the PAI, and all certificates should chain together correctly. If that check is passed, the Commissioner takes the certification ID from the DCL and checks it against the CD ID that it pulled from the device itself to make sure the device is a genuine CSA certified device. The final step is to verify that the public key in the DAC pulled from the Matter device mathematically matches the private key inserted in the device during manufacture. This is done by sending a message to the device during this final step of Device Attestation, and having the message signed by the device and then the signature verified using the public key from the DAC.

Node Operational Credentials

The Node Operational Credentials enable a Node to identify itself within a Fabric. A Node receives its initial set of Node Operational Credentials when it is commissioned to a Fabric by a Commissioner.



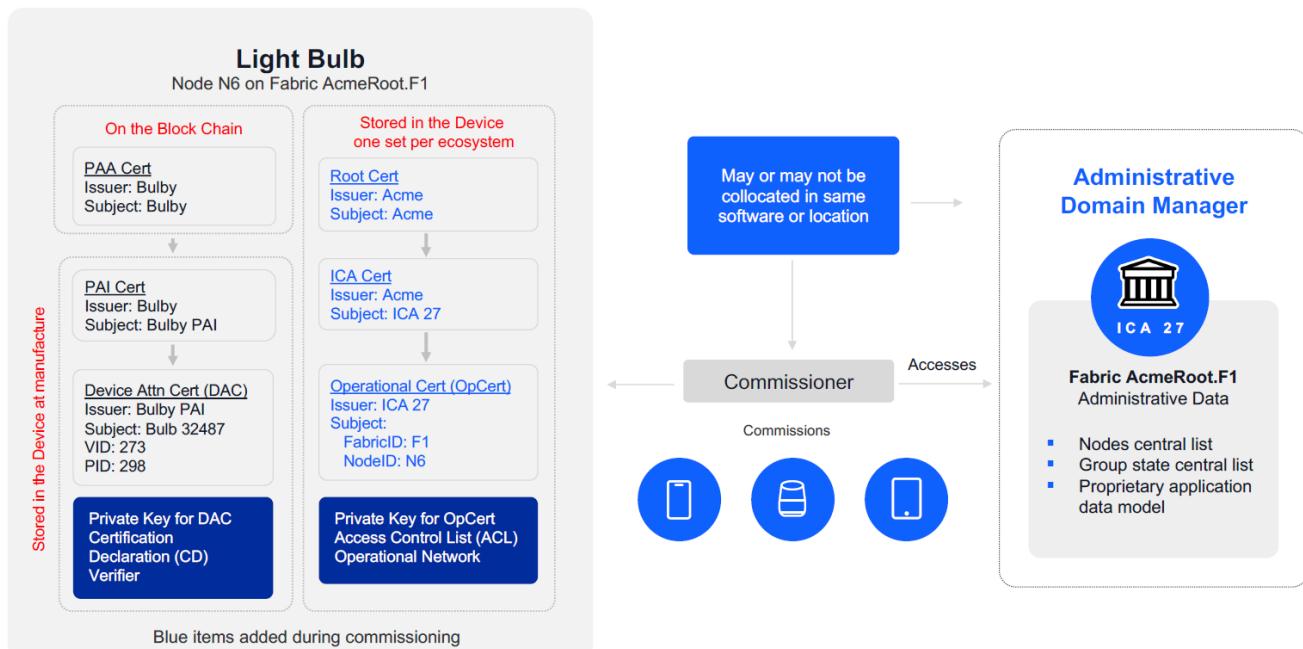
The Node Operational Credentials include the following items:

- Node Operational Key Pair
- Node Operational Certificate (NOC)
- Intermediate Certificate Authority (ICA) Certificate (optional)
- Trusted Root Certificate Authority (CA) Certificate(s)

Note: The Node Operational Credentials are distinct from the Device Attestation credentials.

Commissioning Process

The Commissioning process uses the DAC to establish that the Commissioner is talking to a certified Matter Device and then loads operational identities for each ecosystem that it joins.



4. Configuration

The Commissioner configures the Device by:

- Loading Node Operational Credentials per ecosystem:
 - Fabric ID
 - Node ID
 - Trusted Root Certificate
 - ICA Cert
 - Operational Cert
 - Node Operational Key Pair
 - Access Control List (ACL)
 - Operational Network
 - Time (optional)
- Establishing communication with other Nodes using CASE

This completes all the commissioning steps and now on the Matter Network.

Matter Security Requirements

Matter Security as Specified by CSA

Manufacturing: Matter Devices must be injected with a unique DAC certificate/private key, Onboarding Payload (QR code delivered), Certification Declaration (CD), and other static/dynamic data during manufacturing. (SHALL)

Commissioning: DAC with VID/PID must be checked against the DCL and CD verified to ensure only authentic and certified Matter Devices are commissioned. (SHALL)

Device Communication: Communication between Matter Devices must be secured and encrypted using cryptographic keys and PBKDF. (SHALL)

Software Updates: Devices must support OTA firmware updates to allow vulnerabilities to be patched. (SHALL)

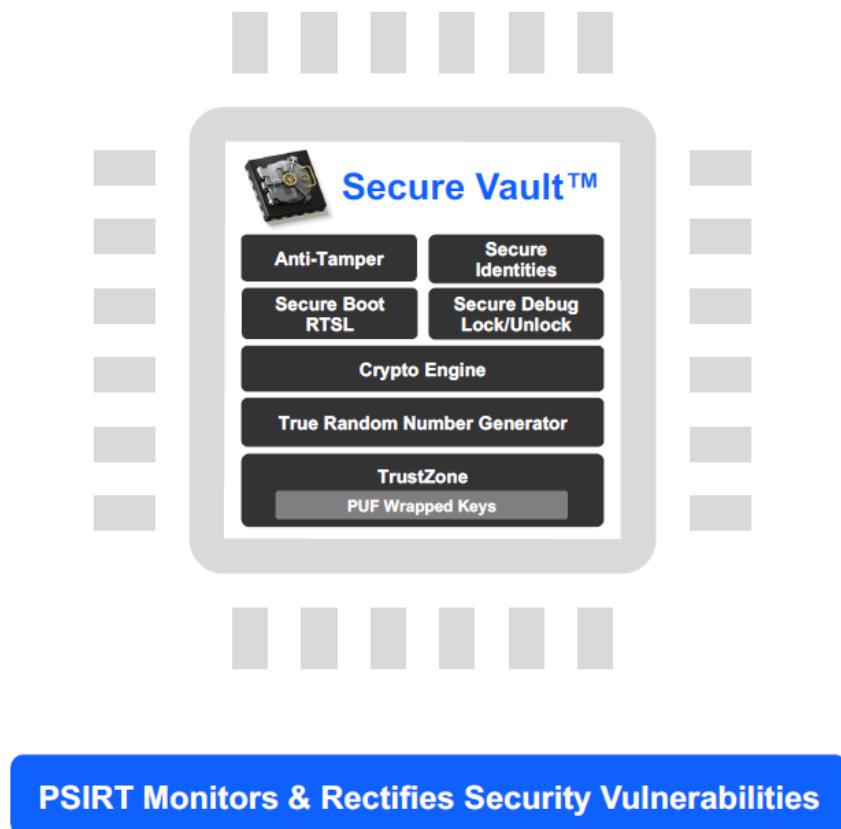
Other Security Specifications

- Authentication and encryption keys must be generated by a "Deterministic Random Bit Generator" seeded by NIST 800-90B TRNG. (SHALL)
- Debug interfaces and access to secure boot trust anchors should be disabled to only allow authorized access (fusing). (SHOULD)

- DACs and operational private key confidentiality should be protected from remote attacks. (SHOULD)
- Vendors should have a public policy and mechanism to identify and rectify security vulnerabilities in a timely manner. (SHOULD)
- The software should be encrypted *at rest* to prevent unauthorized access to core IP. (MAY)
- Some devices should be protected against *physical* attacks to prevent tampering, side-channel, or debug glitching attacks. (MAY)

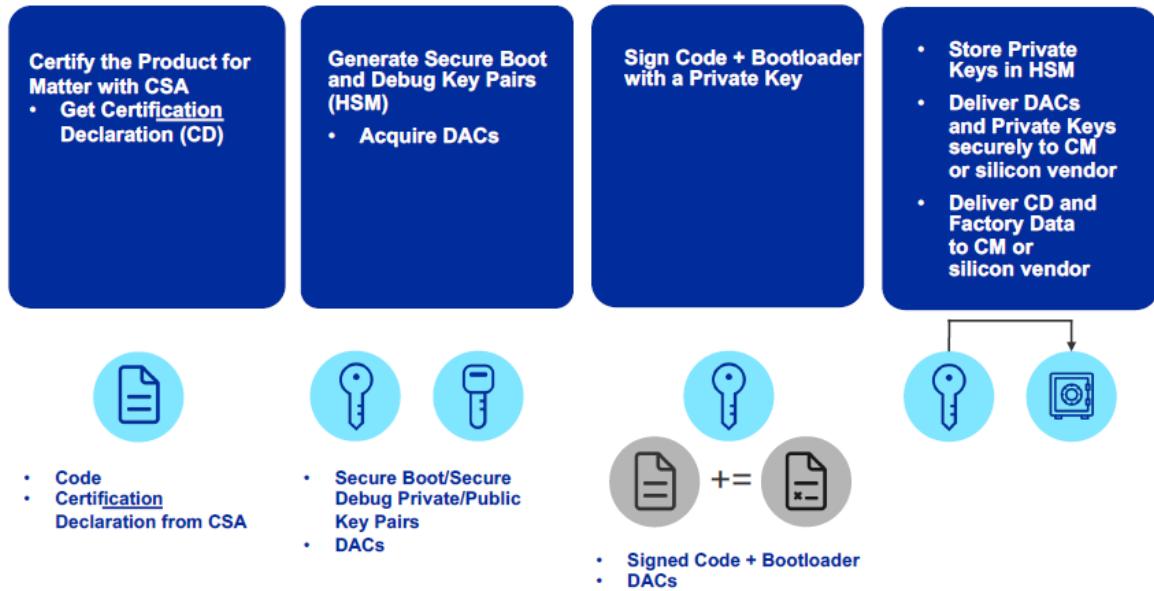
Matter Compliant Security Solution

- Secure Vault Mid or High supports all Matter security functionalities now (Shall) and future (Should, May)
- Uncrackable keys are generated by the True Random Number Generator (TRNG) For DAC, secure boot, secure debug, OTA, software image and communication encryption
- The Crypto Engine assists with special algorithms like SPAKE2+ and CASE with side channel protection
- Secure key storage at PSA/SEIP Level 2 (Mid) and Level 3 (High); Private keys are stored with a TEE/TZ (SV Mid) or PUF Wrapped (SV High)
- Secure Matter Identities (DACs) securely programmed at our factory
- Secure Boot with RTSL ensures code running on the device is trusted.
- Secure OTA firmware updates in conjunction with Secure
- Boot prevents the installation of malicious software and allows for vulnerability patching
- Glitch Mitigated Secure Debug Lock/Unlock only allow authorized access with security tokens that can be revoked
- Anti-Tamper protects from physical attacks (SV High)

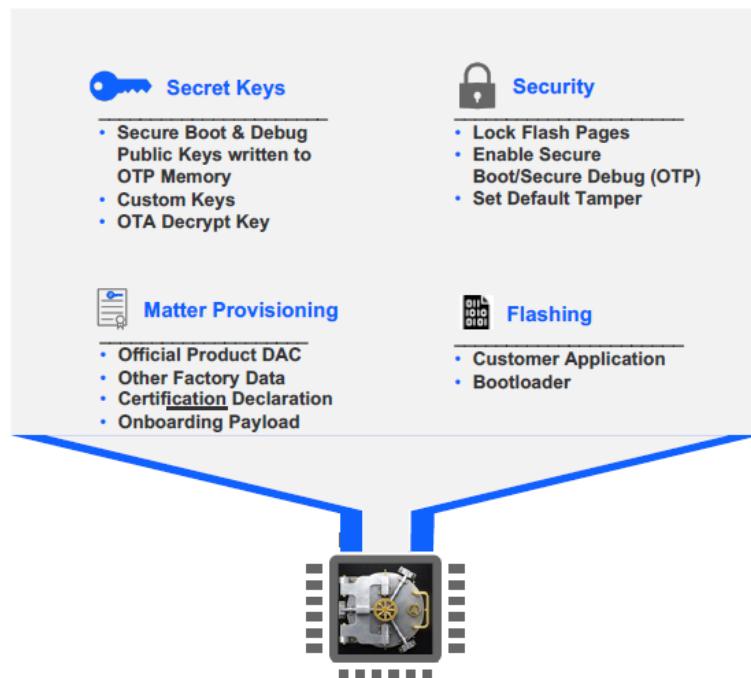


Matter Secure Manufacturing

The following diagram describes the high-level process of what needs to happen before secure programming.



The following diagram describes the high-level steps as part of secure programming.



Introduction

Matter Developer's Guide

The Matter Developer's Guide provides detailed background and instructions for Matter developers working in either the Thread or Wi-Fi models. The Developer's Guide contains a deeper dive into development using examples that are similar to those in the [Quick-Start Guides](#) as well as other topics of interest.

- [Prerequisites](#) provides a more extensive list of hardware options as well as additional detail on software prerequisites.
- The [Matter over Thread Example](#) and [Matter over Wi-Fi Example](#) provide much more detail than the quick-start guides.
- The [Matter Ecosystems](#) displays information on various IoT ecosystems and how Silicon Labs' Matter enabled applications integrate within them.
- The [Unify Matter Bridge](#) is an application that makes legacy devices, such as Z-Wave and Zigbee devices, accessible on a Matter fabric. It does so by acting as an IoT Service in a Unify Framework. This application is developed outside of Simplicity Studio.
- [Detailed Development Topics](#) cover a number of other topics of interest to developers, including commissioning, security, and over-the-air update.

Matter Prerequisites

Prerequisites

If you have already obtained the hardware and software for the [Matter over Thread and Matter over Wi-Fi Quick-Start guides](#), you will already have most of what you need. These pages offer more hardware alternatives and provide additional Detail.

These pages explain the **hardware** and **software** prerequisites for working with the Silicon Labs Matter products.

The [artifacts page](#) provides links to pre-built software image "artifacts" that can be used to set up the Matter Demo for the Thread and Wi-Fi use cases.

Hardware Requirements

Matter Hardware Requirements

To run Matter over Thread or over Wi-Fi requires some Silicon Labs hardware in order to run demos and do development. Following are the hardware requirements for both Thread and Wi-Fi use cases broken down by platform and transport protocol.

The following sections describe the hardware that may be used for Matter+OpenThread (Matter Hub and Accessory Device) and for Matter+Wi-Fi (Accessory Device). The EFRMG24 is the preferred starting point for Matter MCUs (including the Matter Hub RCP and both Accessory Devices). It provides Secure Vault and can use the internal flash of the device to store an upgrade image. The EFR32MG24 is recommended for running the [Matter over Thread and Matter over Wi-Fi Quick-Start guides](#).

Matter Over Thread "Matter Hub" Requirements

If you are running Matter over Thread and do not have a platform on which to run the Open Thread Border Router and chip-tool, Silicon Labs recommends that you run them on a Raspberry Pi. To do so you will need:

- **Raspberry Pi**
 - Raspberry Pi 4 with an SD card with storage >= 64 GB

The Raspberry Pi 4 is used to run the Open Thread Border Router and the chip-tool. In this documentation the combination of this software on the Raspberry Pi is also called the 'Matter Hub' A software image for the Raspberry Pi is provided on the [Matter Artifacts page](#).

- **Radio Co-Processor (RCP)**

The Matter Hub needs a 15.4 Radio Co-Processor (RCP) in order to create and interact with the Thread network. The RCP can be any Silicon Labs development board that is capable of running the OpenThread RCP firmware. The RCP radio board is connected to the Raspberry Pi via USB.

Over 60 Silicon Labs boards support running the RCP firmware. To build an image for a board which is not listed here, download and build your image in Simplicity Studio. Pre-built OpenThread RCP firmware images are provided for the following boards on the [Matter Artifacts page](#):

Note: The EFR32MG24 is the preferred starting point for Matter MCUs. It provides Secure Vault and can use the internal flash of the device to store an upgrade image.

- **MG24 boards:**
 - BRD4186C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@10dBm
 - [XG24-RB4186C](#)
 - BRD4187C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@20dBm
 - [XG24-RB4187C](#)
- **MG12 boards:**
 - BRD4161A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4161A](#)
 - BRD4162A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@10dBm
 - [SLWRB4162A](#)
 - BRD4163A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4163A](#)
 - BRD4164A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4164A](#)

Matter Over Thread Accessory Device Requirements

The Matter Accessory Device (MAD) is the actual device that the Matter application firmware (such as the Matter Light or Matter Switch) runs on. Several different platforms for the Matter Accessory Device are supported. Pre-built binary images for the Matter accessory devices are provided on the [Matter Artifacts page](#). Silicon Labs supports development of Matter Accessory Devices for Matter over Thread on the following platforms:

Note: The EFR32MG24 is the preferred starting point for Matter MCUs. It provides Secure Vault and can use the internal flash of the device to store an upgrade image.

- **MG24 boards:**

- BRD4186C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@10dBm
 - [XG24-RB4186C](#)
- BRD4187C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@20dBm
 - [XG24-RB4187C](#)
- BRD2703A / MG24 Explorer Kit

Note: This board has yet to be released to the public, but it is supported in the Silicon Labs build flow.
- BRD2601B / MG24 Explorer Kit
 - [XG24-DK2601B](#)
- BRD4319A / SLWSTK6006A / Wireless Starter Kit/ 2.4GHz@20dBm

Note: This board has yet to be released to the public, but it is supported in the Silicon Labs build flow.
- BRD4316A / SLWSTK6006A / Wireless Start Kit / 2.4GHz@10dBm
 - [XGM240-RB4316A](#)
- BRD4317A / SLWSTK6006A / Wireless Starter Kit/ 2.4GHz@20dBm
 - [XGM240-RB4317A](#)

- **MG12 boards:**

- [EFR32MG12 Development Kit](#)
 - BRD4161A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4161A](#)
 - BRD4162A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@10dBm
 - [SLWRB4162A](#)
 - BRD4163A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4163A](#)
 - BRD4164A / SLWSTK6000B / Wireless Starter Kit / 2.4GHz@19dBm
 - [SLWRB4164A](#)
 - BRD4166A / SLTB004A / Thunderboard Sense 2 / 2.4GHz@10dBm
 - [Thunderboard Sense 2](#)
 - BRD4170A / SLWSTK6000B / Multiband Wireless Starter Kit / 2.4GHz@19dBm, 915MHz@19dBm
 - [SLWRB4170A](#)

Matter Over Wi-Fi Accessory Device Requirements

Matter Over Wi-Fi Accessory Device Requirements for NCP Mode

The Silicon Labs Matter over Wi-Fi NCP mode demo and development requires two boards: the Silicon Labs EFR32 Radio board to run the Matter code and either the RS9116, SiWx917 or WF200 to run the Wi-Fi protocol stack. Pre-built images for the EFR32 are provided on the [Matter Artifacts page](#). Pre-built images for SiWx917 or RS9116 connectivity firmware are also provided on the [Matter Artifacts page](#).

Note:

1. The EFR32MG24 is the preferred starting point for Matter MCUs. It provides Secure Vault and can use the internal flash of the device to store an upgrade image.
2. The WF200 connectivity firmware image is included in the EFR32MG24 images on the [Matter Artifacts page](#) for running with the WF200 in NCP mode. The Matter application downloads the connectivity firmware onto the WF200 on first-time startup.

The following boards are supported for the Matter over Wi-Fi demos and development:

- **MG24 boards:**

- BRD4186C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@10dBm
 - [XG24-RB4186C](#)

- MG24 with WSTK : [xG24-PK6009A](#)
- BRD4187C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@20dBm
 - [XG24-RB4187C](#)
 - MG24 with WSTK : [xG24-PK6010A](#)
- Wi-Fi NCP Dev Kits & boards
 - RS9116
 - SB-EVK1 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK1](#)
 - SB-EVK2 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK2](#)
 - DB-EVK1 / Dual Band Wi-Fi Development Kit / 2.4GHz & 5GHz
 - [RS9116X-DB-EVK1](#)
 - SiWx917 NCP
 - SiWx917 NCP Mode / Wi-Fi Expansion Board / 2.4GHz
 - BRD8045A (B0 Expansion v2.0)
 - WF200
 - WF200 / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8022A](#)
 - WF200 / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8023A](#)
 - Interconnect board (included in the Wi-Fi kits)
- Interconnect board (included in the Wi-Fi kits)
- SPI Cable (included in the RS9116 kit)
- Jumper Cables (included in the RS9116 kit)

Matter over Wi-Fi Accessory Device Requirements for SoC Mode

The Silicon Labs Matter over Wi-Fi demo and development for SoC mode requires the SiWx917 SoC board that supports Matter over Wi-Fi in a single-chip package - the integrated MCU is dedicated for peripheral and application-related processing (Matter), while the ThreadArch® runs the wireless and networking protocol stacks.

Pre-built images for the SiWx917 connectivity firmware are available as per the instructions on the [Matter Artifacts page](#). The following boards are supported for the Matter over Wi-Fi demos and development:

- **Wi-Fi SoC boards:**
 - SiWx917 / BRD4002A / Wireless Starter Kit
 - SiWx917 SoC Mode
 - SiWx917 SoC / Common Flash Radio Board / 2.4GHz
 - BRD4338A - B0 common flash v2.0
- Note:** Refer to [SiWx917 SoC](#) for more details.

Additional Matter Over Wi-Fi Hardware Requirements

In addition to your Matter over Wi-Fi Accessory Device, you will need the following for both running the demo and for development:

- Windows/Linux/MacOS computer with a USB port
- USB cable for connecting WSTK Board to Computer
- Raspberry Pi with a >32 GB SD Card
- Access point with Internet access
- microSD card (>=32GB) (if using Raspberry Pi)
- **[Optional]** Android Mobile phone (If using the chip-tool on Android)
- Interconnect board (included in the RS9116 kit)
- SPI Cable (included in the RS9116 kit)
- Jumper Cables (included in the RS9116 kit)

Software Requirements

Matter Software Requirements

This page provides information on the required softwares tools, packages and firmware for developing Silicon Labs Matter over Thread and Wi-Fi devices.

Software Tools Required

Below are the software tools both optional and required for developing Matter over Thread and Matter over Wi-Fi applications in both NCP and SoC mode:

1. [Silicon Labs Simplicity Studio](#)

Simplicity Studio is the main IDE and development platform provided by Silicon Labs.

2. (Optional) [Ozone - The J-Link Debugger for Windows](#)

Ozone is a full-featured graphical debugger for embedded applications. With Ozone it is possible to debug any embedded application on C/C++ source and assembly level.

3. (Optional) [Simplicity Commander](#)

Simplicity Commander is a utility that provides GUI and command line access to the debug features of an EFM32 device. It allows you to flash firmware, update the kit firmware, and lock, or unlock debug access.

4. (Optional) [Tera Term](#)

Tera Term is the terminal emulator for Microsoft Windows that supports serial port, telnet and SSH connections.

5. (Optional) SSH Client ([PuTTY](#), Terminal, or similar):

SSH client is used to communicate with the Raspberry Pi over a secure shell.

6. [Raspberry Pi Disk Imager](#)

Raspberry Pi Disk Imager is used to flash the SD Card that contains the operating system for the Raspberry Pi.

Software Packages Required for Wi-Fi EFR32 NCP Devices

1. [GeckoSDK package](#), which can be installed as part of the Simplicity Studio tool installation.

2. [WiseConnect SDK v2.x for RS9116 NCP](#), which can be installed as part of the Simplicity Studio tool installation.

3. [Wisecconnect SDK v3.x for SiWx917 NCP](#), which can be installed as part of the Simplicity Studio tool installation.

4. [Firmware for RS9116 NCP](#)

5. [Firmware for SiWx917 NCP](#)

Software Packages Required for Wi-Fi SiWx917 SoC Devices

1. [GeckoSDK package](#), which can be installed as part of the Simplicity Studio tool installation.

2. [WiSeConnect SDK v3.x](#), which can be installed as part of the Simplicity Studio tool installation.

3. [Firmware for SoC](#)

Artifacts

Matter Software Artifacts

This page provides links to pre-built software image "artifacts" that can be used to set up the Matter Demo for the Thread and Wi-Fi use cases.

Images for the items listed below are available under the "Assets" section at the bottom of this page:

https://github.com/SiliconLabs/matter_extension/releases/tag/v2.2.0

Matter Hub Raspberry Pi Image

The Matter Hub image is intended to be flashed onto an SD card for a Raspberry Pi. The Matter Hub Image provides both an Open Thread Border Router and the Matter chip-tool. Note the image is ~10GB in size so depending on your internet connection this download may take some time. Start the Matter Hub Raspberry Pi image download here:

https://www.silabs.com/documents/public/software/SilabsMatterPi_2.2.0-1.2-extension.zip

Radio Co-Processor (RCP) Images

The Radio Co-Processor firmware is used to turn an EFR into an RCP that can be used with a Raspberry Pi to allow the Raspberry Pi's Open Thread Border Router to access the Thread network. Radio Co-Processor (RCP) images are available in the Assets section of this page:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/ot-rcp-binaries-2.2.0-1.2.zip

Matter Accessory Device Images

The Matter Accessory Device Images are used to turn an EFR into a Matter device. These are pre-built binary images for the Matter Demo. Matter Accessory Device Images are located in the Assets section of this page:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/matter-accessory-device-images_2.2.0-1.2.zip

Matter Bootloader Binaries

If you are using the OTA functionality and especially if you are using an EFR32MG2x device, you will need to flash a bootloader binary on your device along with the application image. Bootloader binaries for all of the Matter supported devices are available here:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/bootloader_binaries_matter_extension_v2.2.0-1.2.zip

RS9116 Firmware

The RS9116 firmware (rs9116_firmware_files_with_rev.zip) is used to update the RS9116 which can be found in the Assets section of this page:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/rs9116_firmware_files_with_rev_2.2.0-1.2.zip

Note: RS9116 chip/module needs to be flashed with proper firmware as mentioned below:

- RS9116.x.x.x.x.rps - This firmware image is valid for RS9116 1.5 revision chip/module

- RS9116.x.x.x.x.rps - This firmware image is valid for RS9116 1.4/1.3 revision chip/module

SiWx917 Firmware for SiWx917 NCP

The SiWx917 firmware (SiWx917NCP_firmware_files.zip) is used to update the SiWx917 NCP which can be found in the Assets section of this page:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/SiWx917NCP_firmware_files_2.2.0-1.2.zip

Note: SiWx917 NCP board need to be flashed with proper firmware as mentioned below:

- SiWG917-B.2.x.X.X.X.rps - This firmware image is valid for BRD8045A (B0 Expansion v2.0) board

SiWx917 Firmware for SiWx917 SoC

The SiWx917 firmware (SiWx917SOC_firmware_files.zip) along with WiSeConnect 3 SDK is used to update the SiWx917 SoC which can be found in the Assets section of this page:

https://github.com/SiliconLabs/matter_extension/releases/download/v2.2.0/SiWx917SOC_firmware_files_2.2.0-1.2.zip

Note: SiWx917 SoC boards need to be flashed with proper firmware as mentioned below:

- SiWG917-B.2.x.X.X.X.rps - This firmware image is valid for BRD4338A(B0 common flash v2.0) board

SiWx917 SoC Configuration Files For JLink RTT Logging

In order to check device logs for the Matter Application on the SiWx917 SoC, the **JLink RTT** must be configured for the SiWx917 SoC device by following the instructions on the [JLink RTT SOC Support](#) for SiWx917 SoC.

The [JLinkDevices.xml](#) and [.elf](#) files referenced in the instructions may be found in the Assets section of this page.

Note:- For EFR32MG2x devices, JLink RTT Logging support is already enabled.

Matter Over Thread Example

Silicon Labs Matter Over Thread Example

These pages describe in detail the steps for running an example lighting application for Matter over Thread. If you are new to Matter and Thread you may benefit from reviewing the [Matter Over Thread Quick Start Guide](#), which provides a step by step tutorial on running the Matter over Thread lighting demo in Simplicity Studio.

At a high level, these pages walk through starting a Thread network, commissioning a new device to the Thread network using Bluetooth LE, and finally sending a basic OnOff command to the end device.

Step 0: Prerequisites

Before beginning your Silicon Labs Matter over Thread project be sure you have satisfied all of the [Matter Hardware](#) and [Matter Software](#) Requirements.

Step 1: Setting up the Matter Hub (Raspberry Pi)

The Matter Hub consists of the Open Thread Border Router (OTBR) and the chip-tool running on a Raspberry Pi. 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.

The Matter Controller sends IPv6 packets to the OTBR, which converts the IPv6 packets into Thread packets. The Thread packets are then routed to the Silicon Labs end device.

See [How to use Matter Hub \(Raspberry Pi\) Image](#) for setup instructions.

Step 2: Flash the RCP

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 are available for the demo

Information on flashing and optionally building the RCP is located here: [How To Build and Flash the RCP](#)

Step 3: Build and Flash the MAD

The Matter Accessory Device (MAD) is the actual Matter device that will be commissioned onto the Matter network and controlled using the chip-tool. Prebuilt MAD images are available for the demo.

Information on flashing and optionally building the Matter Accessory device is located here: [How To Build and Flash the Matter Accessory Device](#)

Step 4: Commission and Control the MAD

Once the Matter Accessory device has been flashed onto your hardware you can commission it from the Matter Hub using the commands provided in the Raspberry Pi image:

Command	Usage
mattertool startThread	Starts the thread network on the OTBR
mattertool bleThread	Starts commissioning of a MAD using chip-tool

Command	Usage
mattertool on	Sends an on command to the MAD using chip-tool
mattertool off	Sends an off command to the MAD using chip-tool

Using the Matter Hub

Setting up the Matter Hub (Raspberry Pi)

The Matter Hub consists of the Open Thread Border Router (OTBR) and the chip-tool running on a Raspberry Pi. Silicon Labs has developed a Raspberry Pi image that can be downloaded and flashed onto an SD Card for the Raspberry Pi.

In short, the Matter Controller sends IPv6 packets to the OTBR, which converts the IPv6 packets into Thread packets. The Thread packets are then routed to the Silicon Labs end device.

How to use the Silicon Labs Matter Raspberry Pi Image (Matter Hub)

Note that if you have already downloaded the Raspberry Pi image and installed it, you may only need to update the image that you already have on your Raspberry Pi in order to use it with the current release. In this case you can follow the instructions on the [Matter Tool Page](#) to update your existing installation.

Step 1. Raspberry Pi Image Download

The provided Raspberry Pi image is used as a Matter Controller with the OTBR.

The image can be downloaded from the [Matter Artifacts page](#)

Please note that this image, even when zipped up, is quite large ~5GB so this download will take a while if you are on a slow connection. This image includes both the Ubuntu operating system as well as the OTBR and chip-tool, hence the size.

Step 2. Flashing your Raspberry Pi

[Raspberry Pi Disk Imager](#) can be used to flash the SD Card that contains the operating system for the Raspberry Pi. Under Operating System select 'Use Custom' and then select the .img file.

Alternatively, a tool like [balenaEtcher](#) can be used to flash the image to a micro SD card.

After flashing the SD card, insert it into the Raspberry Pi and reset the Raspberry Pi by unplugging it from the power source and plugging it back in. Then, wait at least 10 seconds for it to come up and start the SSH server.

Step 3. Finding your Raspberry Pi on the Network

The Raspberry Pi should be connected to a network - this could be Ethernet or a Wi-Fi network.

NOTE: If you cannot connect your Raspberry Pi to a network over Wi-Fi or Ethernet you do have the option to connect a monitor and keyboard to the Raspberry Pi and interact with it that way. In this case you do not need to connect your Raspberry Pi to a network as you can interface with it directly as you would with any computer running Ubuntu Linux.

The preference here is to use Ethernet, however, if you are using Wi-Fi for your connection to the Raspberry Pi, see [Connecting Raspberry Pi to Wi-Fi](#) for instructions on how to connect your Raspberry Pi to a Wi-Fi network.

Once you have connected your Raspberry Pi to the network, you need to connect to your Raspberry Pi over SSH. This requires the IP address of your Raspberry Pi. See [Finding Your Raspberry Pi](#) for more information on finding the IP address and connecting to the Raspberry Pi by SSH.

Raspberry Pi Login Credentials

- user: `ubuntu`
- password: `raspberrypi` OR `ubuntu` (0.3.0 and above)

Note: On later images of the Matter Hub the password has changed to the default "ubuntu". you will be asked to change your password the first time you log in. You may change it to whatever value you like.

Note: When you log into the Raspberry Pi for the first time over SSH you may receive a warning regarding a 'key fingerprint' - this is normal and expected. You can get past this by typing 'yes' at the prompt.

Step 4: Using the Matter Hub

The chip-tool, also referred to as the `mattertool`, is provided as a pre-built application inside the Raspberry Pi image.

Refer to the [chip-tool page](#) for information on using the Matter Hub with `mattertool` commands.

Setting up the RCP

How to Build and Flash the Radio Co-Processor (RCP)

The Radio Co-Processor is a 15.4 stack image flashed onto a Silicon Labs development kit or Thunderboard Sense 2. The 15.4 stack on the development kit communicates with the higher layers of the Thread stack running on the Raspberry Pi over a USB connection.

A complete list of supported hardware for the RCP is provided on the [Matter Hardware Requirements](#) page.

First, in order to flash the RCP, connect it to your laptop directly by USB.

Step 1: Get or Build the Image File to Flash the RCP

We have provided two ways to get the required image to flash the RCP. You can use one of the following options:

1. Use the pre-built 'ot-rcp' image file
2. Build the image file from the 'ot-efr32' repository, which is listed on the [Matter Repositories and Commit Hashes](#) page

Using a Pre-built Image File

RCP image files for all demo boards are accessible through the [Matter Artifacts Page](#). If you are using a pre-built image file, you can skip to [Step #2: Flash the RCP](#).

Building the Image File from the Repository

1. Clone the ot-efr32 repository

The 'ot-efr32' repo is located in Github here: <https://github.com/SiliconLabs/ot-efr32>.

You must have Git installed on your local machine. To clone the repo use the following command:

```
$ git clone https://github.com/SiliconLabs/ot-efr32.git
```

Once you have cloned the repo, enter the repo and sync all the submodules with the following command:

```
$ cd ot-efr32
```

```
$ git submodule update --init
```

After updating the submodules you can check out the correct branch or commit hash for the system. Check the current branch and commit hash used here: [Matter Branches and Commit Hashes](#)

```
$ git checkout <commit hash>
```

2. Build the RCP

Once you have checked out the correct hash, follow the instructions here: <https://github.com/SiliconLabs/ot-efr32/blob/main/src/README.md> to build the RCP image for your EFR platform.

This process will build several images for your board. The filename of the image to be flashed onto the board to create an RCP is 'ot-rcp.s37'.

The output of the build process puts all the image files in the following location: '(git)/ot-efr32/build/(efr32xgxx)'

Step 2: Flash the RCP

Once you get the RCP image, either by downloading a prebuilt image or building the image file from the repo, you can flash it onto your device. This is done directly from your laptop and not through the Raspberry Pi, so make sure that the device is connected directly over USB to your laptop. You can flash your RCP using Simplicity Studio or using standalone Simplicity Commander.

Once you have flashed the image, the device becomes the RCP. Disconnect it from your laptop and connect it via USB to the Raspberry Pi.

The Raspberry Pi's Open Thread Border Router can then use the RCP to communicate with the Thread network.

Creating an End Device

How to Build and Flash the Matter Accessory Device (MAD)

The Matter Accessory Device, such as the lighting-app, is the actual Matter device that you will commission onto the Matter network and control using the chip-tool.

Step 1: Get the Image File to Flash the MAD

We have provided two ways to get the required image to flash the MAD. You can use one of the following options:

1. Use the pre-built image file from either Simplicity Studio or Silicon Labs Matter GitHub
2. Build the image file from Simplicity Studio or out of the Silicon Labs Matter GitHub `matter` repository

- **Using the Pre-Built Image File**

Prebuilt image files are available both on GitHub and inside Simplicity Studio.

- **Simplicity Studio**

To find the demos within Simplicity Studio, even if you do not have a device connected:

1. Go to the Launcher within Simplicity Studio, you can see the Launcher tab in the upper right hand corner
2. Under "Get Started" choose "All Products"
3. Choose a board to start with such as "BRD4186C Rev A01" and click "Start", this will bring up the launcher window for that part
4. In the top navigation choose "Example Projects & Demos"
5. In the left hand navigation choose "Matter" to show all the Matter demos.
6. The demos are marked as "demo" and allow you to "Run" them, Projects can be "Created"
7. Choose the demo you wish to use and click "Run" to flash it onto your board

- **Silicon Labs GitHub**

If you are interested in using prebuilt image files from GitHub, all of the Matter Accessory Device image files are accessible through the [Matter Artifacts Page](#). If you are using a pre-built image file, you can skip forward to Step #2: Flashing the MAD.

If you are coming from Simplicity Studio, you may have already installed the demo image in Simplicity Studio in which case you can skip forward to the next step.

- **Building the Matter Image File**

There are two ways to build a Matter Accessory Device image file. You can build it using the Silicon Labs Matter GitHub Repo or you can build it using Simplicity Studio. The entire build process for Simplicity Studio is covered in the [Matter Over Thread Quick Start Guide](#)

- [Build Using the Matter GitHub Repo](#)
 - [Build Using Simplicity Studio](#)

Step 2: Flash the Matter Accessory Device

For more information on how to flash your Silabs development platform see the following instructions: [How to Flash a Silicon Labs Device](#)

Once your Matter Accessory Device has been flashed it should show a QR code on the LCD. If no QR Code is present it may be that you need to add a bootloader to your device. Bootloader images are provided on the [Matter Artifacts page](#).

Using the Chip-Tool

Using the Mattertool (chip-tool)

The following commands show how to start a new Thread network from the local OTBR, commission an EFR32 Matter End Device (Matter Accessory Device), and then send the on/off commands with the `mattertool` automated script. The `mattertool` script provides an interface into various chip-tool and otbr commands used to create and interact with a Matter network

Basic Mattertool Commands

Command	Usage
<code>mattertool startThread</code>	Starts the thread network on the OTBR
<code>mattertool bleThread</code>	Starts commissioning of a Matter Accessory Device using the chip-tool
<code>mattertool on</code>	Sends the <i>on</i> command to the Matter Accessory Device using the chip-tool
<code>mattertool off</code>	Sends the <i>off</i> command to the Matter Accessory Device using the chip-tool

You can also use the full chip-tool command set (still using mattertool)

```
$ mattertool levelcontrol read current-level 106 1
```

Advanced Information on the Matter Hub

Image tree

- home
 - ubuntu (you are here)
 - connectedhomeip (git repo: <https://github.com/project-chip/connectedhomeip.git>)
 - ot-br-posix (git repo: <https://github.com/openthread/ot-br-posix.git>)
 - scripts (in-house scripts)
 - configurations.sh
 - matterTool.sh
 - setupOTBR.sh

Open Thread Border Router (OTBR)

For information on what commits to use for the OTBR and RCP, consult the [Matter Repositories and Commit Hashes page](#)

The pre-installed OTBR is configured for the infrastructure interface eth0.

Bash script to modify, reinstall or update the OTBR:

```
$ otbrsetup
```

This bash script centralizes and simplifies the local OTBR installation.

Available commands:

Command	Description
-h, --help	Prints help options
-if, --interface <eth0 wlan0>	Select infrastructure interface. Default eth0
-i, --install	Bootstrap, set up and install the OTBR. Usually for a new installation
-s, --setup	Runs the OTBR setup only, use this to change the configured infrastructure interface (use in combination with -if wlan0 for Wi-Fi)
-u, --update	Update the OTBR installation after the repo is updated

Usage:

Change infrastructure to wlan0: \$ otbrsetup -if wlan0 -s

Rerun full install for eth0 interface: \$ otbrsetup -i

Upgrading the OpenThread Border Router (OTBR)

Change OTBR commit reference/version

```
$ cd /home/ubuntu/ot-br-posix
$ git fetch
$ git checkout <SHA>
$ otbrsetup -u
```

Upgrading the Matter - Chip-tool

For more information on the commit hashes used for this demo please consult the following page: [Matter Repositories and Commit Hashes](#)

To change the chip-tool commit reference/version, follow these steps:

```
$ cd /home/ubuntu/connectedhomeip
$ git fetch
$ git checkout <SHA>
$ mattertool buildCT
```

The mattertool script centralizes and simplifies the use of chip-tool and starting a clean thread network.

Available commands:

Command	Description
help	Prints help options
startThread	Start a new thread network and store the operational thread dataset for the commissioning purpose (bleThread)
bleThread	For Matter Bluetooth LE thread commissioning with an EFR32 device
bleWifi	For Matter Bluetooth LE Wi-Fi commissioning with an EFR32 device

Command	Description
buildCT	Clean build of the chip-tool
cleanVars	Erase every Set variable used in the script. They will be set back to default or randomized value
off	Turn off the Light on the already-commissioned EFR32 device
on	Turn on the Light on the already-commissioned EFR32 device
toggle	Toggle the Light on the already-commissioned EFR32 device
parsePayload	Parse the given Payload (QrCode string)
rebuildCT	Rebuild the chip-tool
vars	Print the Variables in use by the script

Some options/arguments can be added to the command to update the values of the variables used by the script.

Available commands:

Command	Description
-h, --help	Prints help options
-n, --nodeId DIGIT	Specify the Nodeid you are trying to reach
-e, --endpoint DIGIT	Specify an endpoint for the desired cluster
-d, --dataset HEX_STRING	Thread Operation Dataset to be provisioned
-s, --ssid STRING	Wi-Fi AP SSID that the end devices need to connect to
-p, --password STRING	Wi-Fi AP password

These configurations are held until overwritten, cleared with cleanVars or when Raspberry Pi reboots.

Active variables used by mattertool:

Variable	Value
MATTER_ROOT	/home/ubuntu/connectedhomeip
CHIPTOOL_PATH	/home/ubuntu/connectedhomeip/out/standalone/chip-tool
NODE_ID	31354
THREAD_DATA_SET	<the_value_you_get>
PINCODE	20202021
DISCRIMINATOR	3840
SSID	<your_SSID>
lastNodeid	0

You can preset them with export X=Y before running the script or use some available options to change some of them.

In most cases, MATTER_ROOT, CHIPTOOL_PATH, PINCODE, and DISCRIMINATOR should remain at the default set value.

For commissioning commands (bleThread, bleWifi) NODE_ID will be randomized if it is the same as the last pairing

When the startThread command is used, THREAD_DATA_SET will be assigned with the right operation dataset for the created Thread Network.

Scripts Alias

The commands presented above are linked to scripts. You can edit `.bashrc` and rename the following alias to your liking.

```
$ alias mattertool='source $HOME/scripts/matterTool.sh'
```

```
$ alias otbrsetup='source $HOME/scripts/setupOTBR.sh'
```

Overview

Silicon Labs Matter Over Wi-Fi Example

This section walks through the steps to build the Matter 1.2 applications, such as lighting, light-switch, window covering, thermostat, door lock, and on/off plug on silicon labs Wi-Fi devices. A complete list of hardware supported for Wi-Fi is included on the [Hardware Requirements page](#).

Step 1: Matter Wi-Fi Prerequisites

Before running the Matter Wi-Fi demo or developing for Wi-Fi, make sure that you have all the required hardware and software for each use case.

1. [Matter Hardware Requirements](#)
2. [Matter Software Requirements](#)

Step 2: Building the chip-tool for Wi-Fi

To run the Matter Wi-Fi demo, you will need to run the chip-tool on one of two platforms, either Linux/Mac or Raspberry Pi. Refer to [Building the Chip-Tool](#).

Step 3: Building the Matter Accessory Device (MAD) for Wi-Fi

Refer to [Getting Started with Matter Development](#) for instructions.

Step 4: Running the Example Network

This contains instructions to run the Matter Wi-Fi applications using chip-tool [Running the Matter Demo on Matter Accessory Device](#) on a Linux Machine (either Laptop or Raspberry Pi). Follow this after successfully executing the above steps.

Additional Information

- [Enabling Features](#)
- [Matter Shell](#)
- [Matter OTA](#)

Getting Started

Getting Started with Matter over Wi-Fi

To get started with Matter over Wi-Fi, download the latest version of Simplicity Studio as described in [Software Installation](#) and select one of the getting started guides in this section:

- **System-on-chip (SoC) mode:** Both the application and connectivity stack runs on the SiWx91x™ chipset. Refer to [Getting Started with SoC Mode](#).
- **Network Co-Processor (NCP) mode:** The application runs on an external micro-controller unit (MCU) host and the connectivity stack runs on the Silicon Labs Wi-Fi Processor. Refer to [Getting Started with EFR32 in NCP Mode](#).

Setting up the Matter over Wi-Fi Development Environment

Refer to the [Release Notes](#) to know more about the latest releases from Silicon Labs.

To control the Matter Accessory Device, a controller is required which is termed as **chip-tool**. The chip-tool can be set up in two ways:

- **On the Raspberry Pi:** See [Setting up chip-tool on Raspberry Pi](#).
- **On Linux:** See [Setting up chip-tool on Linux](#).

Running the Matter over Wi-Fi Demo

To run the Matter Demo over Wi-Fi , refer to [Running the Matter Demo over Wi-Fi](#).

Software Installation

Matter Over Wi-Fi Development Software Requirements

This document provides information and procedures to install the required software, tools, and packages for Silicon Labs Matter over Wi-Fi Matter Accessory Device (MAD) development.

Required Software Tools

These are the generic software tools required for both NCP and SoC devices.

1. [Silicon Labs Simplicity Studio](#): Simplicity Studio is the main IDE and development platform provided by Silicon Labs.
2. [Ozone - The J-Link Debugger for Windows](#): Ozone is a full-featured graphical debugger for embedded applications. With Ozone, it is possible to debug any embedded application on C/C++ source and assembly level.
3. [Simplicity Commander](#): Simplicity Commander is a utility that provides GUI and command line access to the debug features of an EFM32 device. It allows you to flash firmware, update the kit firmware, and lock or unlock debug access.
4. [Tera Term](#): Tera Term is the terminal emulator for Microsoft Windows, which supports serial port, telnet, and SSH connections.
5. [PuTTY](#) (SSH Client, Terminal, or similar): SSH client is used to communicate with the Raspberry Pi over a secure shell.
6. [Raspberry Pi Disk Imager](#): Raspberry Pi Disk Imager is used to flash the SD Card that contains the operating system for the Raspberry Pi.

Install Software Packages

The following packages will be installed during the installation of Simplicity Studio:

Gecko SDK Extension

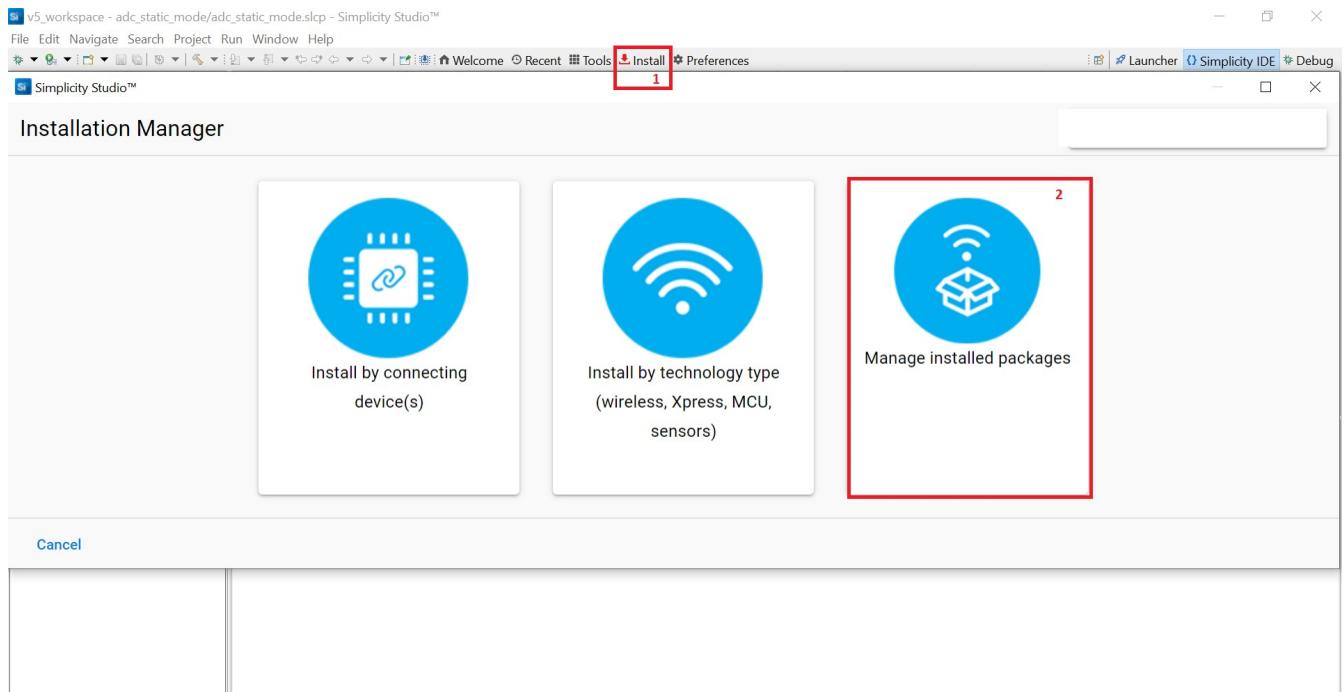
If you already selected the Gecko SDK extension while installing Studio, you may skip this section. You may install Gecko SDK by following these steps:

- Installation Manager (recommended)
- Manage SDKs Window

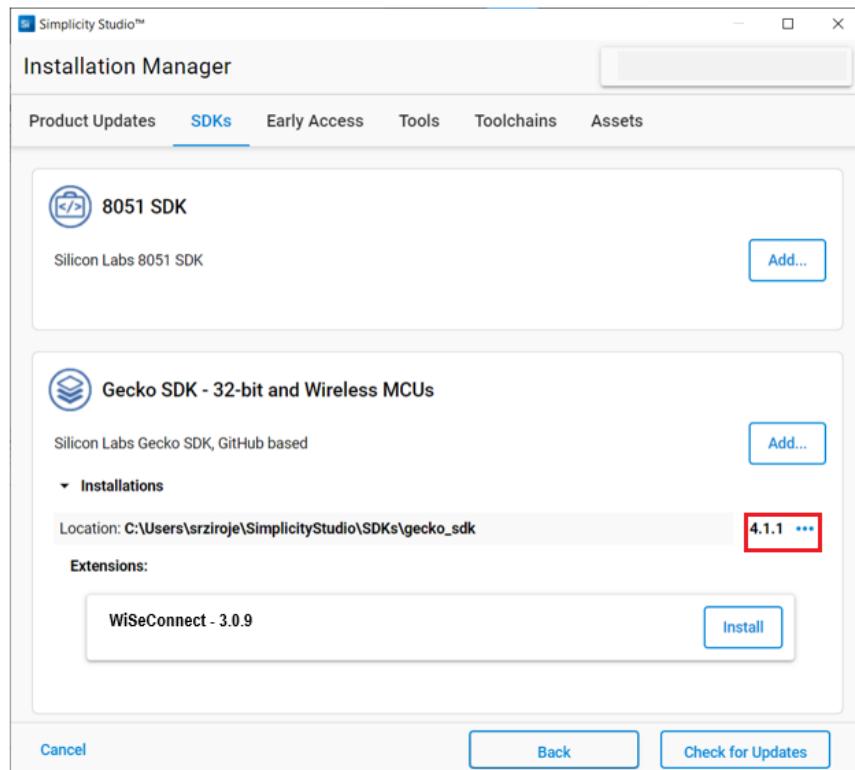
Note: Version numbers mentioned in the screenshot might be outdated. Install the latest packages available with the studio.

Install Gecko SDK Through the Installation Manager

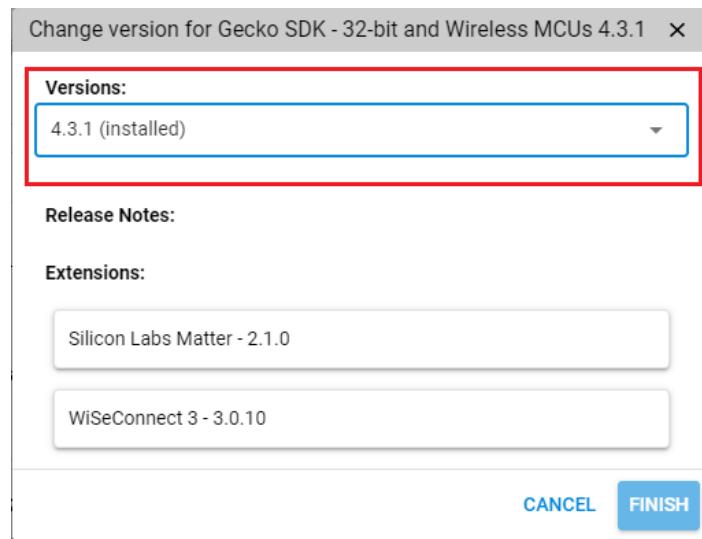
1. Log in to Simplicity Studio.
2. In the Simplicity Studio home page, select **Install > Manage installed packages**.



3. Select the **SDKs** tab.
4. Next to the Gecko SDK version, click the three dots button.



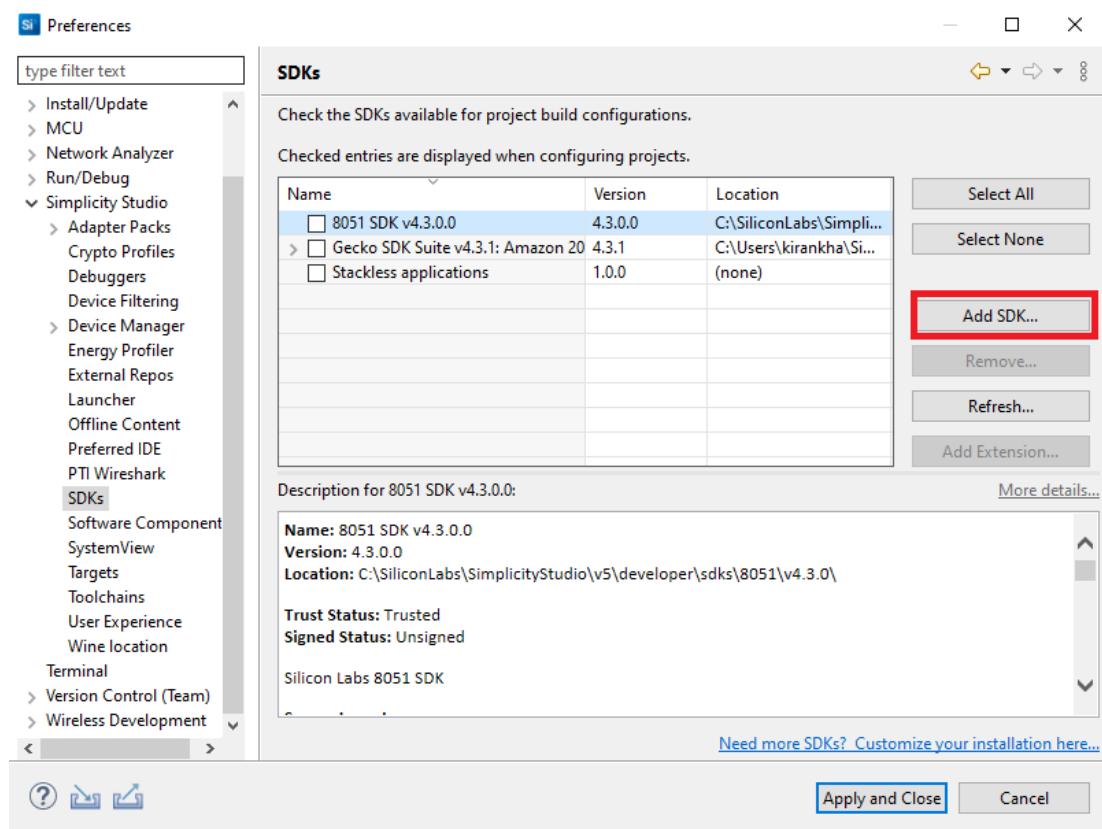
5. Select Change Version and select New Version from dropdown to install, and click Finish.



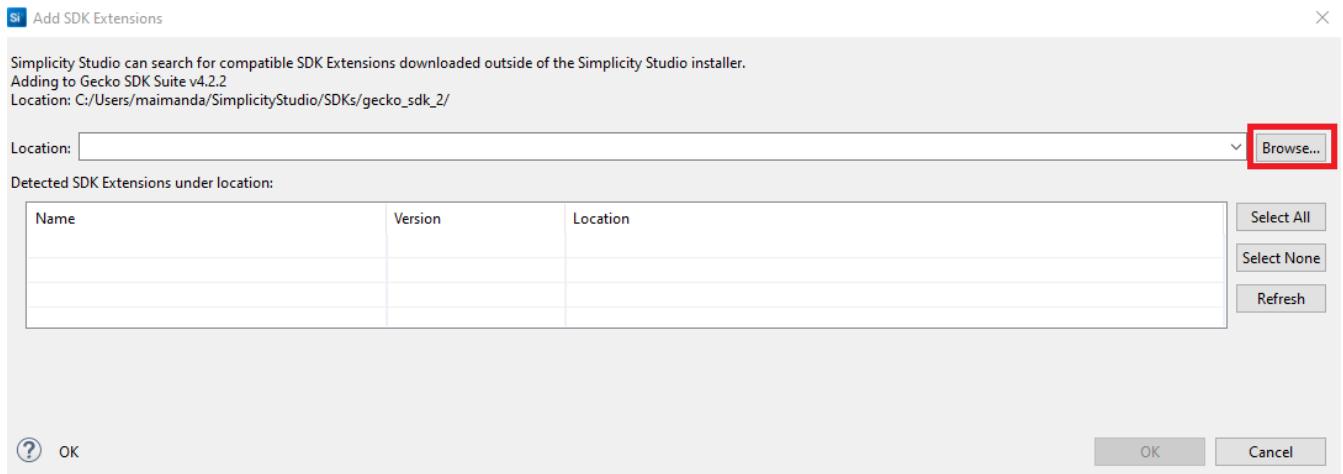
Install the Gecko SDK Through the "Manage SDK" Window

1. Download the Gecko SDK v4.x source code from the following URL after substituting 4.x.x with the desired release version:
https://github.com/SiliconLabs/gecko_sdk.git
- If you don't know the release version, go to the GitHub releases page and select the version to download.
2. Unzip the downloaded Gecko SDK-4.x.x.zip file.
3. Launch Simplicity Studio and log in.
4. In the **Debug Adapters** panel, select the radio board.
5. In the **General Information** section, click **Manage SDKs**.

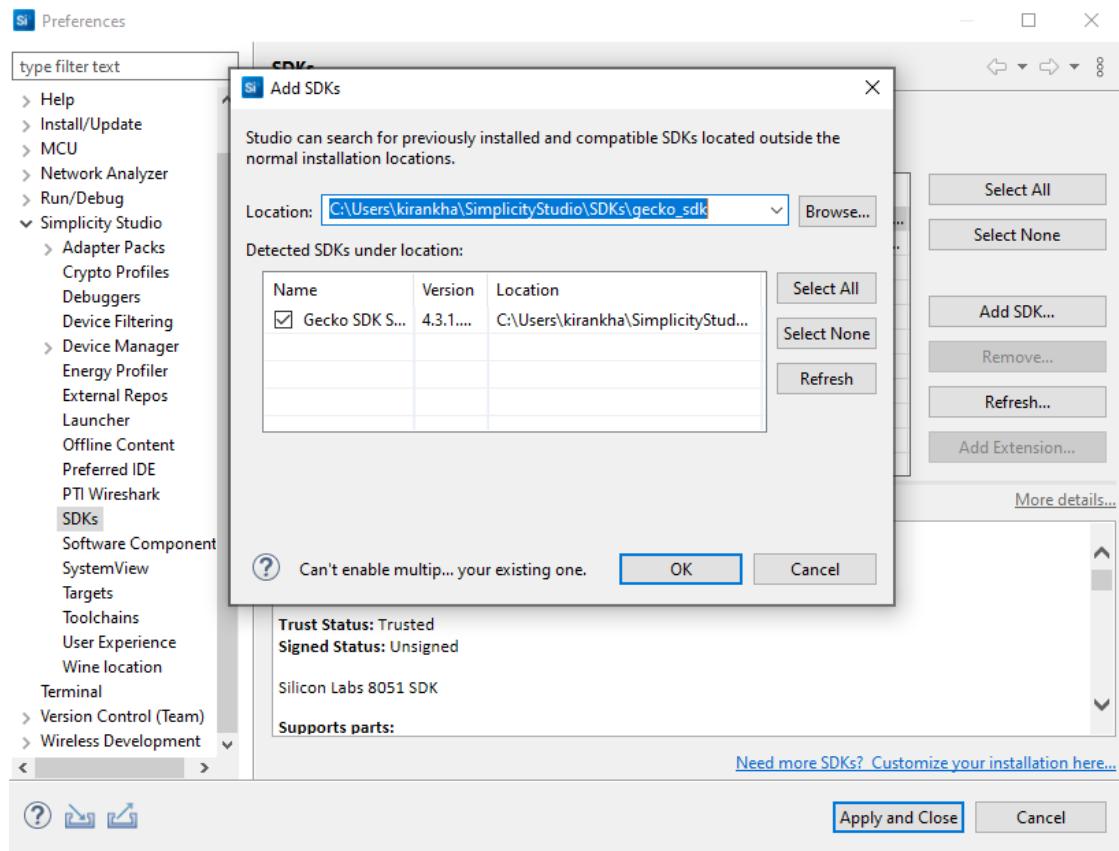
6. The Preferences window will be opened in the SDKs section.
7. Select **Add SDK**.



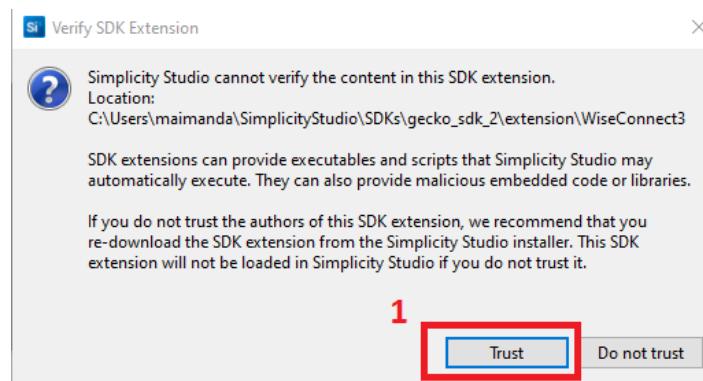
8. In the Add SDK Extensions window, click **Browse**.



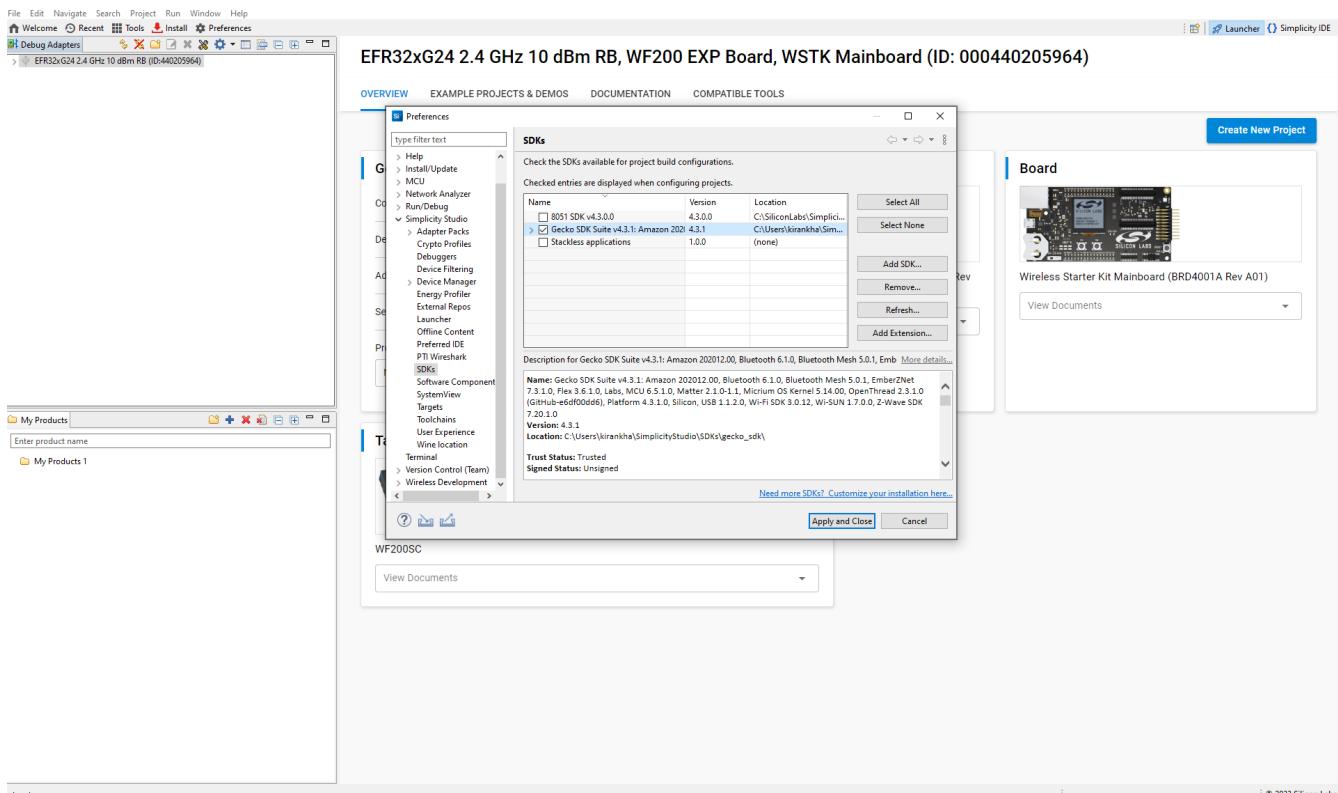
9. Locate and select the Gecko SDK sub-folder extracted in step 2 above, which contains the source code.
10. Studio will detect the Gecko SDK extension.
11. Select the detected extension and click **OK**.



12. If a Verify SDK Extension popup is displayed, click Trust.



13. The selected GSDK extension will be displayed.



14. Click **Apply** and **Close**.

Install WiSeConnect SDK v2.x or v3.x Extension

If you already selected the WiSeConnect extension while installing Simplicity Studio, you may skip this section.

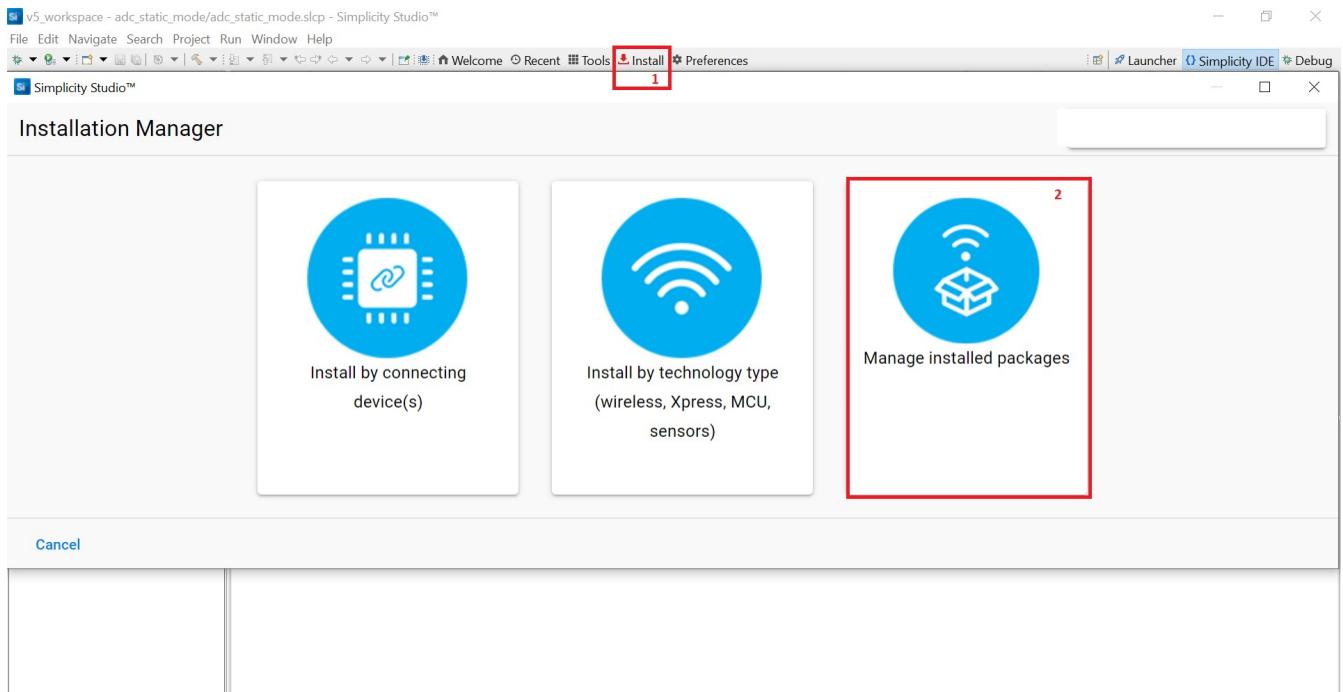
Before installing the WiSeConnect SDK v2.x or v3.x extension, if necessary, upgrade to a compatible GSDK version by following the steps above.

Install WiSeConnect SDK v2.x or v3.x through one of the following alternative paths:

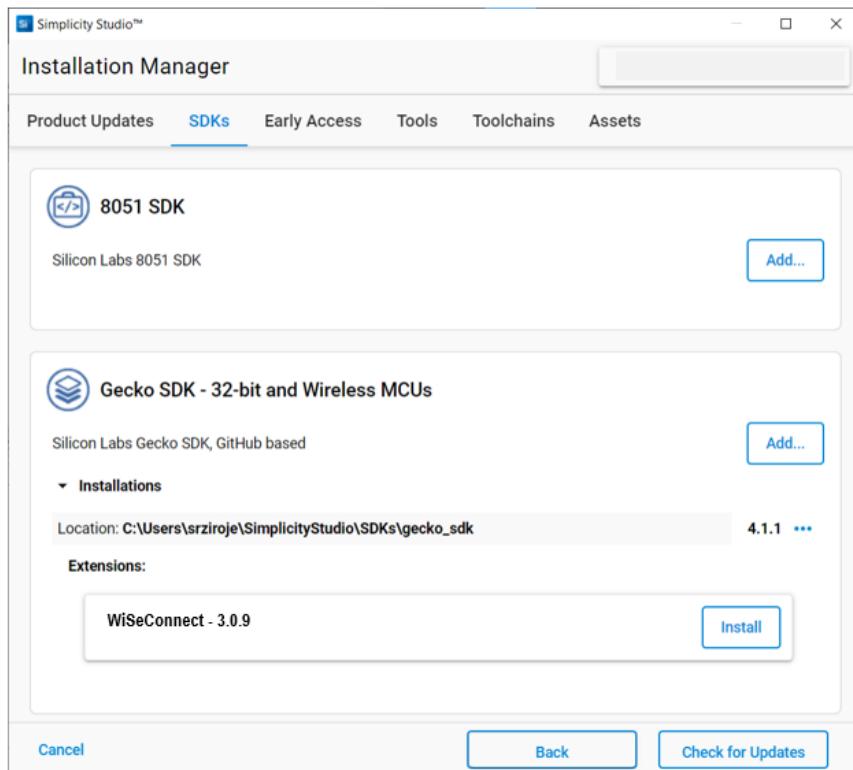
- Installation Manager (recommended)
- Manage SDKs Window

Install WiSeConnect SDK Through the Installation Manager

1. Log in to Simplicity Studio if not already done.
2. In the Simplicity Studio home page, select **Install > Manage installed packages**.



3. Select the **SDKs** tab.
4. Next to the WiSeConnect SDK v2.x or v3.x extension, click **Install**.

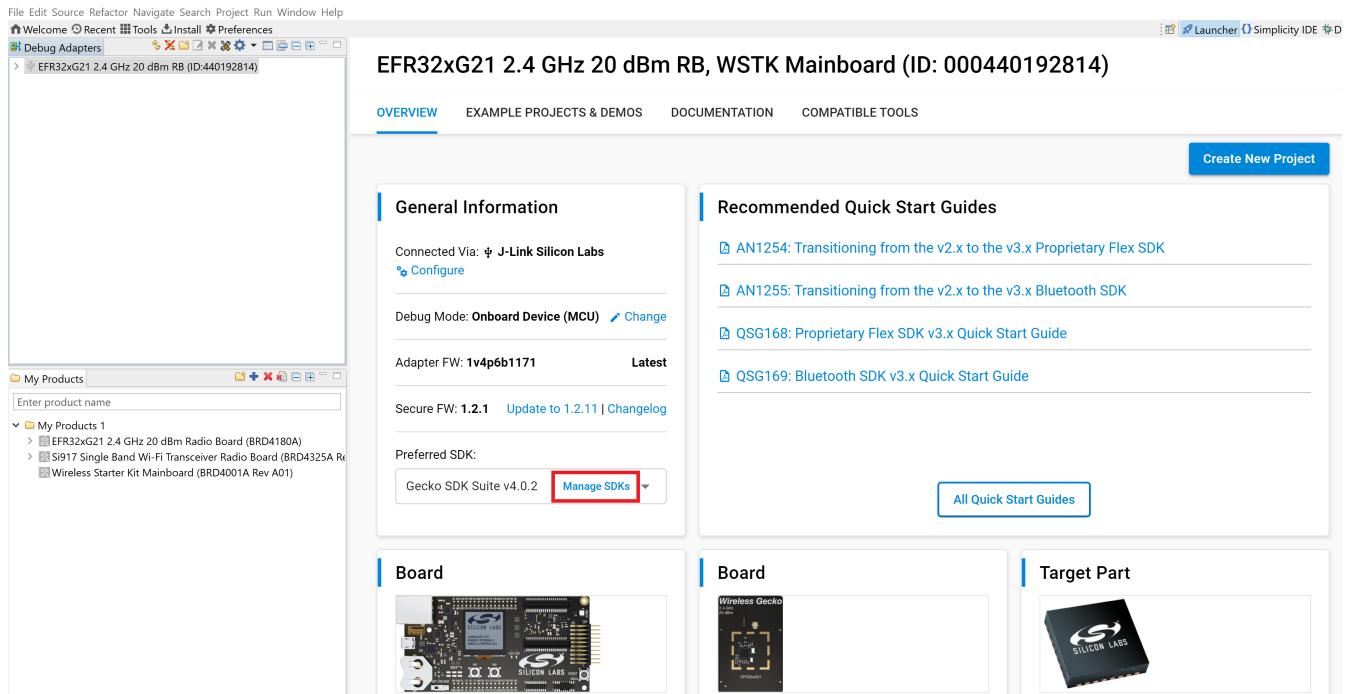


Install WiSeConnect SDK Through the Manage SDKs Window

1. Download the WiSeConnect v3.x source code from the following URL after substituting 3.x.x with the desired release version:
<https://github.com/SiliconLabs/wiseconnect/archive/refs/tags/v3.x.x.zip>

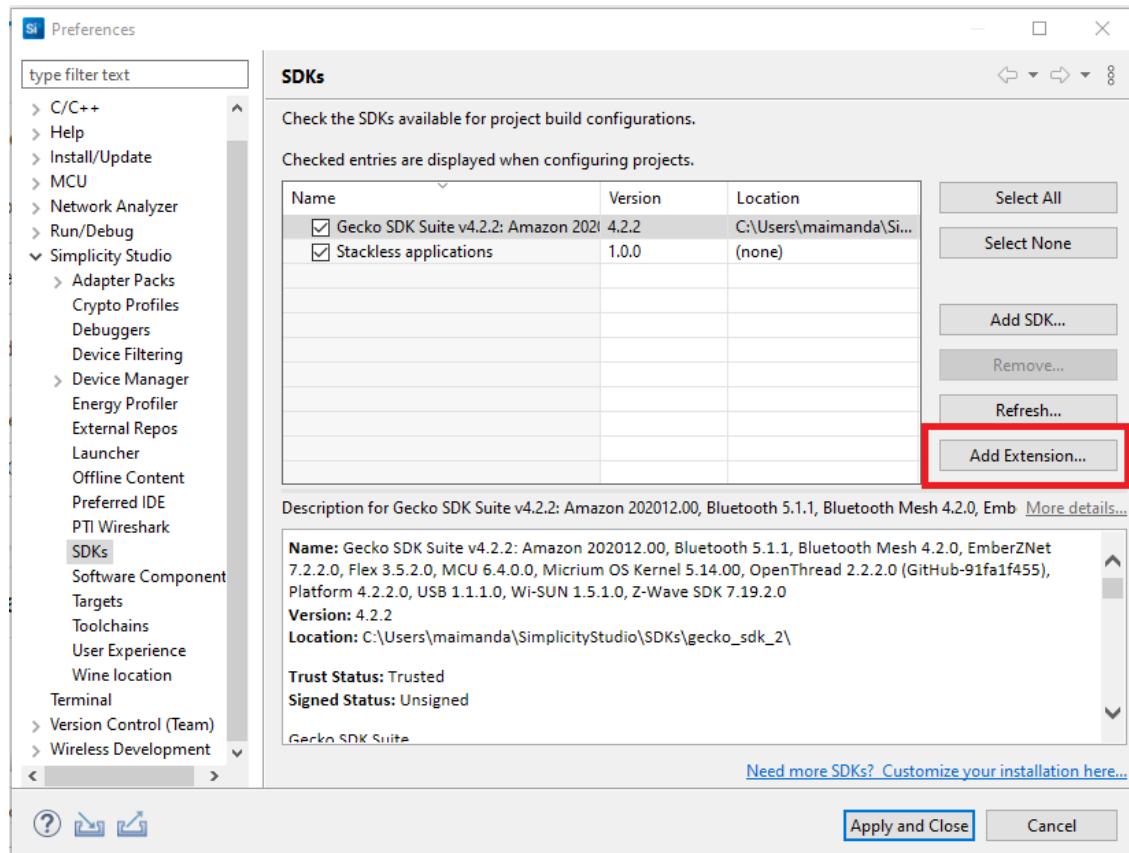
If you don't know the release version, go to the GitHub releases page and select the version to download.

2. Unzip the downloaded wiseconnect-3.x.x.zip file.
3. Launch Simplicity Studio and log in.
4. In the **Debug Adapters** pane, select your radio board.
5. In the **General Information** section, click **Manage SDKs**.

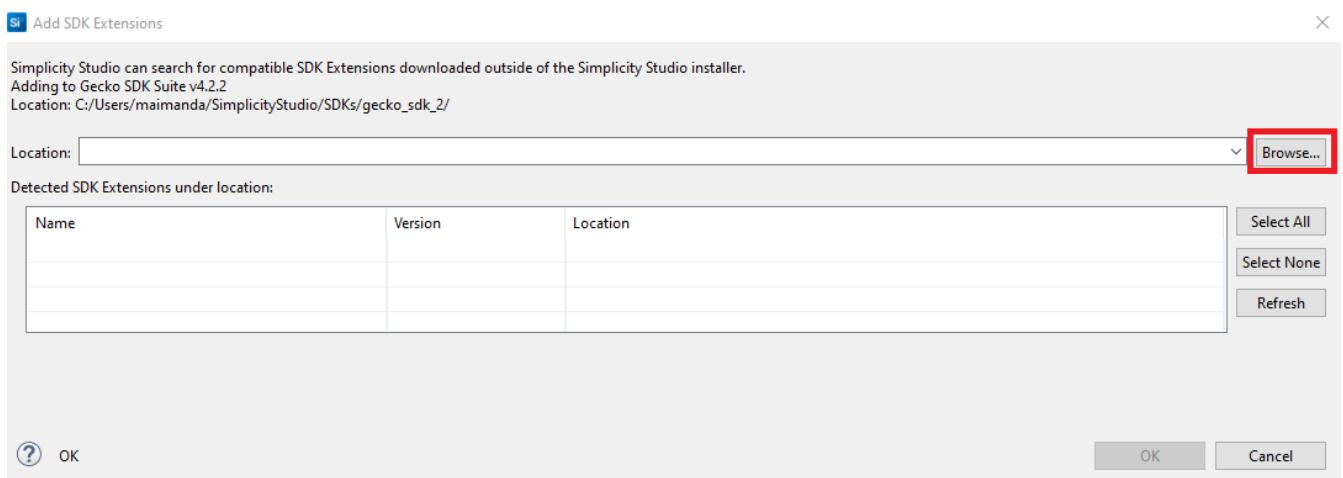


6. The **Preferences** window will be opened in the **SDKs** section.

7. Select Gecko SDK Suite vx.x.x and click **Add Extension**.



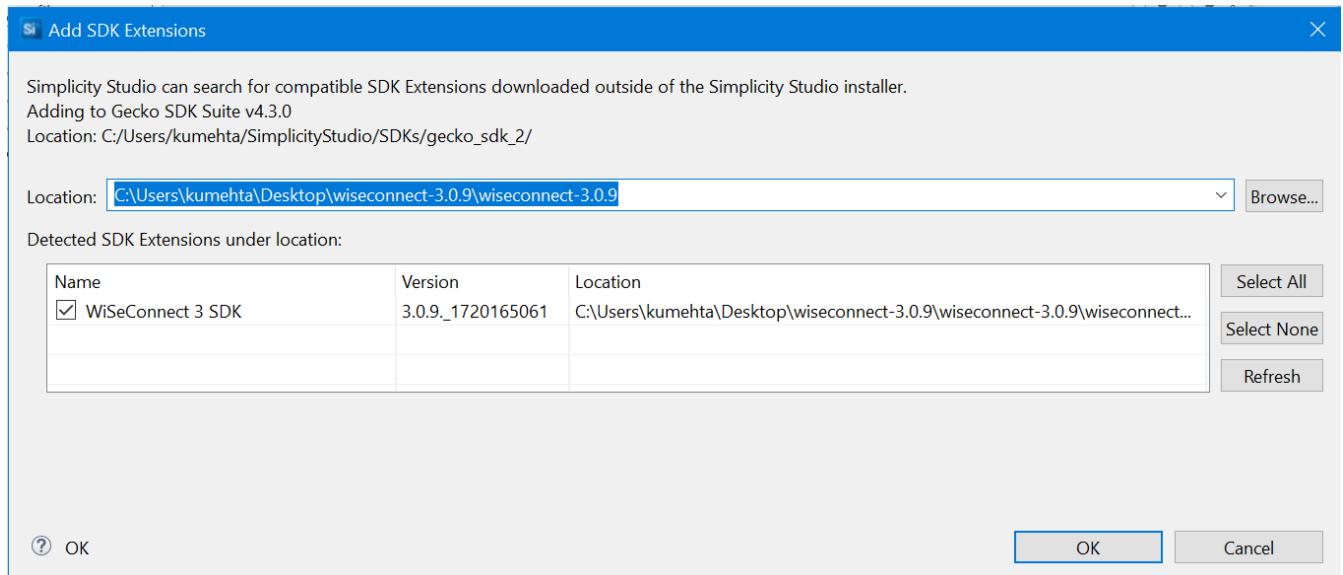
8. In the Add SDK Extensions window, click **Browse**.



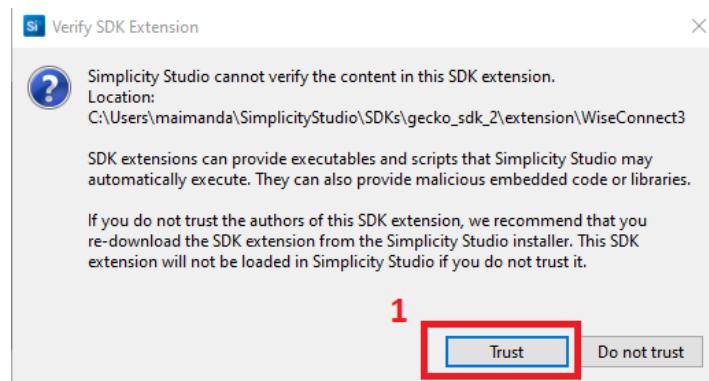
9. Locate and select the WiSeConnect SDK v2.x or v3.x sub-folder extracted in step 2 above, which contains the source code.

10. Studio will detect the WiSeConnect 3 SDK extension.

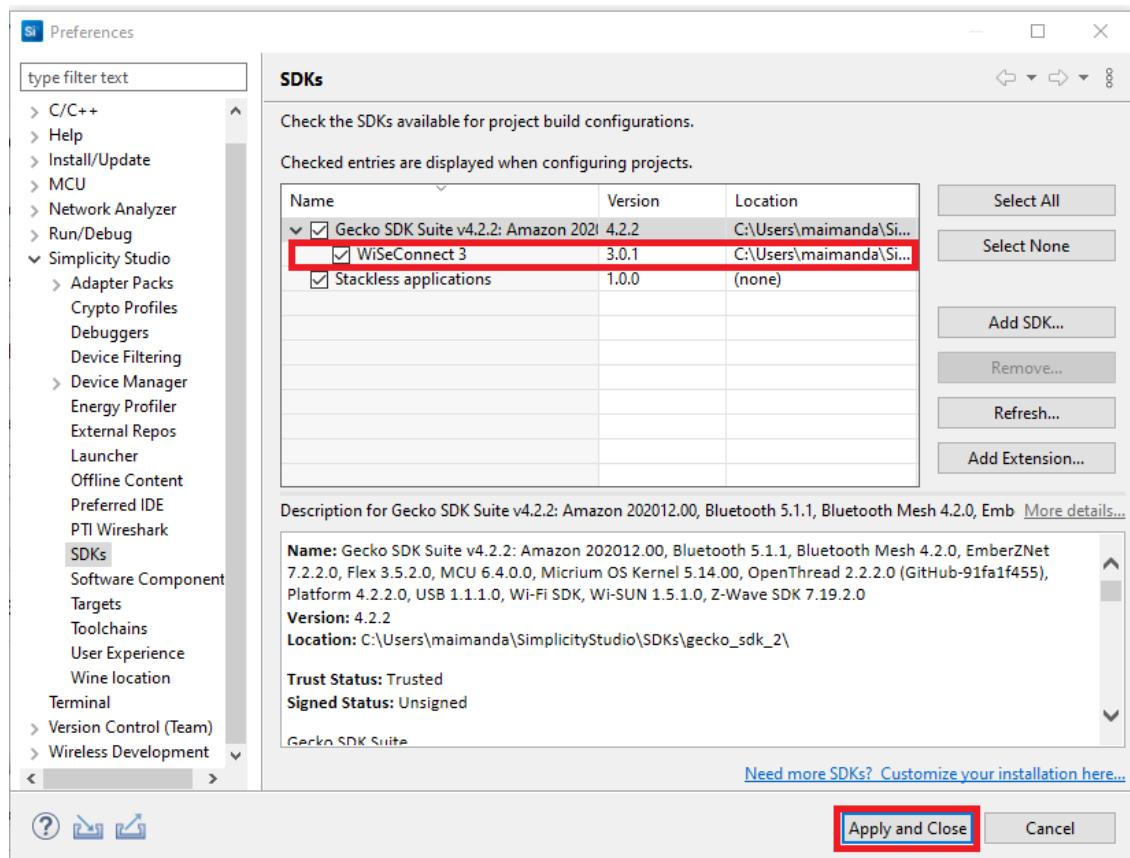
11. Select the detected extension and click **OK**.



12. If a Verify SDK Extension popup is displayed, click Trust.



13. The selected WiSeConnect SDK v2.x or v3.x extension will be displayed.



14. Click Apply and Close.

Get Started with SoC

Getting Started with SoC Mode

This guide describes how to get started developing an application for the SiWx91x in System-on-chip (SoC) mode, where both the application and the networking stack run on the SiWx917 chipset.

Check Prerequisites

In order to run Matter over Wi-Fi, check for the following prerequisites:

Hardware Requirements

The following hardware devices are required for executing Matter over Wi-Fi:

- Silicon Labs Wireless starter/development kit (WSTK)
- SiWx917 SoC development kit
- Wi-Fi Dev Kit
 - SiWx917
 - SoC mode:
 - BRD4388A (B0 2.0 common flash) SiWx917
- Windows/Linux/MacOS computer with a USB port
- USB cable for connecting WSTK Board to Computer
- Raspberry Pi with a >32 GB SD Card
- Access Point with Internet Access

Software Requirements

Below are the software tools, packages, and images required for executing Matter over Wi-Fi:

Software Tools Requirements

- Simplicity Commander for flashing firmware/binary
- Tera Term
- Simplicity Studio
- Putty for controlling EFR32 hardware using chip-tool controller
- JLink RTT for logging only

Software Packages

- Gecko SDK v4.x
- WiseConnect SDK v3.x

Firmware Images

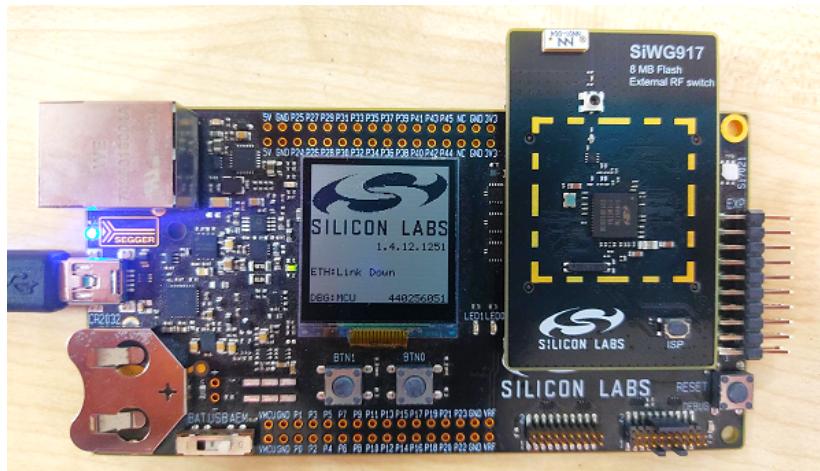
- Download the Firmware images from [Matter Artifacts Page](#).
- For Flashing the firmware images, Refer to [Flashing Firmware Images](#).

Installation of the Wi-Fi Software Tools and Packages

Refer to the [Wi-Fi Software Installation Page](#).

Connect SiWx917 SOC to Computer

1. Mount the SiWx917 radio board on the SiWx917 WSTK board.



2. Connect your SiWx917 Wireless Starter Kit (WSTK) board to your computer using a USB cable.
3. Simplicity Studio will detect and display your radio board.

Troubleshooting a Board Detection Failure

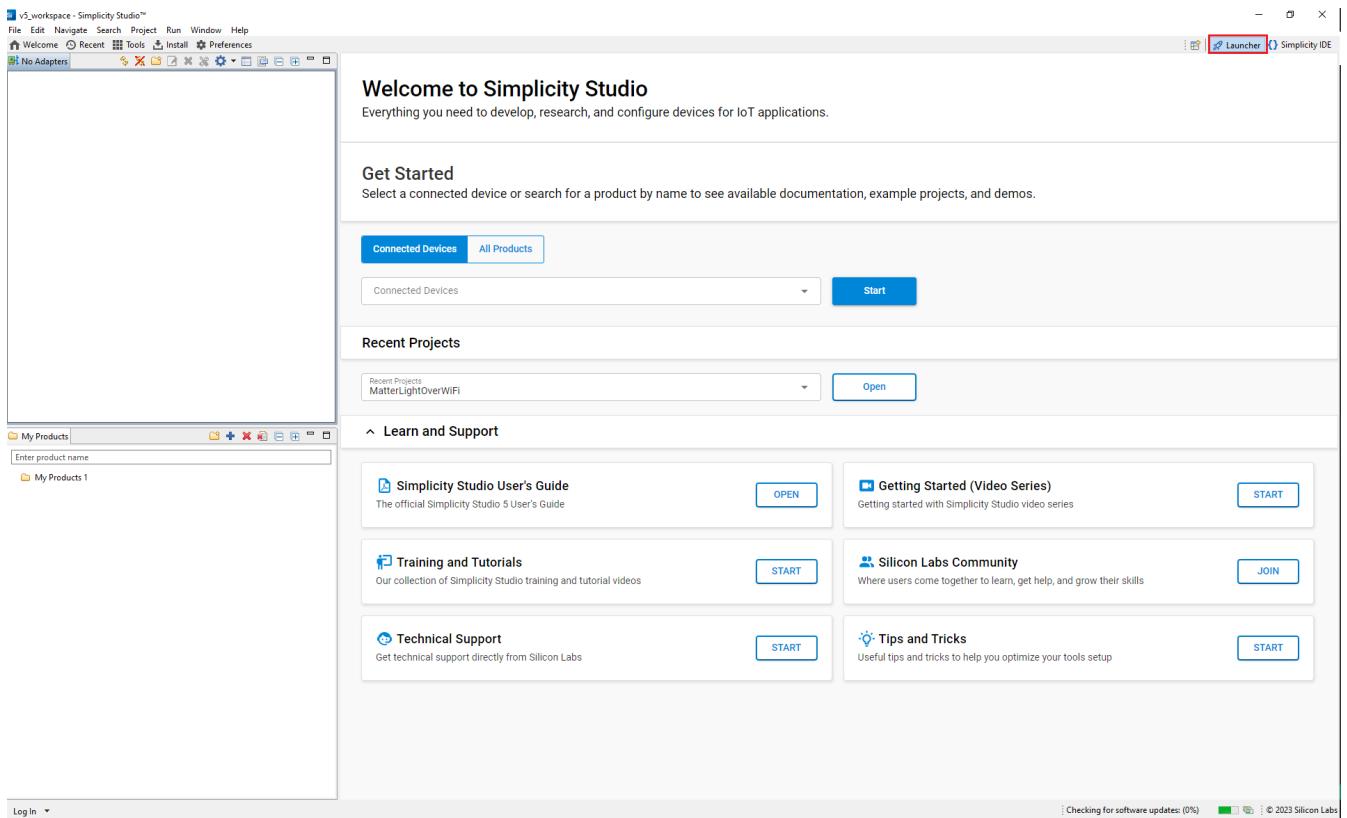
If Simplicity Studio does not detect the SiWx917 SoC board, try the following:

- In the Debug Adapters panel, click Refresh (the icon of two looping arrows).
- Press the RESET button on the SiWx917 SoC radio board.
- Power-cycle the SiWx917 SoC radio board by disconnecting and reconnecting the USB cable.

Building the 917 SoC Matter Accessory Devices Using Simplicity Studio

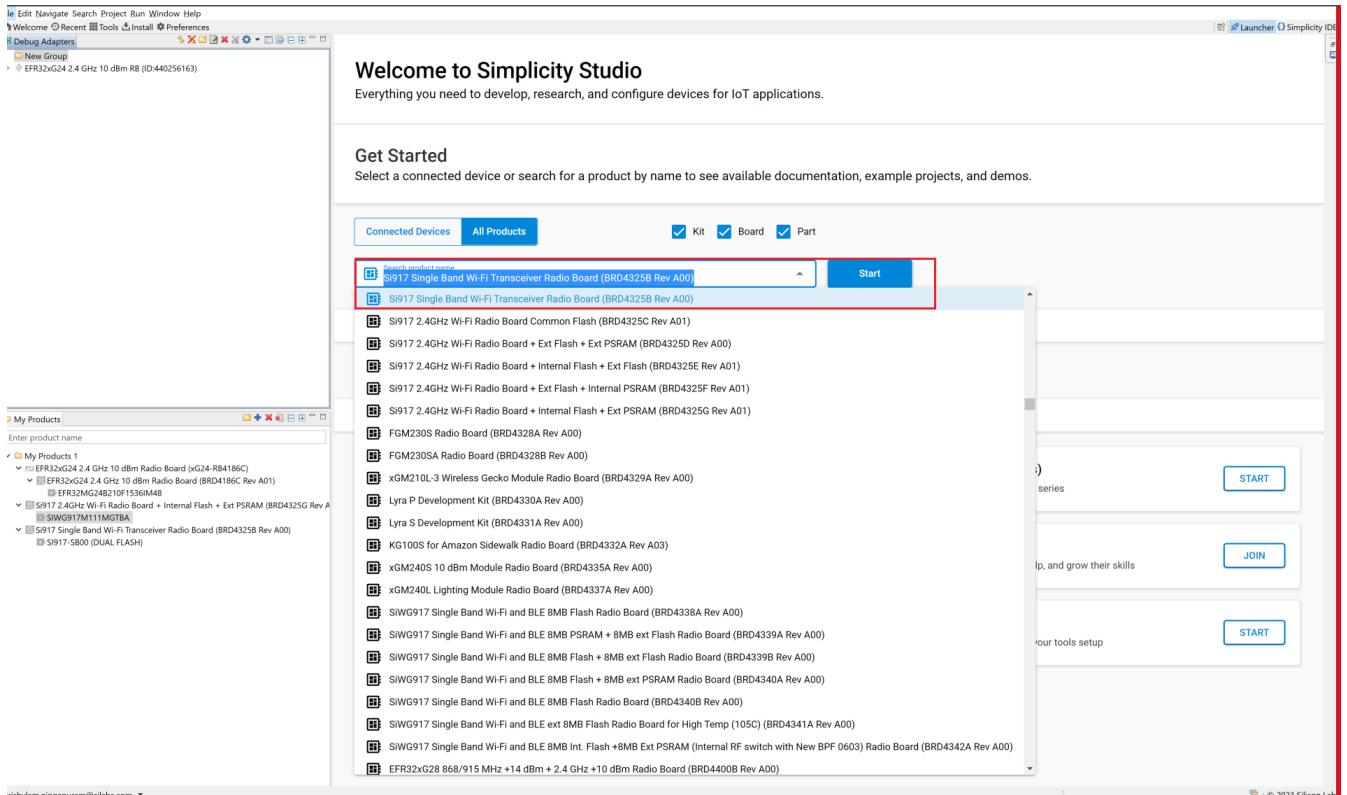
In Simplicity Studio 5, create the Light Matter Accessory Devices (MAD):

1. [Download](#) and Install Simplicity Studio 5.
2. To install the software packages for Simplicity Studio, refer to the [Software Package Installation Section](#)
3. Switch to the Launcher view (if not already in it).

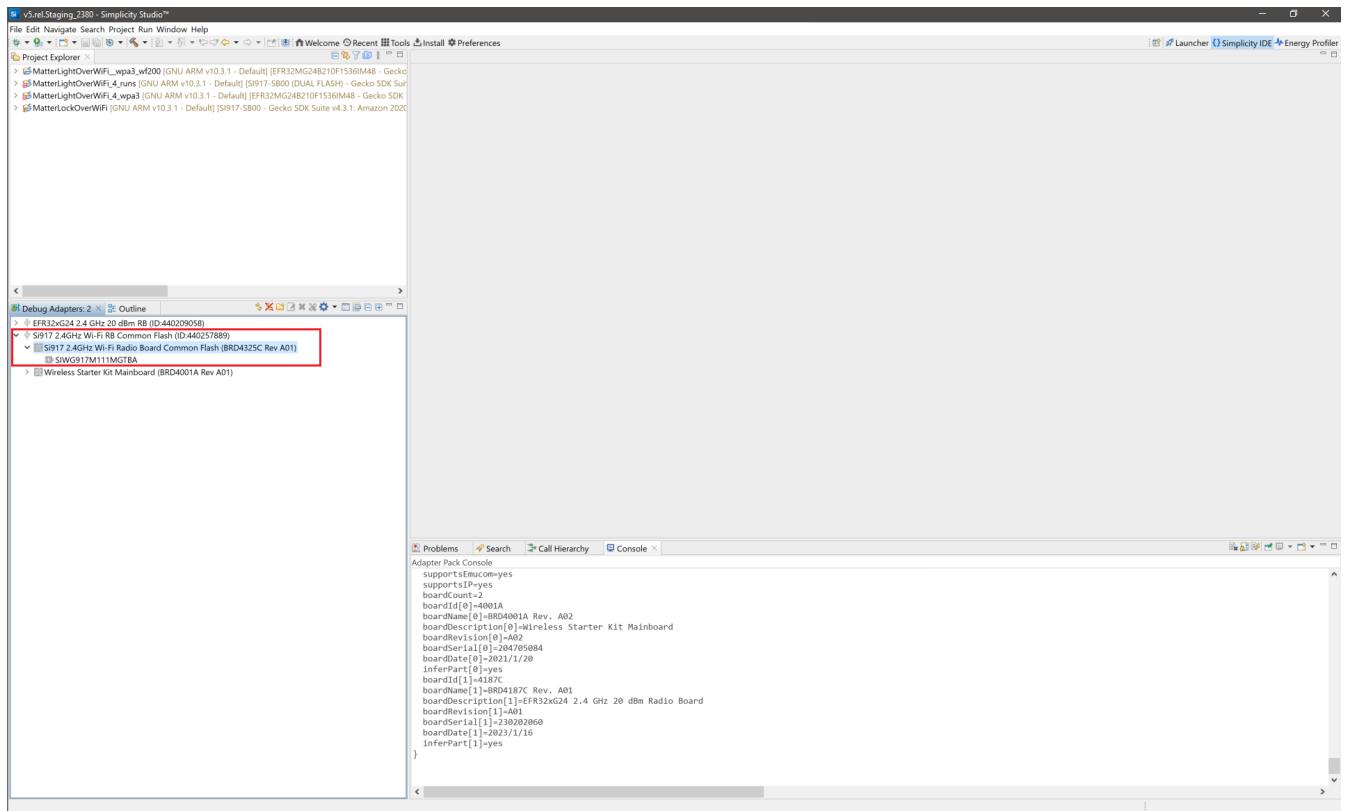


4. Go to All Products in the launcher tab and select one compatible board from the following supported list of SiWx917 SOC dev boards.

- BRD4338A (Common Flash)

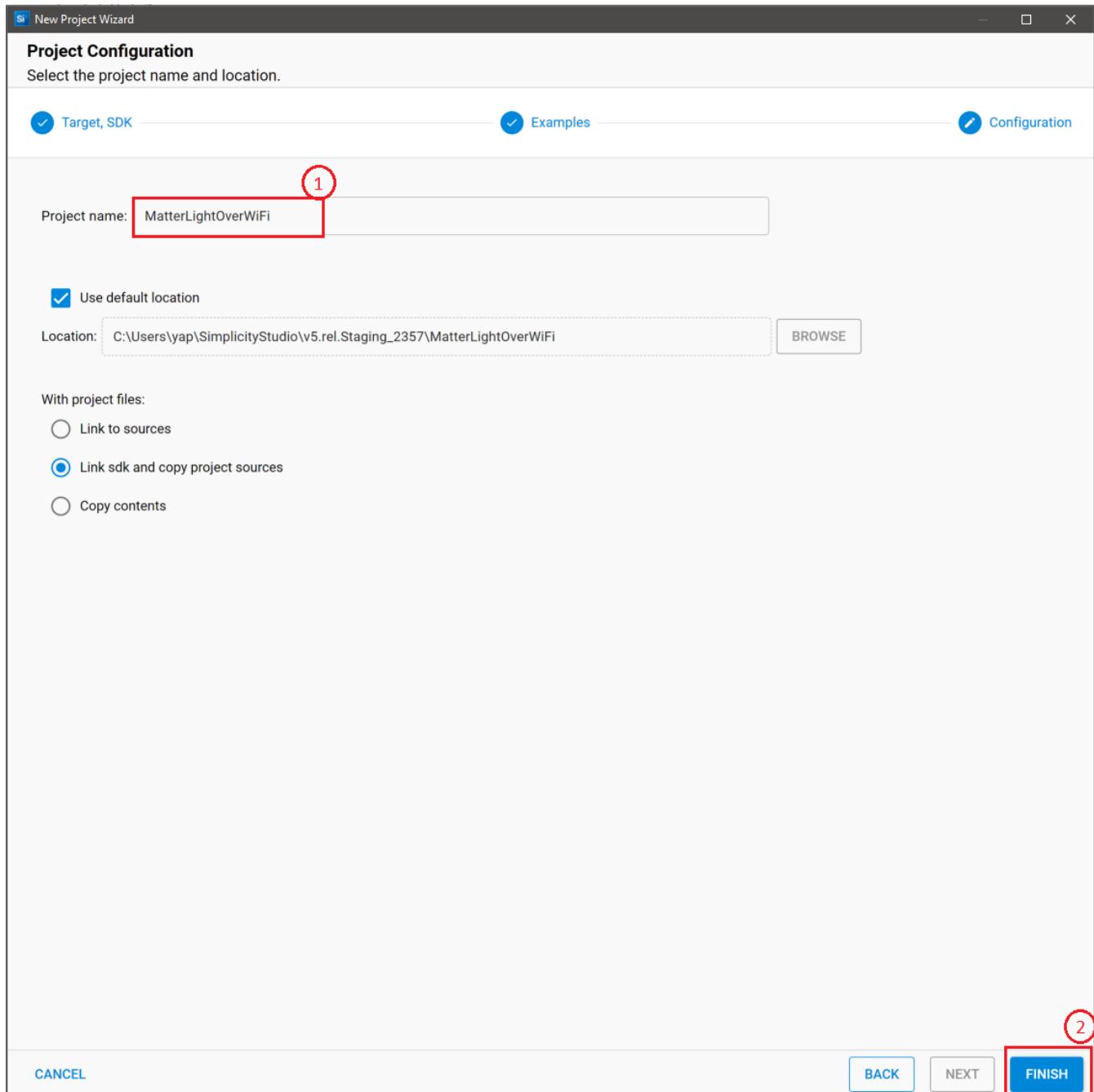


5. Once the board shows up in the Debug Adapters view, select it.



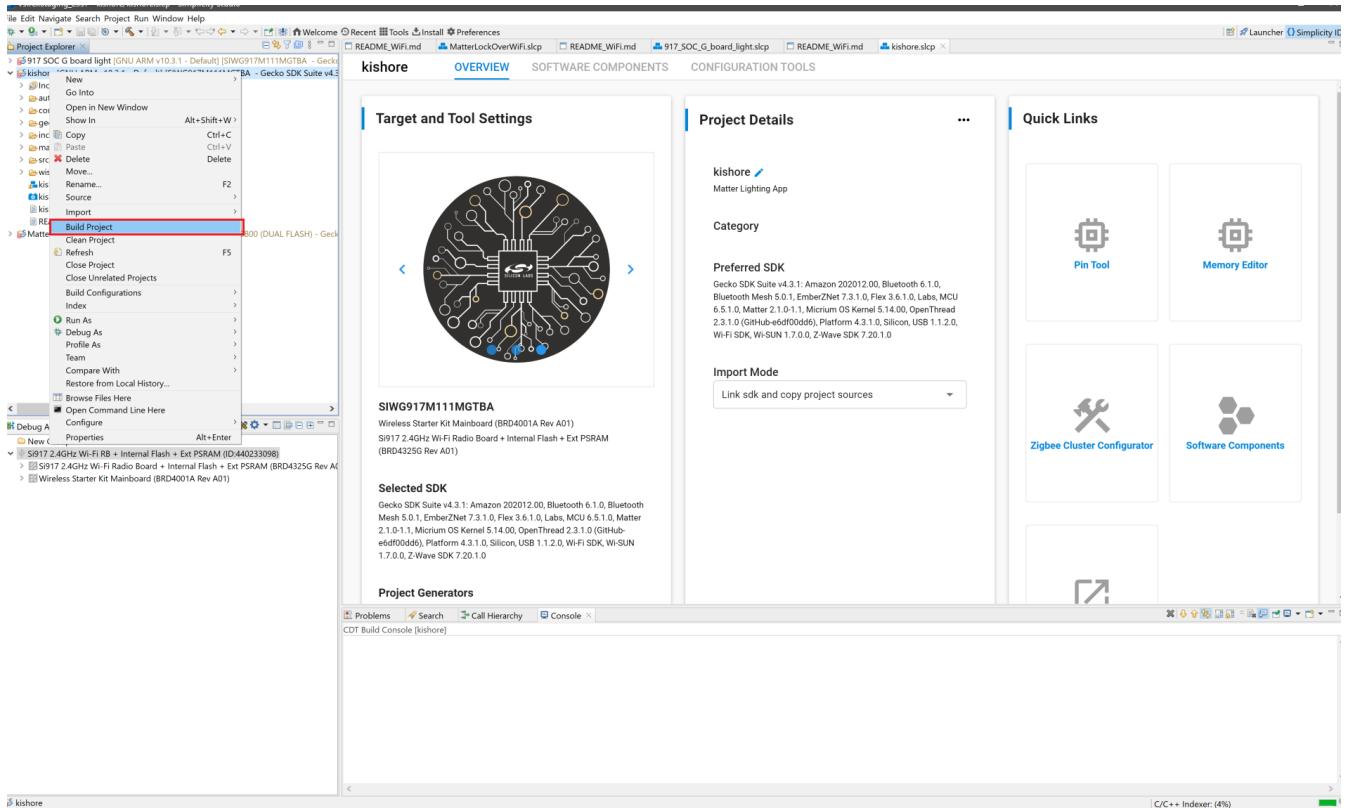
6. Open the Example Projects and Demos tab, select the **Matter** filter and enter "Wi-Fi" in **Filter on keywords**. Click **CREATE**.

The screenshot shows the 'Example Projects and Demos' tab of the Simplicity Studio interface. A red box highlights the 'EXAMPLE PROJECTS & DEMOS' tab. Another red box highlights the 'Matter' checkbox under the 'What are Demo and Example Projects?' section. A third red box highlights the 'CREATE' button for the 'Matter - SoC Lock over Wi-Fi' project. The left sidebar shows a list of products, including the Si917 2.4GHz Wi-Fi RB + Internal Flash + Ext PSRAM (BRD4325B Rev A00).



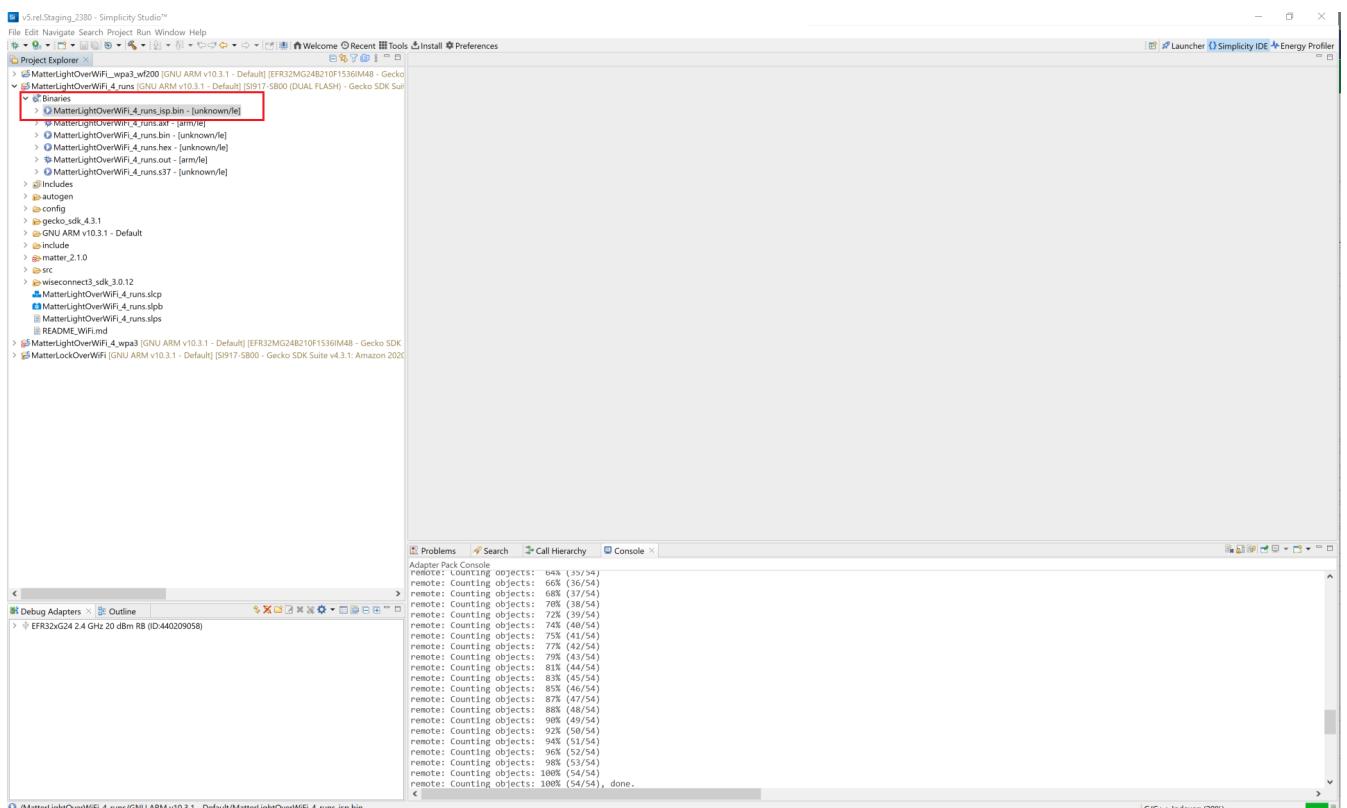
8. Once the project is created, right-click on the project and select *Build Project* in the Project Explorer tab.

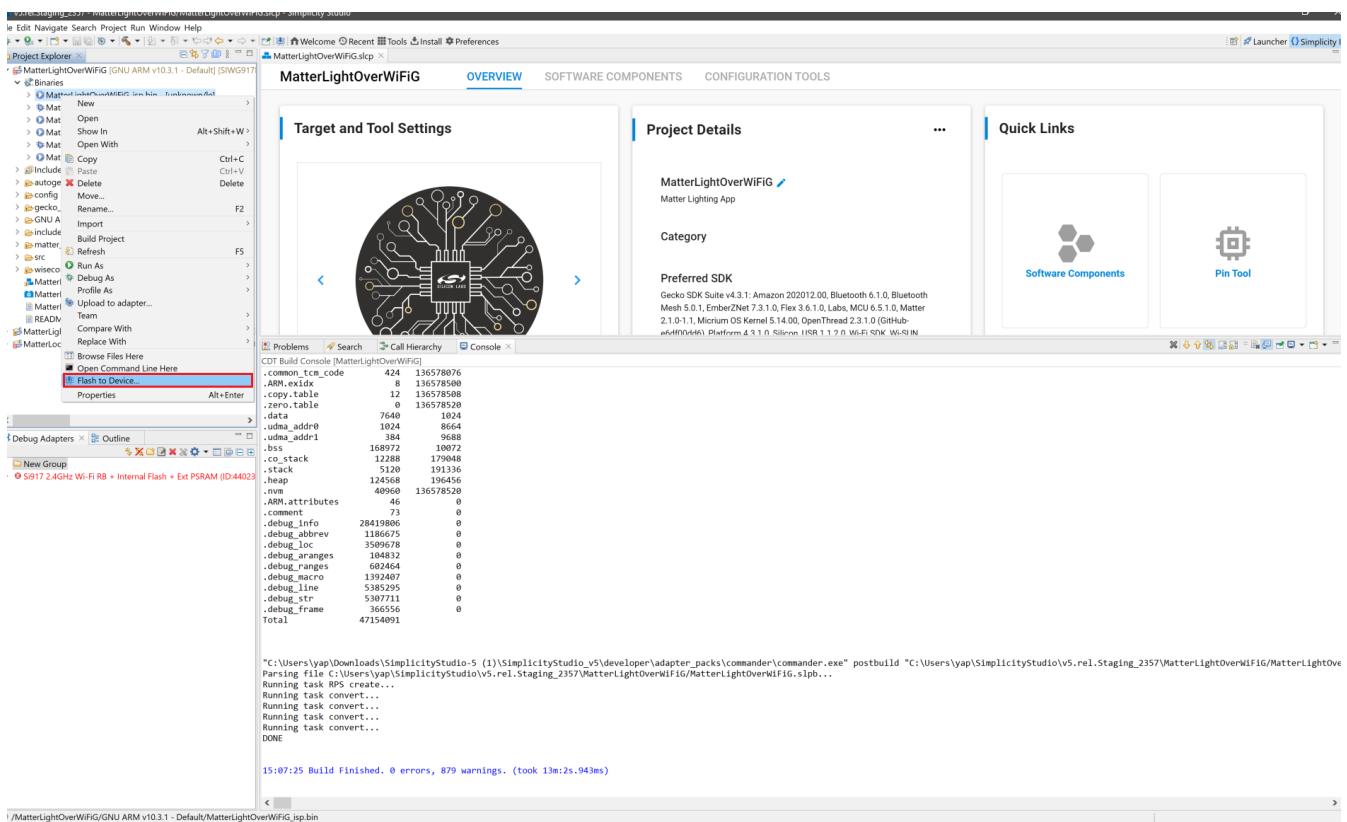
Get Started with SoC



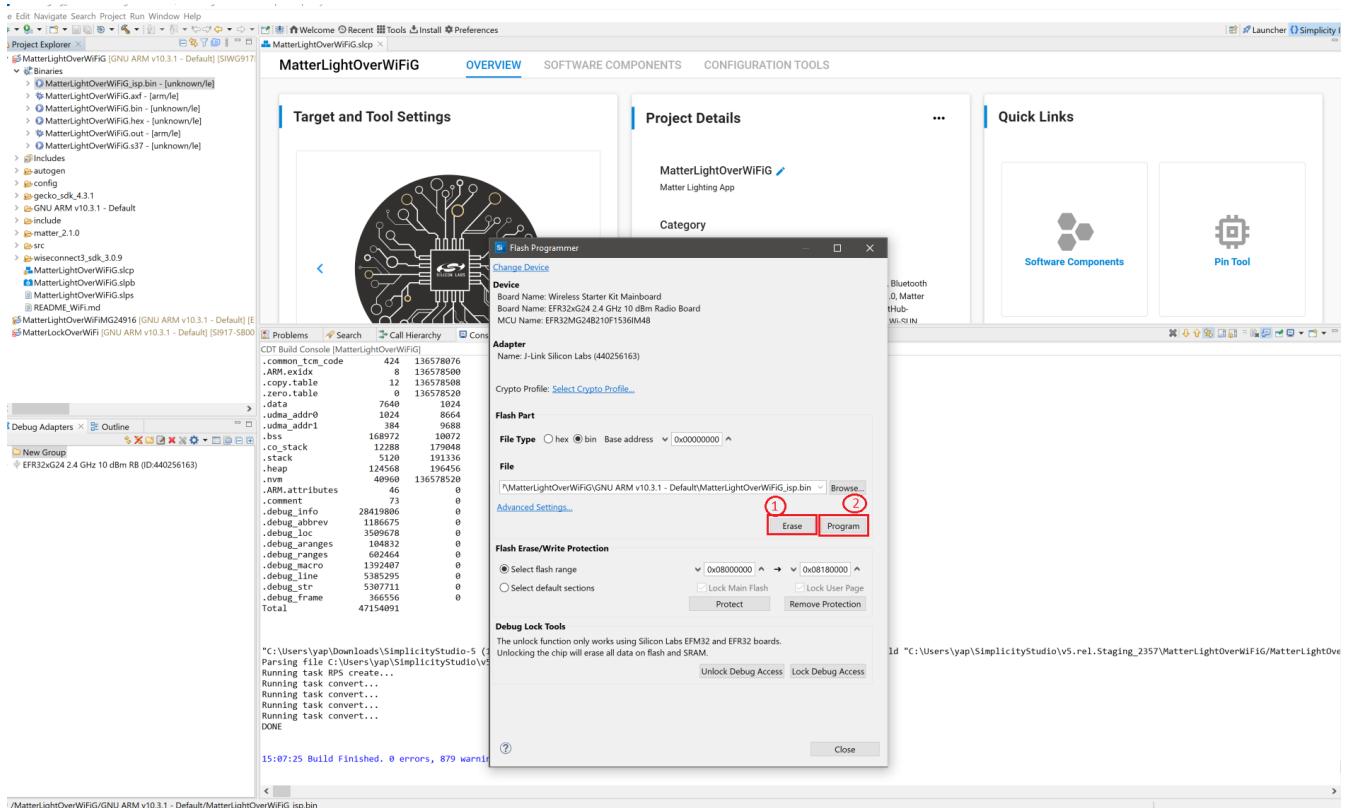
9. To flash the application, connect the compatible dev board to the PC if not yet done.

10. Once the project is compiled successfully, go to the Project Explorer view and select the binary to be flashed.



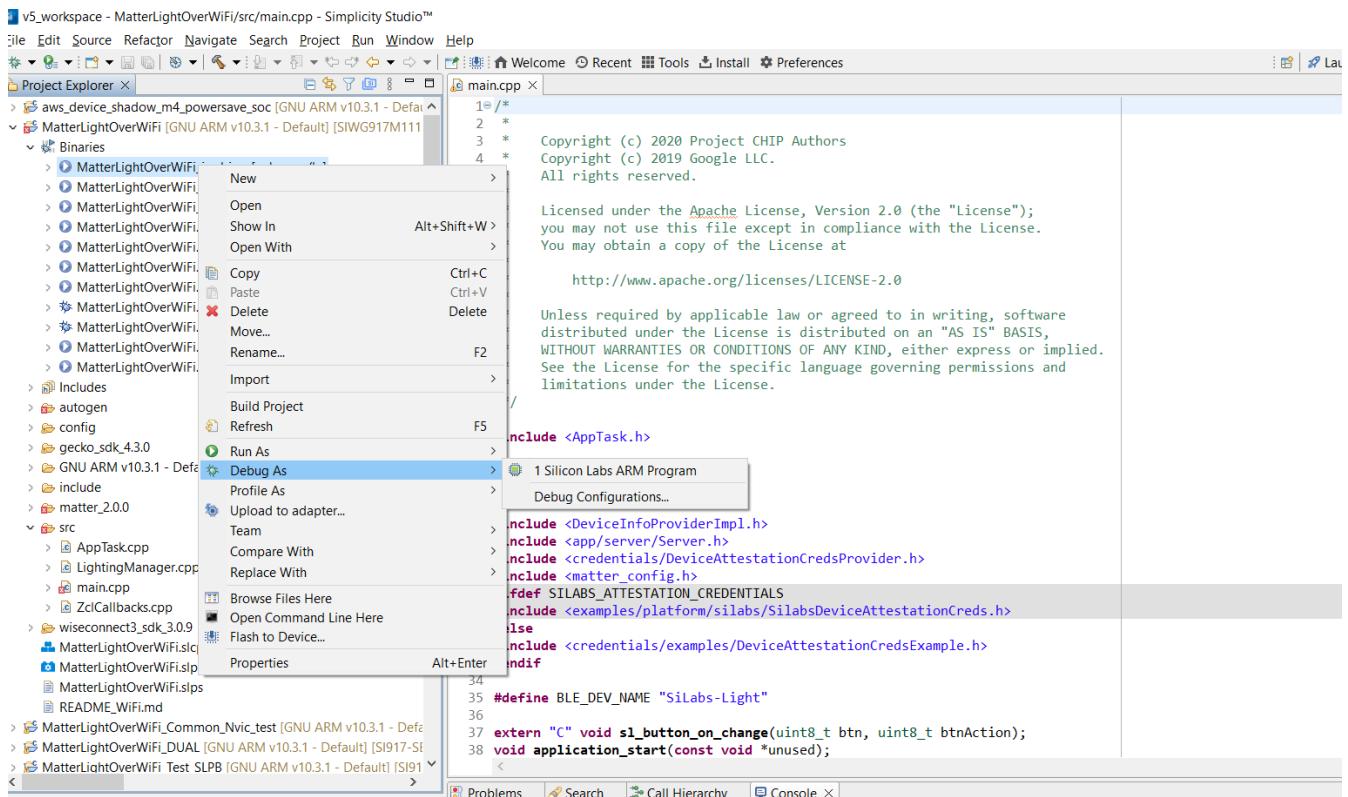


12. The Flash programmer window opens. Click **Program** to start flashing.



Note: Output of the SiWX917 SoC application will be displayed on the J-Link RTT Viewer.

In order to debug your Matter Application, Right-click on the selected Matter Project and click on *Debug As*.



Get Started with NCP

Getting Started with EFR32 Host in NCP Mode

This page describes how to get started with developing an application on EFR32 host in Network Co-Processor (NCP) mode, where the application runs on the EFR32 host and the connectivity stack runs on the Wi-Fi chipset.

Check Prerequisites

In order to run Matter over Wi-Fi, check for the following prerequisites:

Hardware Requirements

The following hardware devices are required for executing Matter over Wi-Fi:

- Silicon Labs Wireless starter/development kit (WSTK)
- Silicon Labs Wi-Fi development Kits & boards
 - For Network Co-Processor (NCP) variants,
 - Silicon Labs EFR32 - is used as a host processor and, with the WF200, provides Bluetooth LE capabilities
 - Silicon Labs Wi-Fi Processor
 - RS9116 development kit
 - WF200 expansion board
 - SiWx917 NCP expansion board
- MG24 boards:
 - BRD4186C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@10dBm
 - [XG24-RB4186C](#)
 - MG24 with WSTK : [xG24-PK6009A](#)
 - BRD4187C / SLWSTK6006A / Wireless Starter Kit / 2.4GHz@20dBm
 - [XG24-RB4187C](#)
 - MG24 with WSTK : [xG24-PK6010A](#)
- Wi-Fi Dev Kits & boards
 - RS9116
 - SB-EVK1 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK1](#)
 - SB-EVK2 / Single Band Wi-Fi Development Kit / 2.4GHz
 - [RS9116X-SB-EVK2](#)
 - DB-EVK1 / Dual Band Wi-Fi Development Kit / 2.4GHz & 5GHz
 - [RS9116X-DB-EVK1](#) **Note:** Matter only supported over 2.4GHz on this Dev kit.
 - SiWx917
 - SiWx917 NCP Mode / Wi-Fi Expansion Board / 2.4GHz
 - BRD8045A (B0 Expansion v2.0)
 - WF200
 - WF200 / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8022A](#)
 - WFM200S / Single Band Wi-Fi Expansion Board / 2.4GHz
 - [SLEXP8023A](#)
- Windows/Linux/MacOS computer with a USB port
- USB cable for connecting WSTK Board to Computer
- Raspberry Pi with a >32 GB SD Card
- Access Point with Internet Access
- Interconnect board (included in the Wi-Fi kits)
- SPI Cable (included in the RS9116 kit)

- Jumper Cables (included in the RS9116 kit)

Software Requirements

Below are the software tools, packages and images required for executing Matter over Wi-Fi:

Software Tools Requirements

- Simplicity Commander for flashing bootloader on EFR32 Boards and Siwx917 NCP.
- Tera Term for flashing firmware on EFR32 NCP Boards.
- Simplicity Studio
- Putty for controlling EFR32 hardware using chip-tool controller
- Ozone Debugger for logging and debugging (Optional)
- JLink RTT for logging only (Optional)

Software Packages

- Gecko SDK v4.x
- WiseConnect SDK
 - For RS9116 use WiseConnect SDK v2.x
 - For SiWx917 use WiseConnect SDK v3.x

Firmware Images

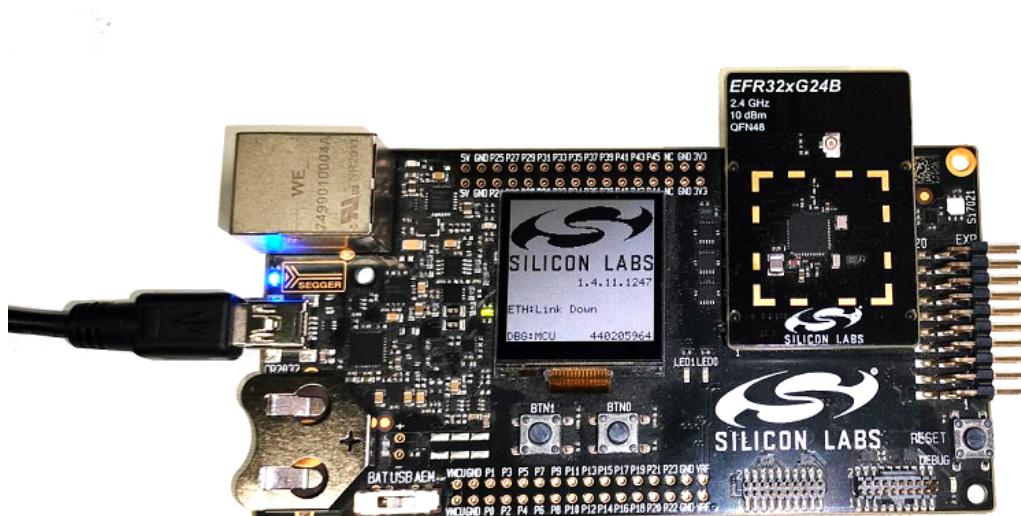
- Download the Firmware images from [Matter Artifacts Page](#)
- For Flashing the firmware images, Refer to [Flashing Firmware Images](#).

Installation of the Wi-Fi Software Tools and Packages

Refer to the [Wi-Fi Software Installation Page](#)

Connect the Boards to a Computer

1. Mount the EFR32 radio board on the EFR32 WSTK board.



2. Connect the NCP expansion board to the EXP header on the EFR32 WSTK board.



3. Toggle the upper switch on the NCP expansion board to EXP-UART.
4. Connect the EFR32 WSTK board to computer using a USB cable.

Troubleshoot Board Detection Failure

If Simplicity Studio does not detect the EFR32 radio board, try the following:

- In the **Debug Adapters** panel, click the **Refresh** button (the icon with two looping arrows).
- Press the **RESET** button on the EFR32 radio board.
- Power-cycle the EFR32 radio board by disconnecting and reconnecting the USB cable.

Building and Flashing an Application

This section describes how to create a project for the EFR32 boards.

1. In Simplicity Studio, click **Example Projects and Demos**, select a project, and click **Create**.

EFR32xG24 2.4 GHz 10 dBm Radio Board (BRD4186C Rev A00)

OVERVIEW EXAMPLE PROJECTS & DEMOS DOCUMENTATION COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos

Example Projects

Solution Examples

What are Demo and Example Projects?

Wireless Technology Clear

- Bluetooth (51)
- Bluetooth Mesh (20)
- Connect (9)
- Matter (45) 2
- RAIL (22)
- Thread (17)
- Zigbee (26)

Device Type Clear

- Host (1)
- NCP (0)
- RCP (0)
- Soc (30)

Ecosystem Clear

- Amazon (0)

MCU Clear

- 32-bit MCU (0)
- Bootloader (0)

Matter - SoC Light Switch over Wi-Fi
This project builds a Matter Light Switch for Wi-Fi WF200 that can be developed inside Simplicity Studio

[View Project Documentation](#) [CREATE](#)

Matter - SoC Lighting over Thread
This is a Matter Lighting Application over Thread for BRD4186C

[View Project Documentation](#) [RUN](#)

Matter - SoC Lighting over Wi-Fi
This project builds a Matter Lighting app for Wi-Fi RS9116 and 917NCP that can be developed inside Simplicity Studio

[View Project Documentation](#) [CREATE](#)

Matter - SoC Lock over Thread
This project builds a Matter Lock that can be developed inside Simplicity Studio

[View Project Documentation](#) [CREATE](#)

Matter - SoC Lock over Wi-Fi

Demo

Matter - SoC Light Switch over Wi-Fi
This is a Matter Light Switch Application for BRD4186C to be used with RS9116/917NCP Wi-Fi Evaluation kit

[View Project Documentation](#) [RUN](#)

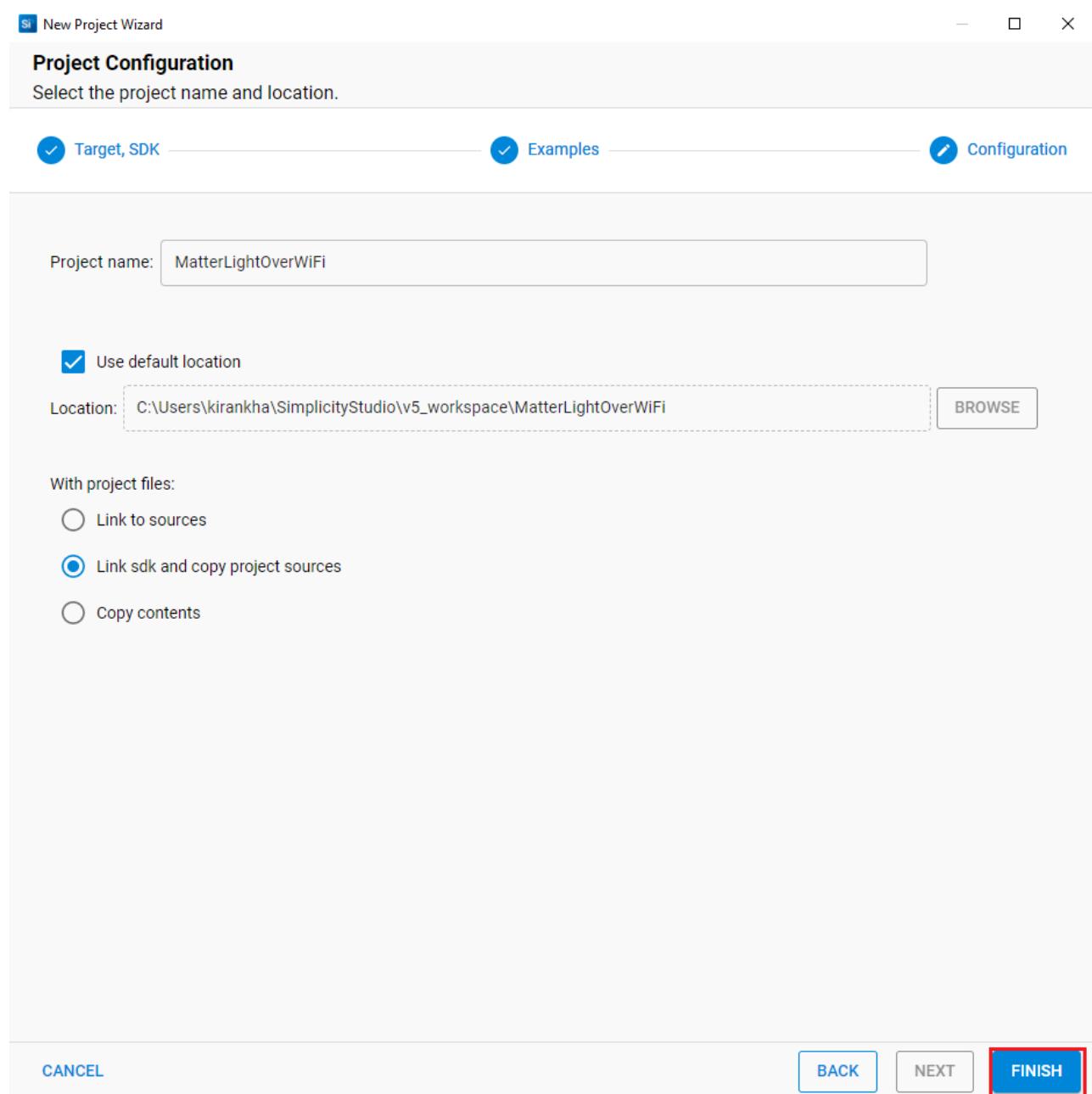
Matter - SoC Lighting over Thread
This project builds a Matter Lighting app that can be developed inside Simplicity Studio

[View Project Documentation](#) [CREATE](#)

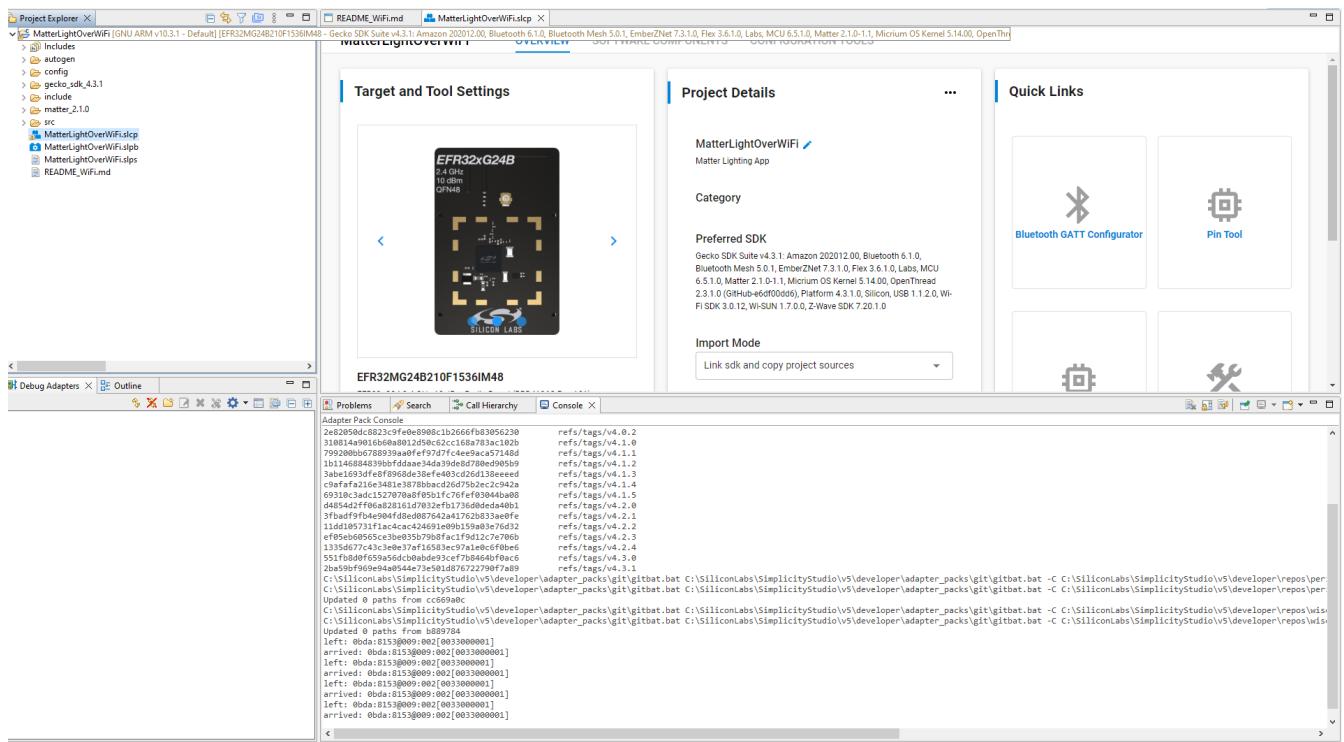
Matter - SoC Lighting over Wi-Fi
This is a Matter Lighting Application for BRD4186C to be used with RS9116/917NCP Wi-Fi Evaluation kit

[View Project Documentation](#) [RUN](#)

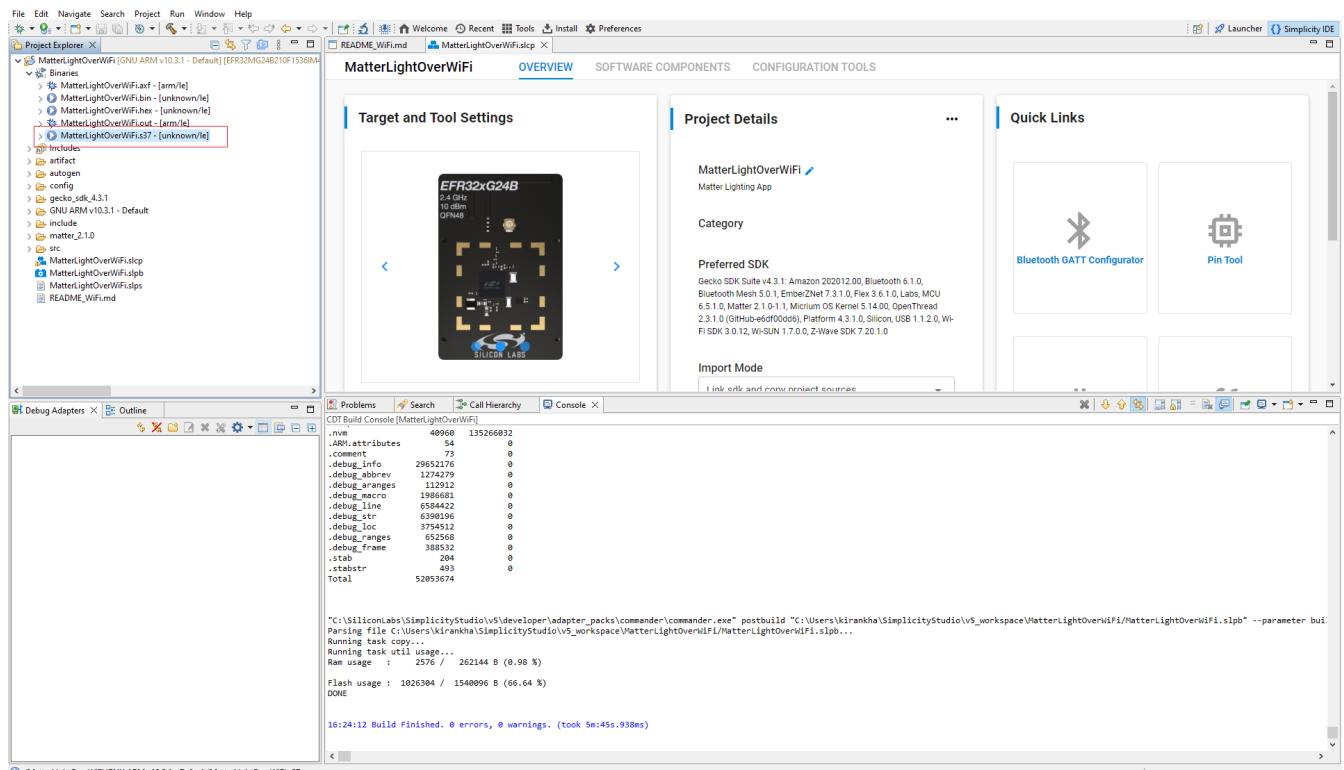
2. In the New Project Wizard window, click **Finish**.



3. Once the project is created, right-click the project and select **Build Project** in the Project Explorer tab.



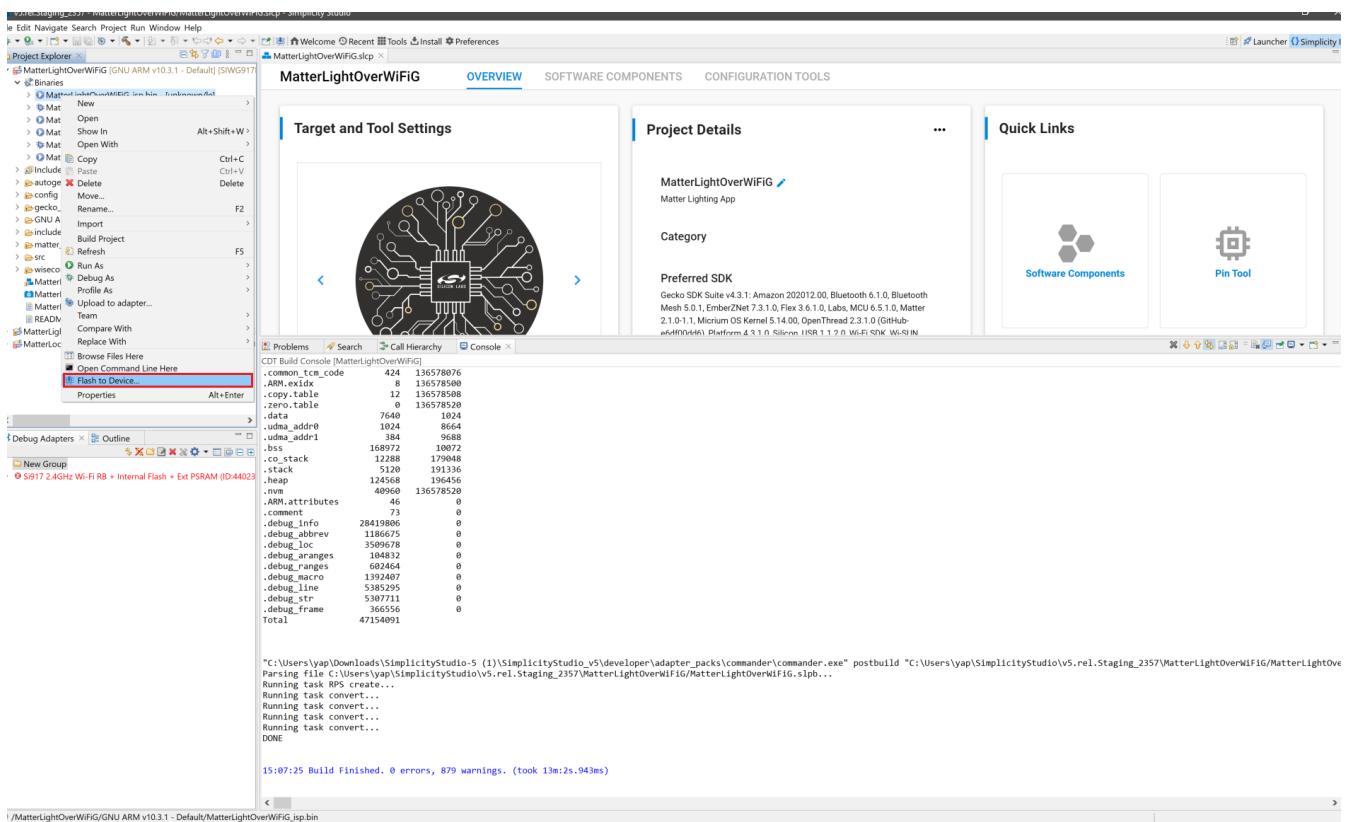
4. Once the project is compiled successfully, go to the Project Explorer view and expand the binaries folder to flash the binary.



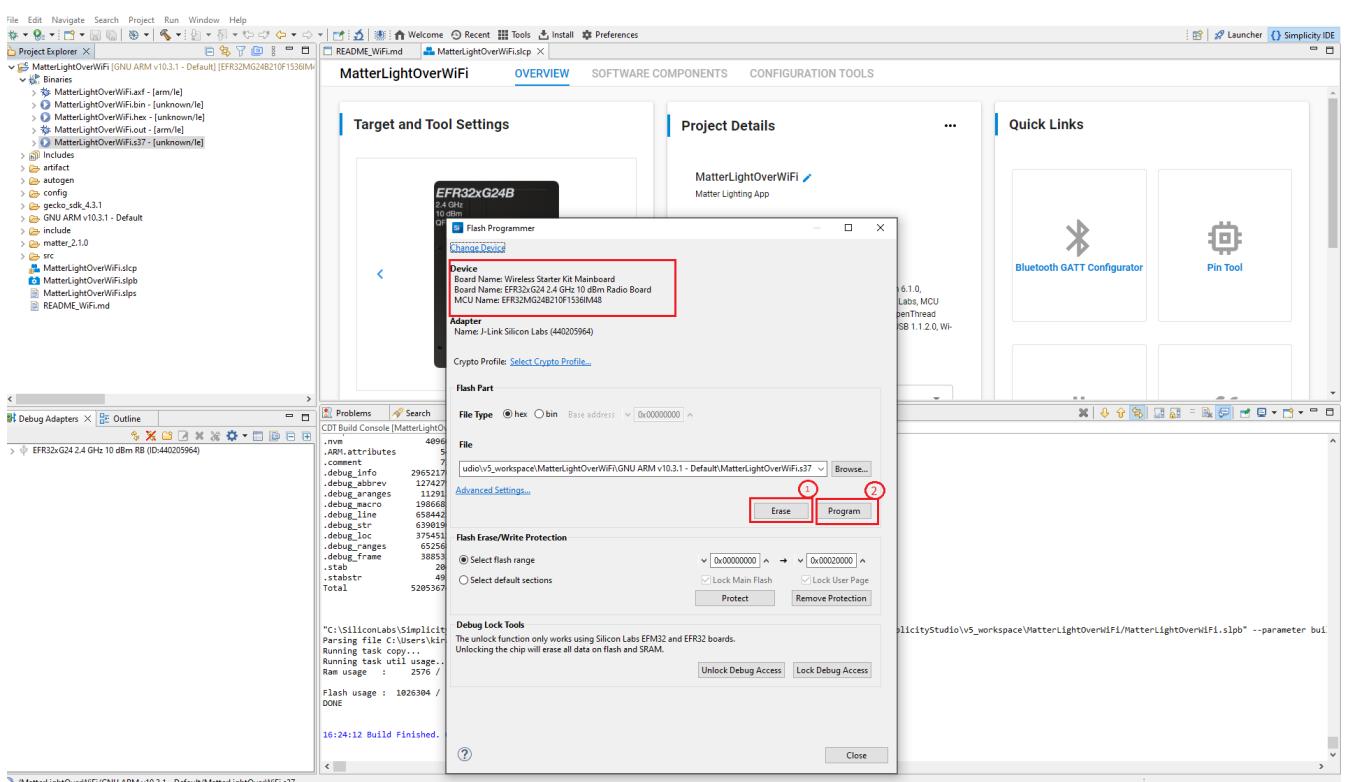
5. Right-click the selected .s37 binary and click **flash to device**.

Get Started with NCP

 SILICON LABS



6. Flash programmer window will be opened. Click **Erase** and then **Program** to start flashing.



Note: Output of the EFR32 NCP Host application will be displayed on the J-Link RTT Viewer.

Set up Chip-Tool

Building the Chip-Tool

This page covers:

- [Building the chip-tool for Linux](#)
- [Building the chip-tool for Raspberry Pi](#)

Build Environment for Linux

This section goes through the steps required to build the chip-tool for Linux.

Do not execute any commands on this page as ROOT (no *su* required), unless specified.

Prepare Linux Packages

Update the latest packages by typing following commands in the terminal:

```
$ sudo apt update  
$ sudo apt install
```

Prerequisites for Matter (CHIP) Project on Linux

Install Packages on Ubuntu Laptop/PC

- Open the Linux terminal from Start menu.
- Install required packages on Ubuntu Laptop/PC using the following commands:

```
$ sudo apt install git gcc g++ pkg-config libssl-dev libdbus-1-dev  
libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev
```

Build the chip-tool Environment

To build chip-tool environment, first set up the software and then compile the chip-tool.

Software Setup

If you have not downloaded or cloned the repository, you can run the following commands on a Linux terminal running on either Linux machine, WSL or Virtual Machine to clone the repository and run bootstrap to prepare to build the sample application images.

1. To download the [SiliconLabs Matter codebase](#), run the following commands.

```
$ git clone https://github.com/SiliconLabs/matter.git
```

2. Bootstrapping:

```
$ cd matter
$ ./scripts/checkout_submodules.py --shallow --recursive --platform efr32
$ . scripts/bootstrap.sh
# Create a directory where binaries will be updated/compiled called `out`
$ mkdir out
```

Compiling the chip-tool

To control the Wi-Fi Matter Accessory Device, you must compile and run the chip-tool on either a Linux, Mac, or Raspberry Pi. The chip-tool builds faster on the Mac and Linux machines so that is recommended, but if you have access to a Raspberry Pi, that will work as well.

1. Build the chip-tool.

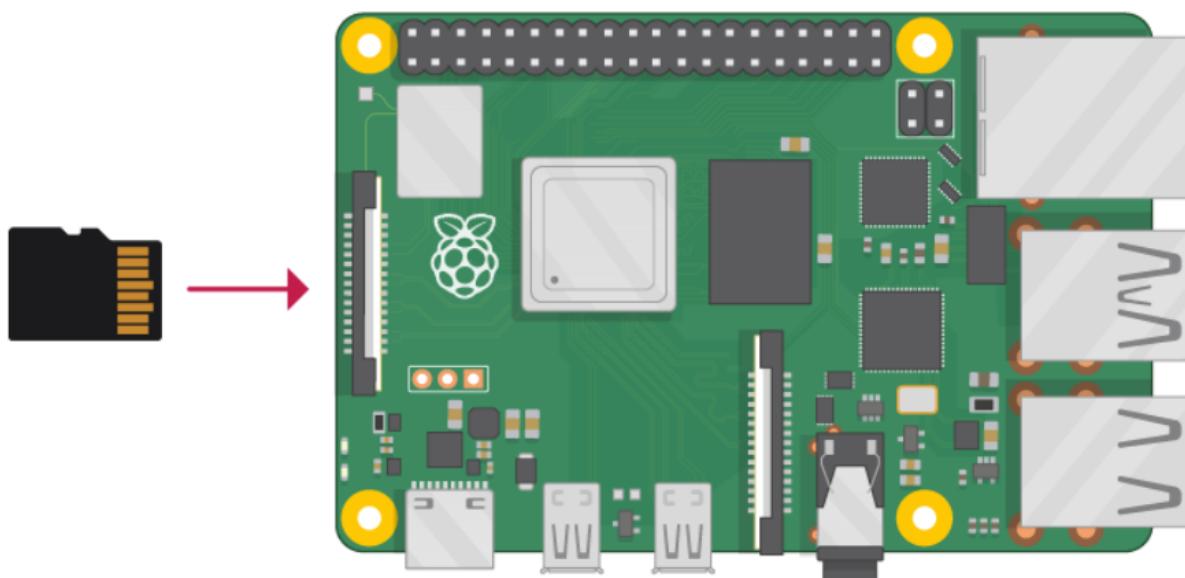
```
$ ./scripts/examples/gn_build_example.sh examples/chip-tool out/standalone
```

This will build chip-tool in `out/standalone`.

Build Environment using Raspberry Pi 4

Flash the Ubuntu OS onto the SD Card

1. Insert the flashed SD card (directly or using a card reader) into the laptop/PC that will run the Raspberry Pi Imager tool.
2. Launch Raspberry Pi 4 Imager.
3. Flash the Pi image using any one of the following procedure:
 - Click 'Choose OS' --> 'Other General-purpose OS' --> 'Ubuntu' --> 'Ubuntu xx.xx 64-bit server OS'
Note: Flash the latest version of Ubnutu Server (64-bit server OS for arm64 architecture)
 - Download the Matter Hub Raspberry Pi Image provided on the [Matter Artifacts page](#), then click 'Choose OS' --> 'Use custom' --> select the Matter Hub Raspberry Pi Image which you downloaded.
4. Click **Storage** and select the **SD card detect**.
5. This Raspberry Pi 4's console can be accessed in multiple ways. In [this guide](#), Raspberry Pi 4 is being accessed using Putty.
6. Enter the details like User name, Password, SSID, and its password to connect to network. Click **Save**.
7. Click **Write** and then **Yes** when you are asked for permission to erase data on the SD card. It will then start flashing the OS onto the SD card.
8. When it is done, click **Continue**.
9. Remove the SD card from the reader and insert it into the Raspberry Pi as shown below:



On powering up the board, the red and green lights should start blinking.

Start Using the Raspberry Pi

1. Power-up the RPi4B. Once it is booted up, check the Raspberry Pi's IP address. Refer to [Finding Raspberry Pi IP address](#) in the References chapter to get the IP address or enter the Hostname directly in PuTTY.
2. Once you find the IP address, launch Putty, select **Session**, enter the IP address of the Raspberry Pi, and click **Open**.
3. Enter the username and password given at the time of flashing and click **Enter**.
Note: If the username and password are not provided while flashing then by default:
 - Username: ubuntu
 - Password: ubuntu
4. Switch to root mode and navigate to path "/home/ubuntu/connectedhomeip/out/standalone" to find the chip-tool. Matter hub/chip-tool are ready and working. Keep the PuTTY session open for later steps.
5. Update the latest packages by running following commands in the terminal:

```
$ sudo apt update  
$ sudo apt install
```

6. Install required packages using the following commands:

```
$ sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev
```

If you see any popups between installs, you can select **OK** or **Continue**.

Build Environment

1. Follow the instructions in [the Project CHIP GitHub Site](#), in the section "Installing prerequisites on Raspberry Pi 4".
2. To build the environment, follow the [Software setup](#) and [Compiling chip-tool](#) steps given in [Software setup](#).

Bluetooth Setup

Because Bluetooth LE (BLE) is used for commissioning on Matter, make sure BLE is up and running on Raspberry Pi. Raspberry Pi internally has some issues with BLE that may cause it to crash.

```
$ sudo systemctl status bluetooth.service
```

To stop BLE if it is already running:

```
$ sudo systemctl stop bluetooth.service
```

To restart the Bluetooth service, first enable it:

```
$ sudo systemctl enable bluetooth.service
```

When you check the status of the Bluetooth service, it will be inactive because it has been enabled but not restarted:

```
$ sudo systemctl status bluetooth.service
```

Restart the service:

```
$ sudo systemctl restart bluetooth.service
```

Now the status of the service should be active and running:

```
$ sudo systemctl status bluetooth.service
```

Running the Matter Demo

Running the Matter Demo over Wi-Fi

Follow the procedure below to run Matter demo over Wi-Fi.

Flashing the Connectivity Firmware

- To flash the connectivity firmware on an NCP device, refer to the following guide: [Upgrading Connectivity Firmware for NCP Devices](#).
- To flash the connectivity firmware on a SiWx917 SoC, refer to the following guide [Upgrading Connectivity Firmware for SoC Devices](#).

Flashing the Bootloader Binary

- The bootloader binary is supported on EFR32 boards only and it can be flashed using the Simplicity Commander software or Simplicity Studio.
- To Flash the Bootloader Binary for EFR32 Board, refer to the following guide: [Flashing Bootloader Binaries on EFR32 Devices](#).

Building and Flashing the Matter Application using Simplicity Studio

- To build and flash an application for the EFR32, refer to [EFR32 Building and Flashing Application](#).
- To build and flash an application for SiWx917 SoC, refer to [SiWx917 SOC Building and Flashing Application](#).

Flashing the Matter Pre-Built Binaries Using Simplicity Commander

- To flash the application for EFR32 Board using Simplicity Commander, refer to [Flash EFR32 Binaries using Simplicity Commander](#).
Note: For EFR32, you must use the .s37 format file only.
- To flash the application for the SiWx917 SoC Board using Simplicity Commander, refer to [Flash SiWx917 SoC Matter Pre-Built Binaries](#).
Note: For SiWx917 SoC, use the .rps format file only.

Setting up the Raspberry Pi

To set up the Raspberry Pi, refer to [Setting up the chip-tool on Raspberry PI](#).

Next Steps

- [Running the Demo](#)
- [Debugging the Application](#)

Flash Firmware

Upgrading the Wi-Fi Connectivity Firmware

- It is recommended that an upgrade of the NCP combos connectivity firmware be done under the following circumstances:
 - When the EFR32 evaluation kit (EVK) is first received.
 - When the radio board is first received.
 - When upgrading to a new version of the WiSeConnect SDK v2.x or v3.x extension.

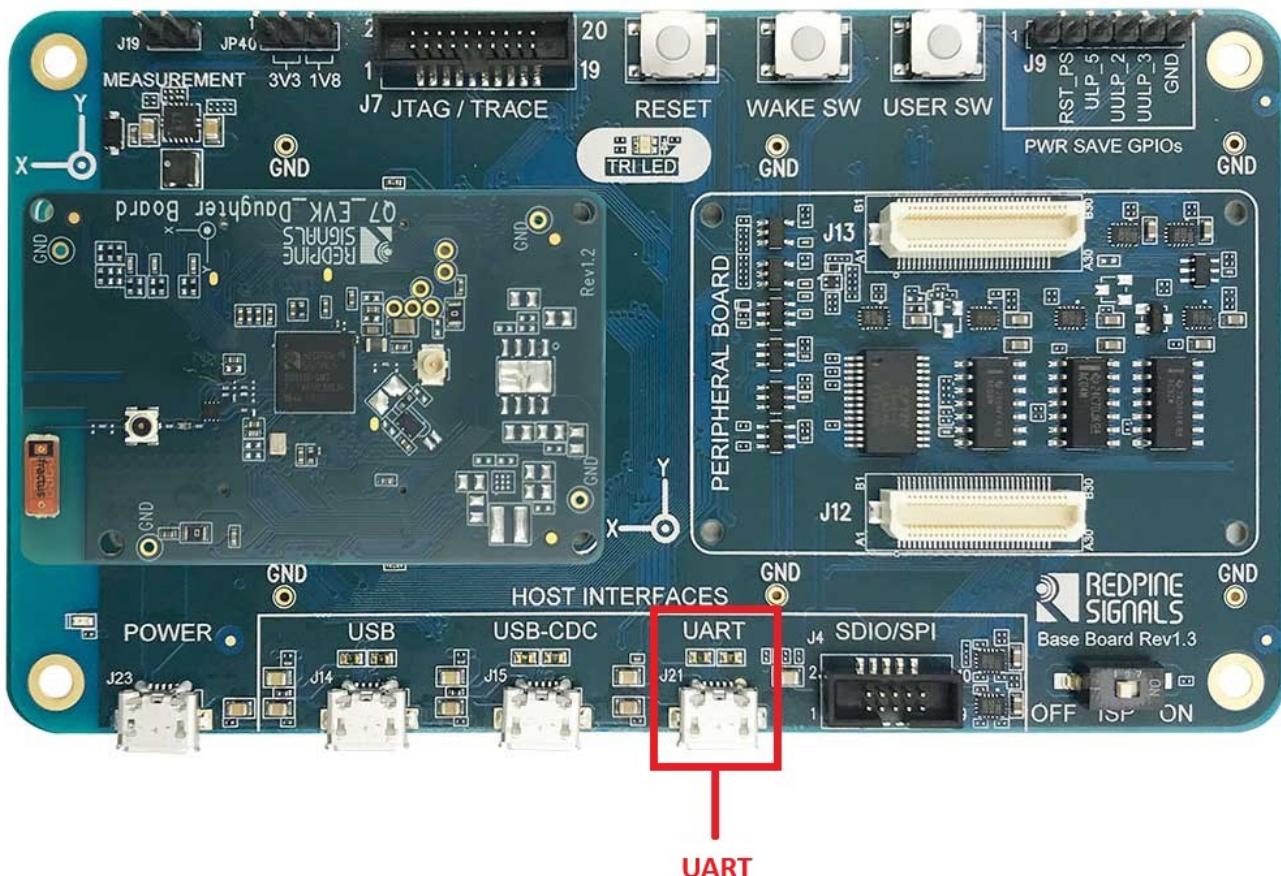
Upgrading the Connectivity Firmware on NCP devices

- The SiWx917 NCP or RS9116 EVK connectivity firmware can be upgraded using Teraterm or kermit.

Connectivity Firmware Upgrade Using Teraterm

Firmware Upgrade On RS9116

- Connect the EVK to PC using the USB interface labeled UART as identified below.



- If this is the first time connecting the EVK to your PC, verify that it is properly detected by the PC. The EVK will appear to the PC as a COM port labeled USB Serial Port (COMx)
- Configure your terminal application with the following settings:
 - Configure the serial port settings to 115200 baud / 8-bit data / No parity / 1 stop bit

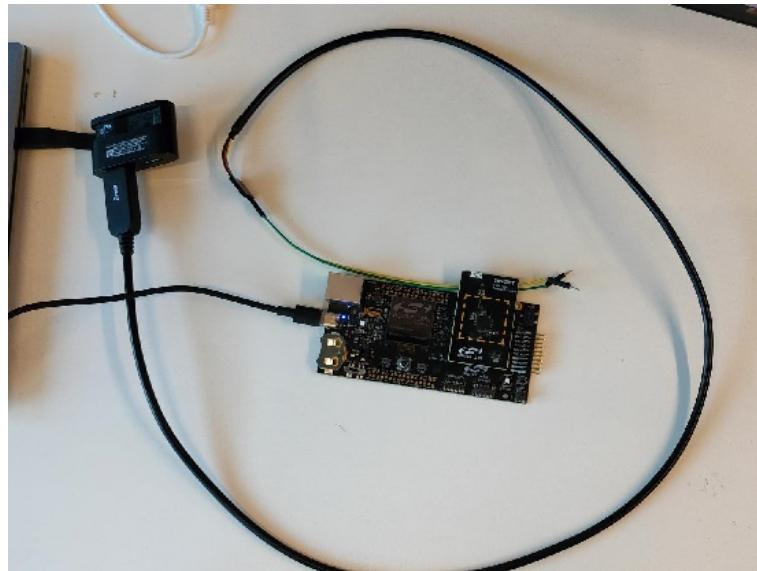
- Enable local echo
 - Set receive and transmit new-line characters to CR+LF
4. Refer to [Setup Tera Term and Updating the Firmware](#).

Instructions are the same for both SiWx917 NCP and RS9116 EVK.

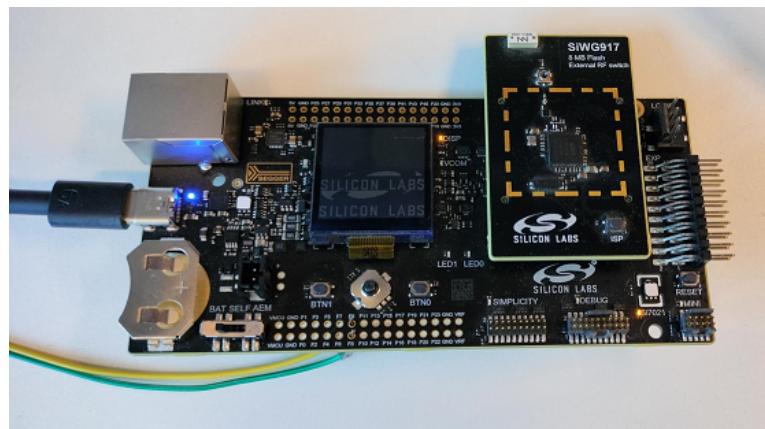
5. Once firmware flashing is done The console displays **Loading...** followed by **Loading Done.**

Firmware Upgrade On SiWx917 NCP

1. Connect USB-UART Cable to Machine and WPK board as well with SOC Mounted on it.



2. Connect USB-UART Cable 2(Yellow) to F9 and 3(Green) to F8 on WPK Board shown below.



3. Configure your terminal application with the following settings:

- Configure the serial port settings to 115200 baud / 8-bit data / No parity / 1 stop bit
- Enable local echo
- Set receive and transmit new-line characters to CR+LF

4. Refer to [Setup Tera Term and Updating the Firmware](#).

Instructions are the same for both SiWx917 NCP and RS9116 EVK.

5. Once firmware flashing is done The console displays **Loading...** followed by **Loading Done.**

Troubleshooting an NCP Firmware Update Failure

If the firmware update fails, try the following:

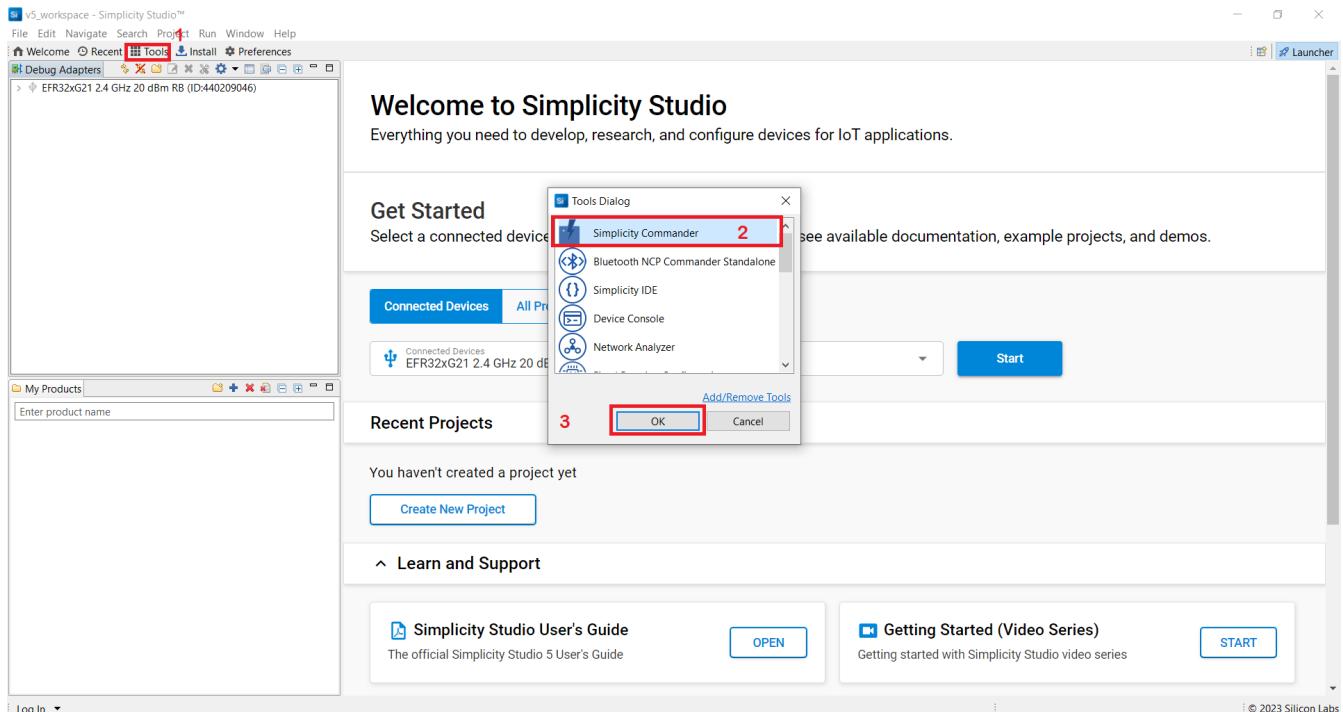
- Toggle the power switch towards AEM (Advanced Energy Monitoring) on the WPK board.
- Perform the following steps and try the firmware update again
 - Press the RESET button on the WSTK board.
 - Retry the firmware upgrade.

Upgrading the Connectivity Firmware on SoC Devices

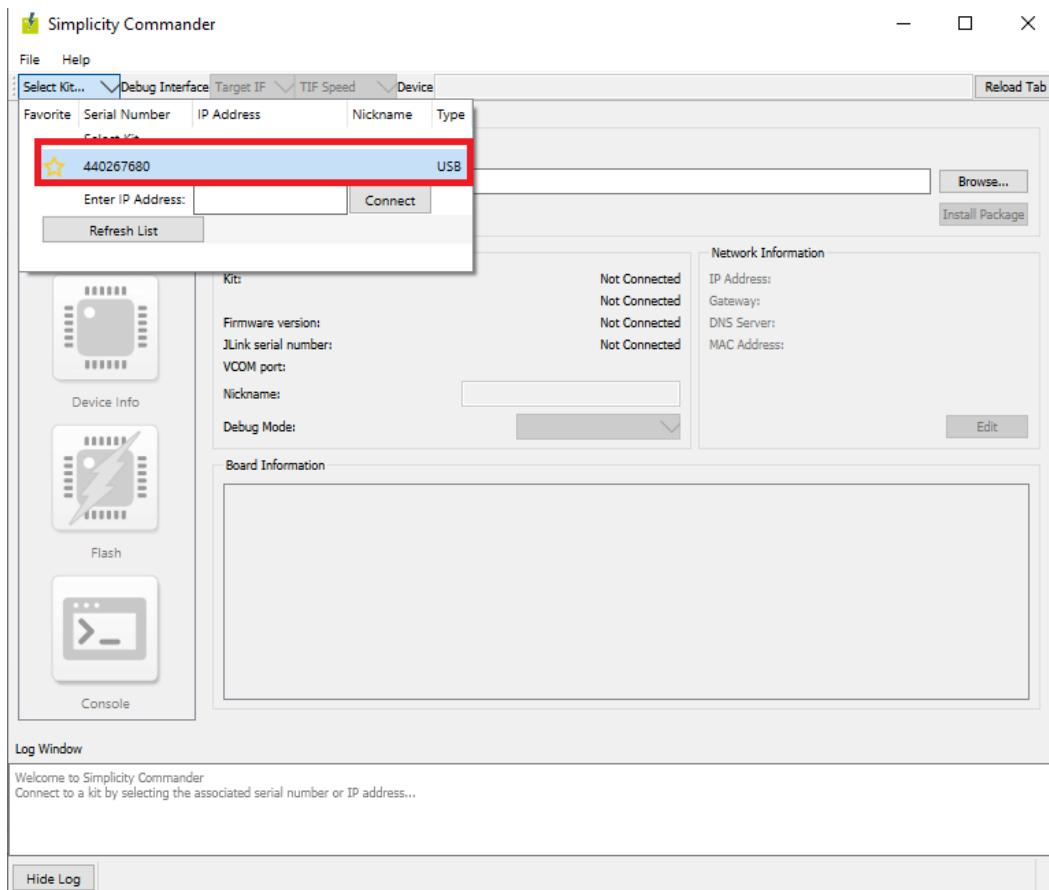
- SiWx917 SOC connectivity firmware can be upgraded using Simplicity Commander.

Connectivity Firmware Upgrade Using Simplicity Commander

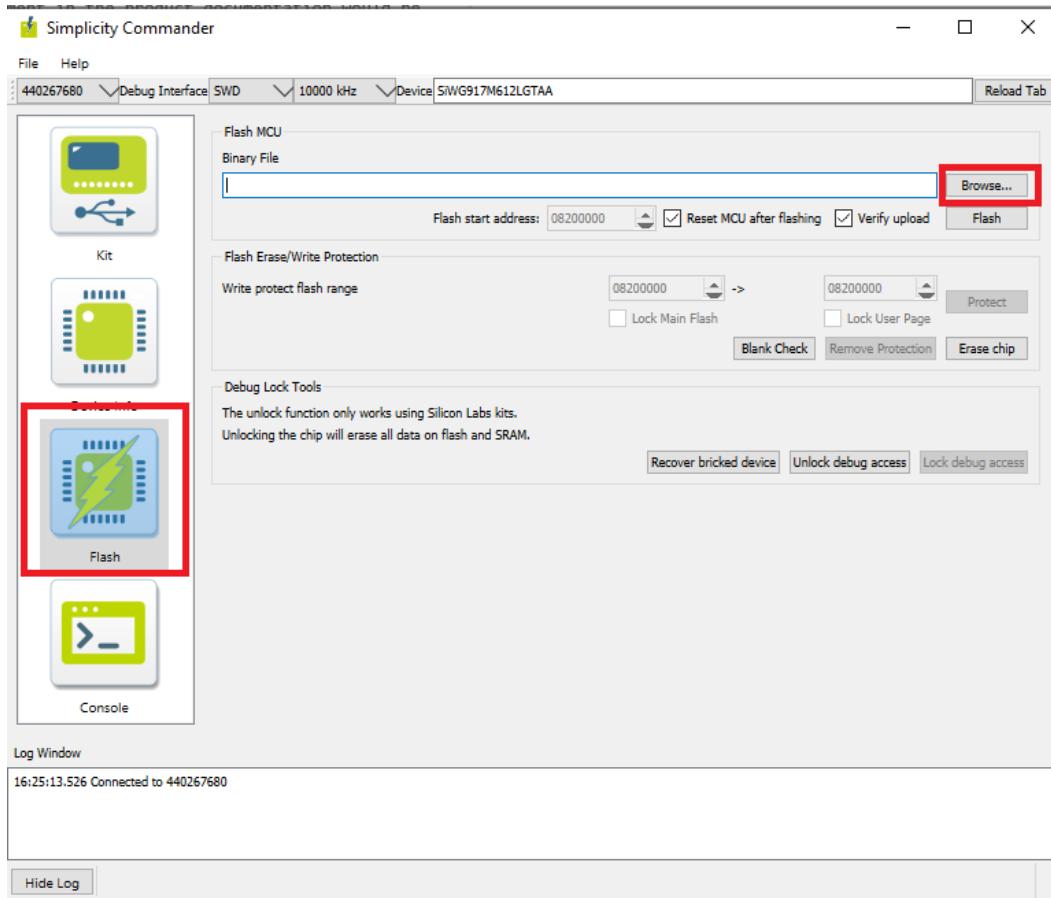
1. In the Simplicity Studio home page, click Tools.
2. In the Tools dialog, select Simplicity Commander and click OK.



3. In the Simplicity Commander window, click Select Kit and choose your radio board.

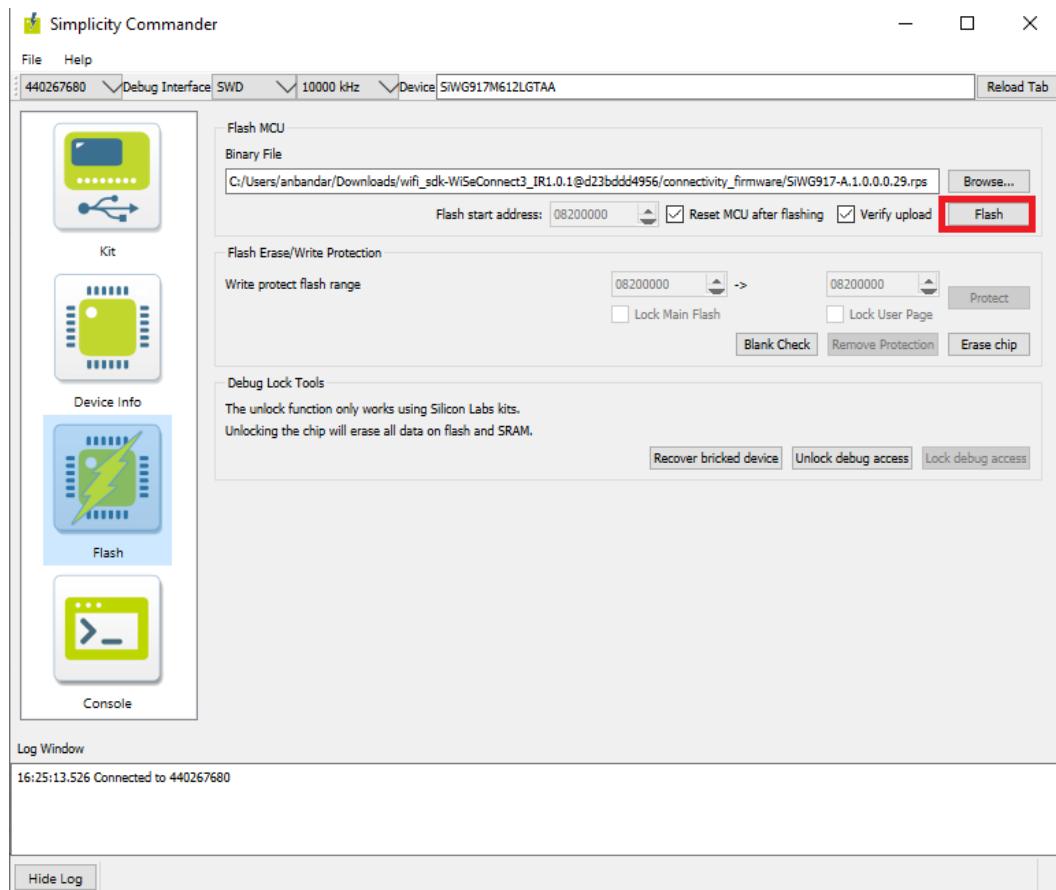


4. In the navigation pane, go to the Flash section.
5. Click Browse next to the Binary File field.

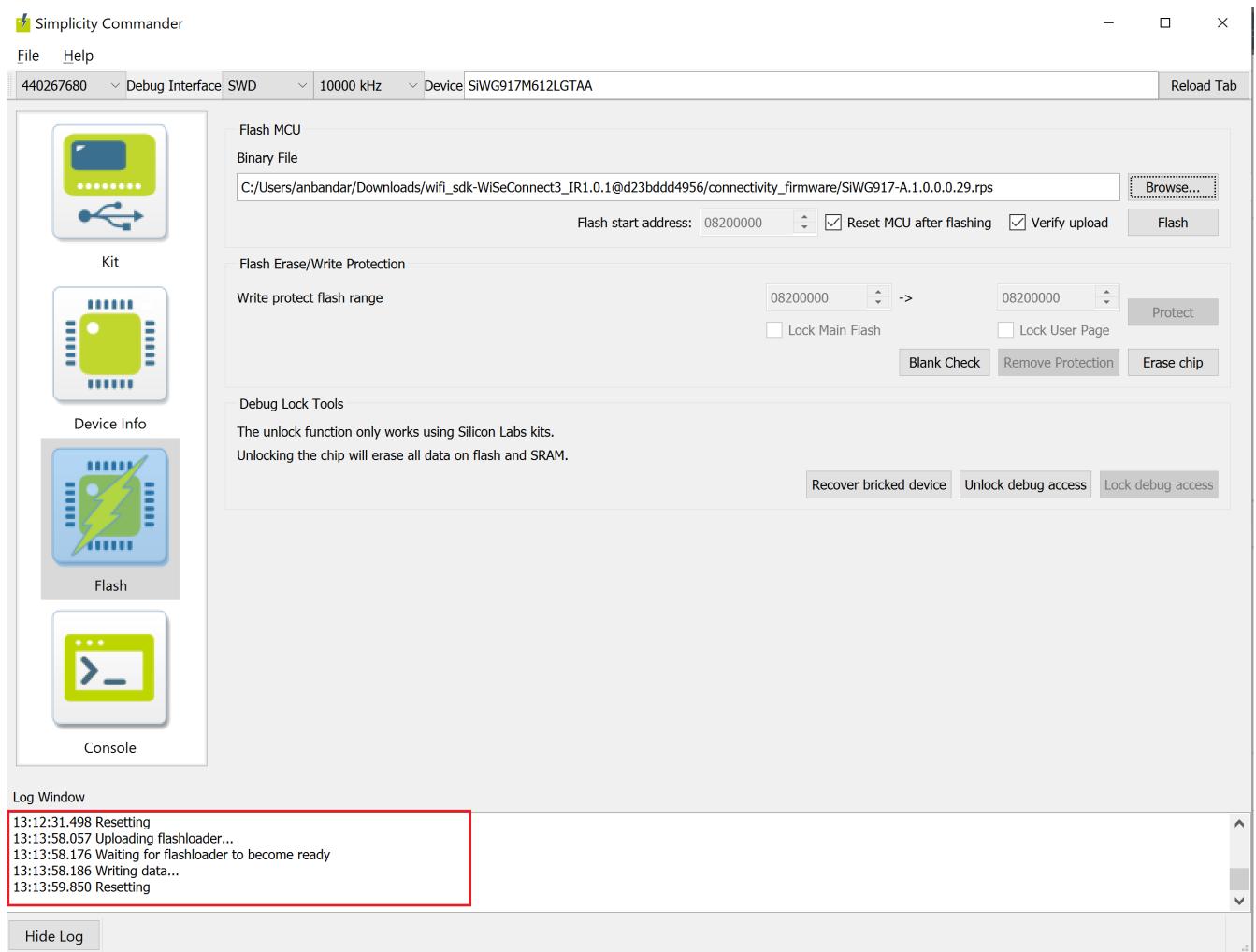


6. Refer to [Firmware for SiWx917 SoC](#) to identify the correct firmware to be flashed into the specific hardware. Locate and select the firmware file to flash.

7. Click Flash



8. The firmware will be flashed and the Log Window will display a "Resetting" message.



Troubleshoot SiWx917 SOC Firmware Update Failure

If the firmware update fails, try the following:

- Toggle the power switch towards AEM (Advanced Energy Monitoring) on the WSTK board.
- Perform the following steps and try the firmware update again
 - Press the RESET button on the WSTK board.
 - Retry the firmware upgrade.

Flash Bootloader

Flashing the Matter Binaries Using Simplicity Commander

To flash the application for EFR32 and SiWx917 SOC Board Simplicity Commander software will be used.

Before flashing the application for EFR32 Boards, flash **bootloader images** as per board variants:

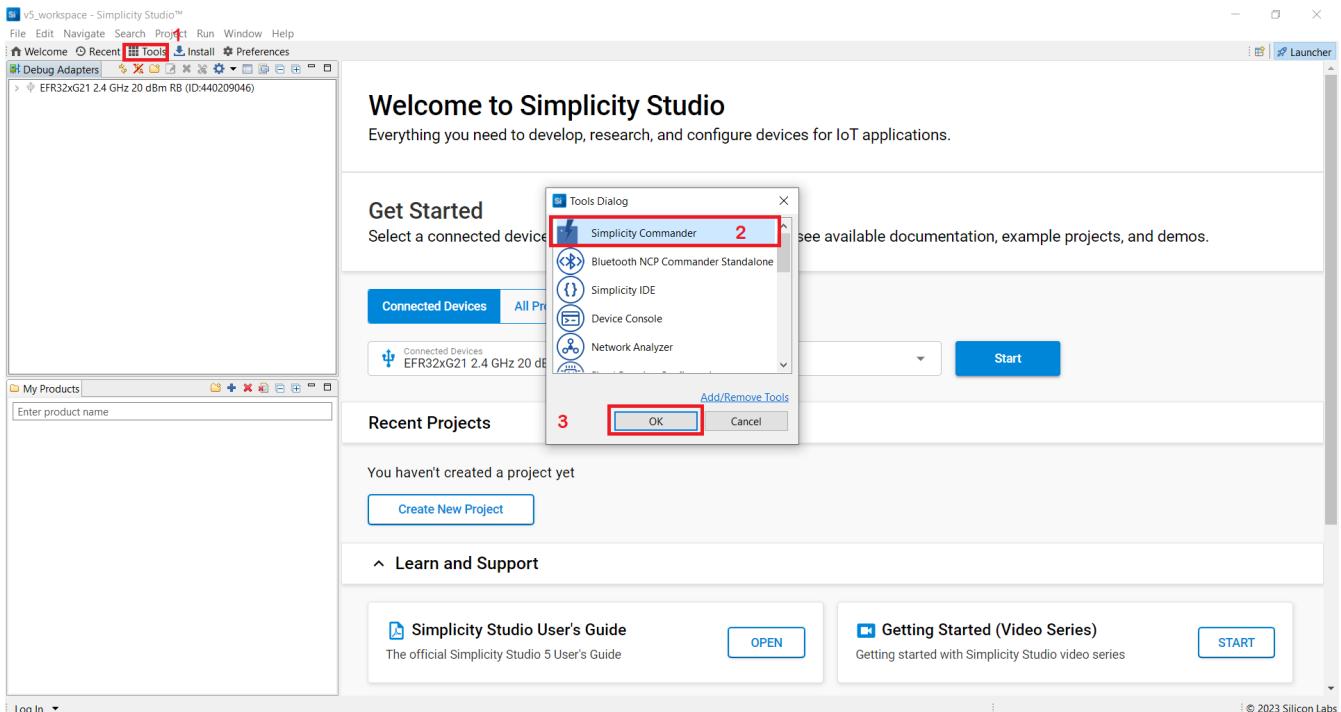
- **BRD4186C Board**
 - For MG24 + RS9116 :- Internal Bootloader (bootloader-storage-internal-single-512k-BRD4186C-gsdk4.1)
 - For MG24 + WF200 :- External Bootloader (bootloader-storage-spiflash-single-1024k-BRD4186C-gsdk4.1)
- **BRD4187C Board**
 - For MG24 + RS9116 :- Internal Bootloader (bootloader-storage-internal-single-512k-BRD4187C-gsdk4.1)
 - For MG24 + WF200 :- External Bootloader (bootloader-storage-spiflash-single-1024k-BRD4187C-gsdk4.1)

Bootloader binaries are available in the respective path of codebase `third_party/silabs/matter_support/matter/efr32/bootloader_binaries` folder. Silicon Labs recommends always flashing the latest bootloader binaries from the codebase.

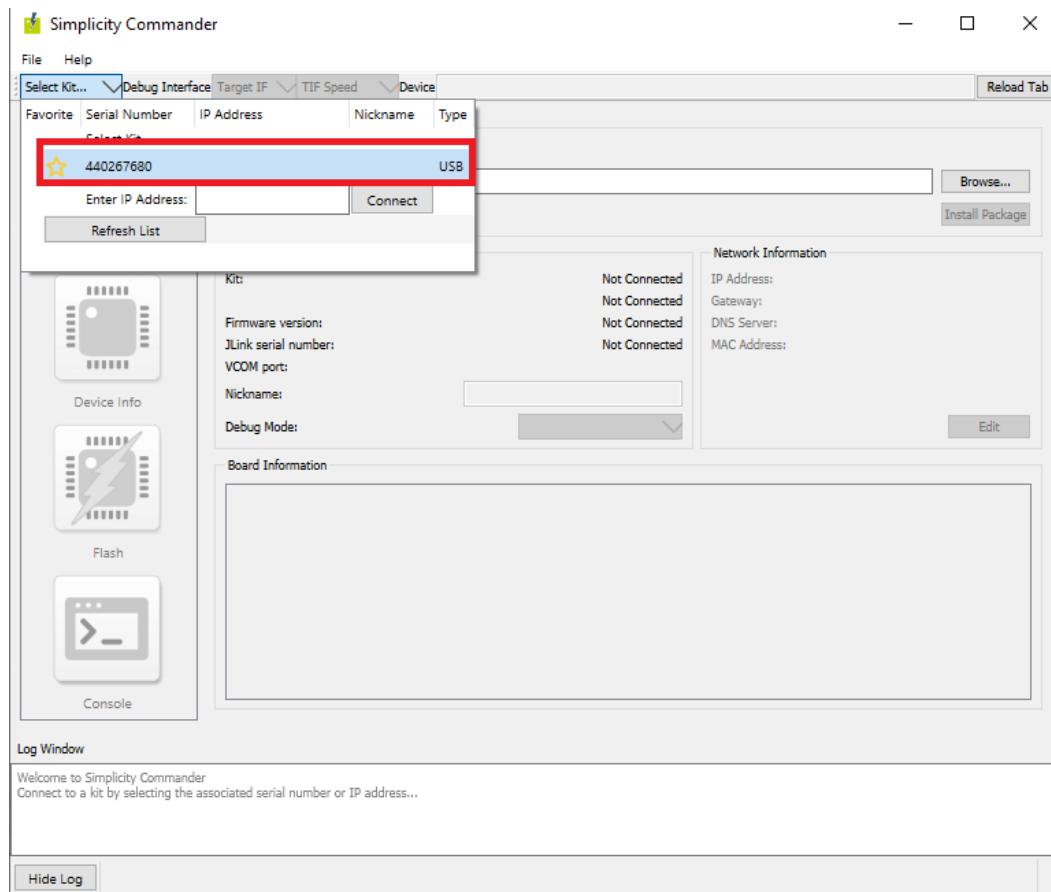
Note: Bootloader binaries are flashed using Simplicity Commander only. It supports EFR32 Boards only.

Flashing the Bootloader Binaries for EFR32 Board using Simplicity Commander

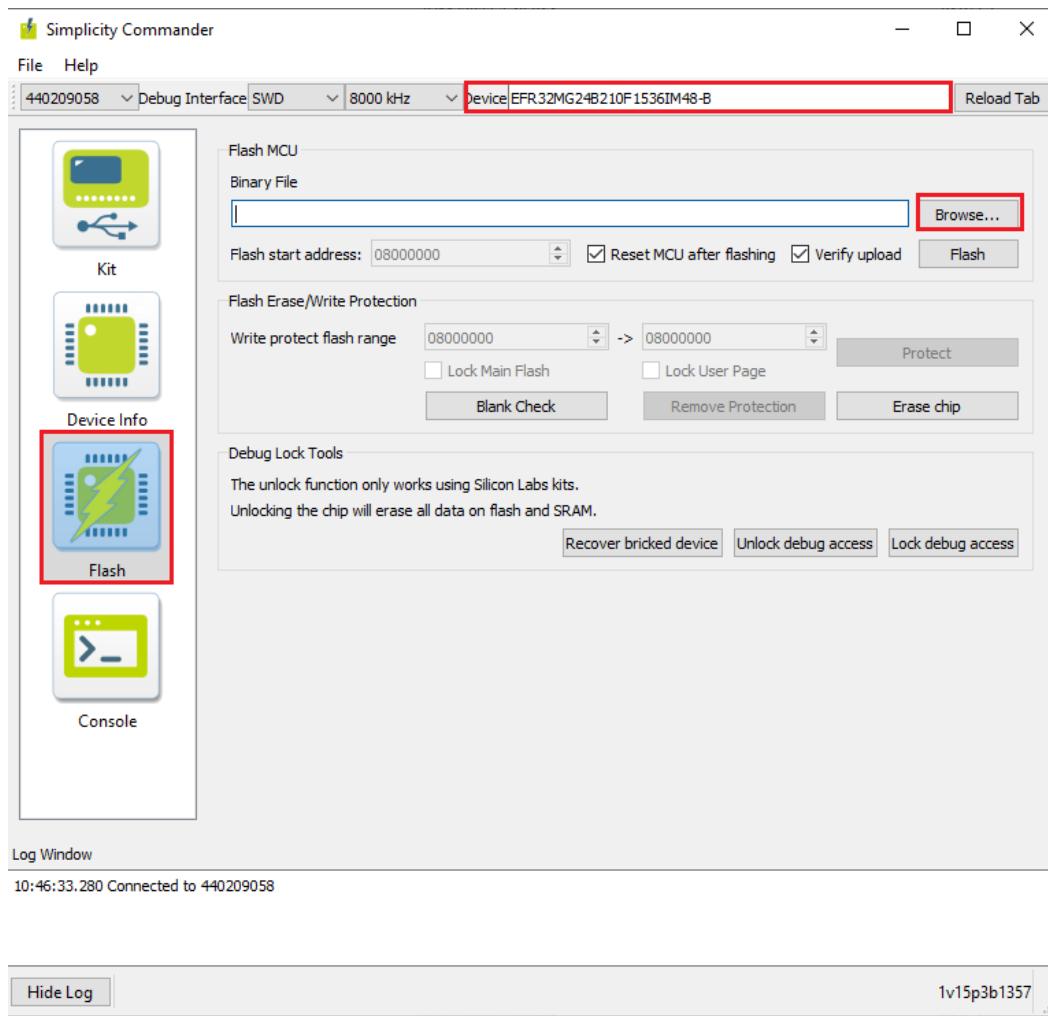
1. In the Simplicity Studio home page, click Tools.
2. In the Tools dialog, select Simplicity Commander and click OK.



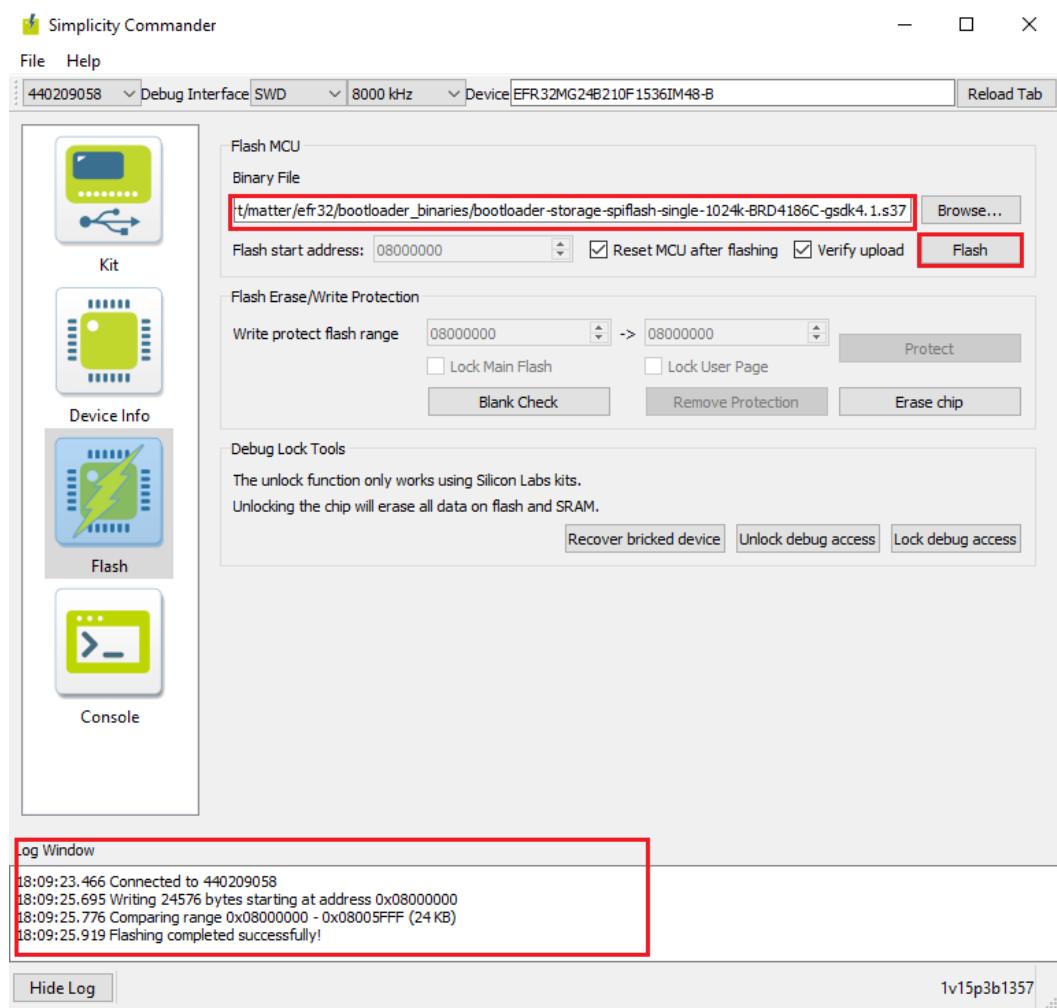
3. In the Simplicity Commander window, click **Select Kit** and choose your radio board.



4. In the navigation pane, go to the Flash section.
5. Above beside "Reload tab" board will be displayed, click Browse... next to the Binary File field and locate bootloader binary.

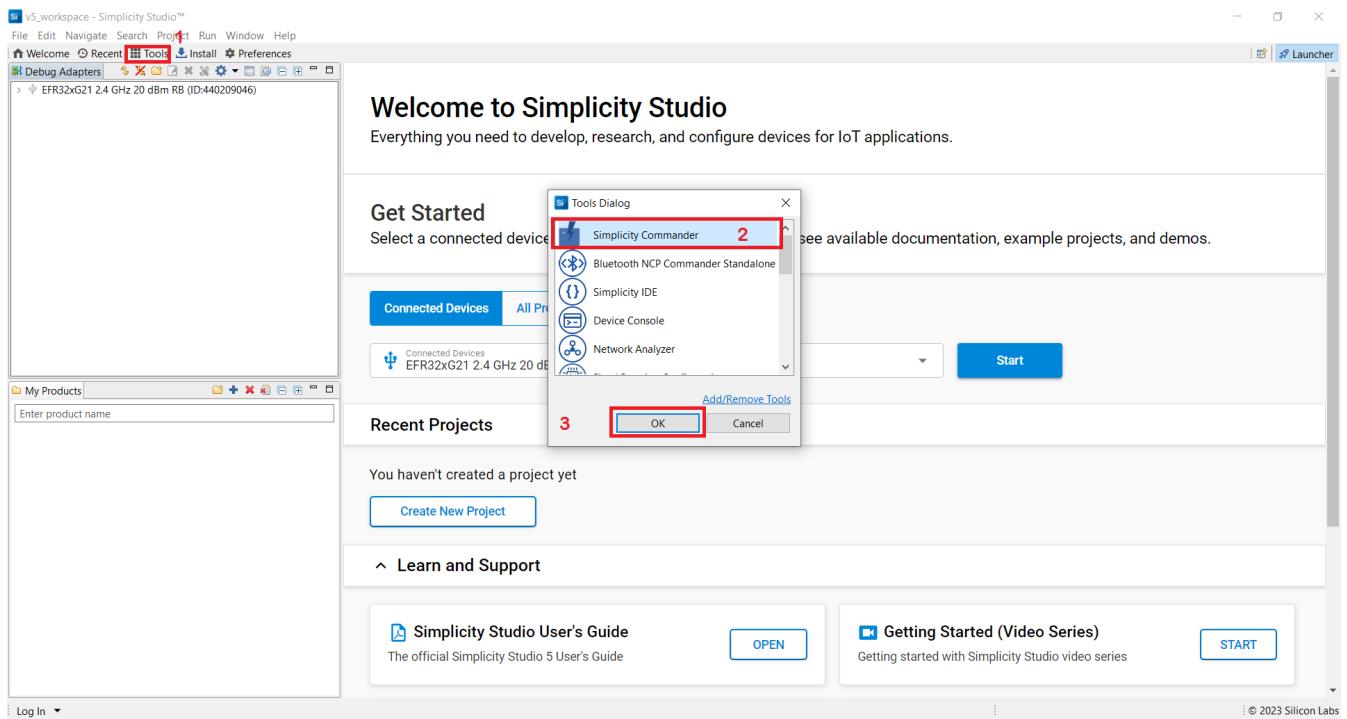


6. Click Flash, the bootloader will be flashed and the Log Window will display a "Flashing completed Successfully" message.

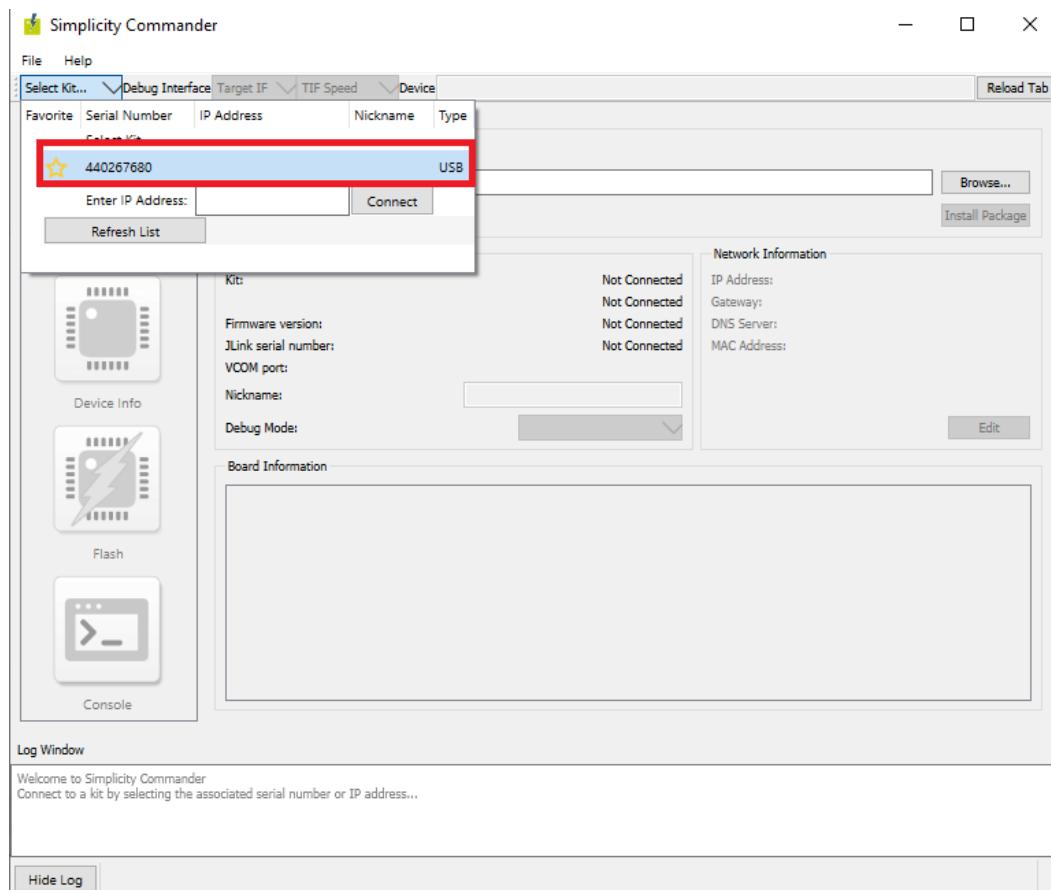


Flashing the EFR32 Matter Binary using Simplicity Commander

1. In the Simplicity Studio home page, click **Tools**.
2. In the **Tools** dialog, select **Simplicity Commander** and click **OK**.

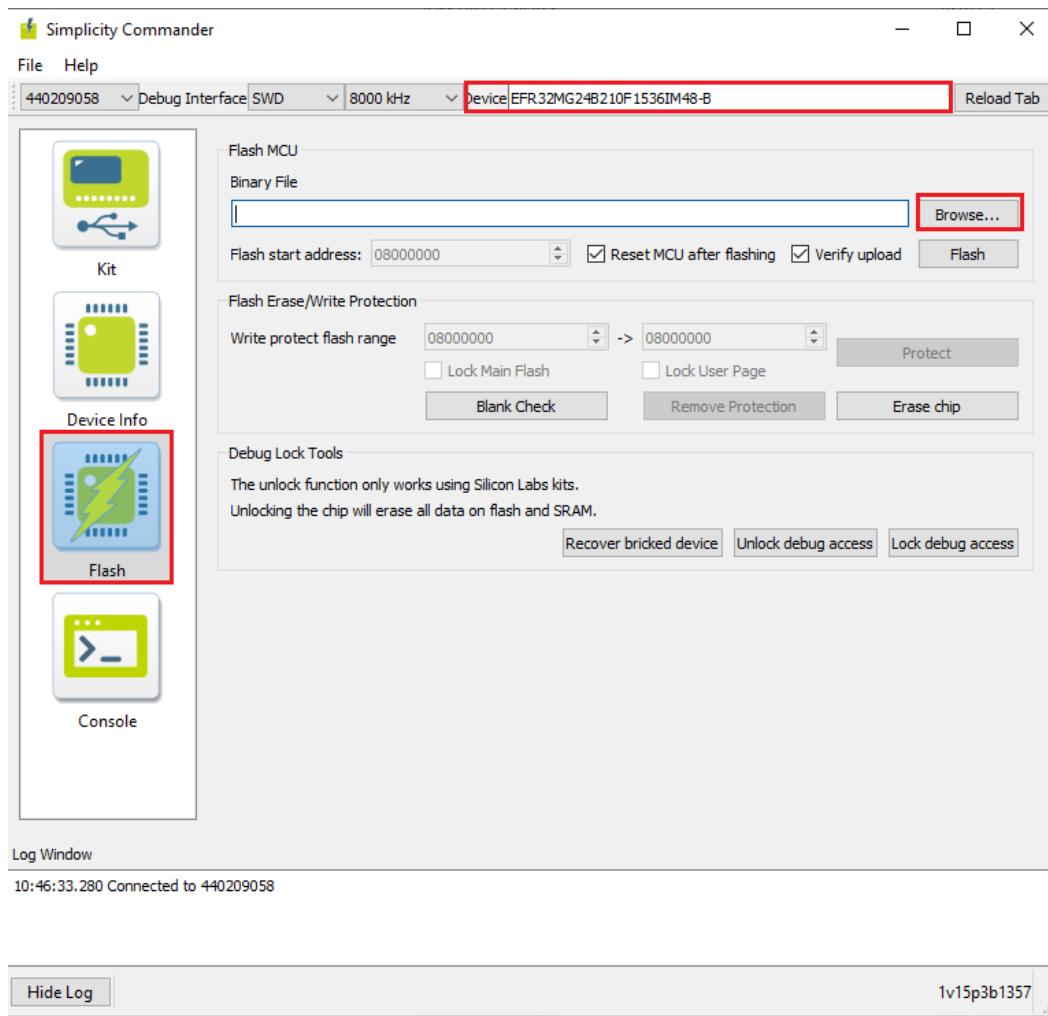


3. In the Simplicity Commander window, click Select Kit and choose your radio board.

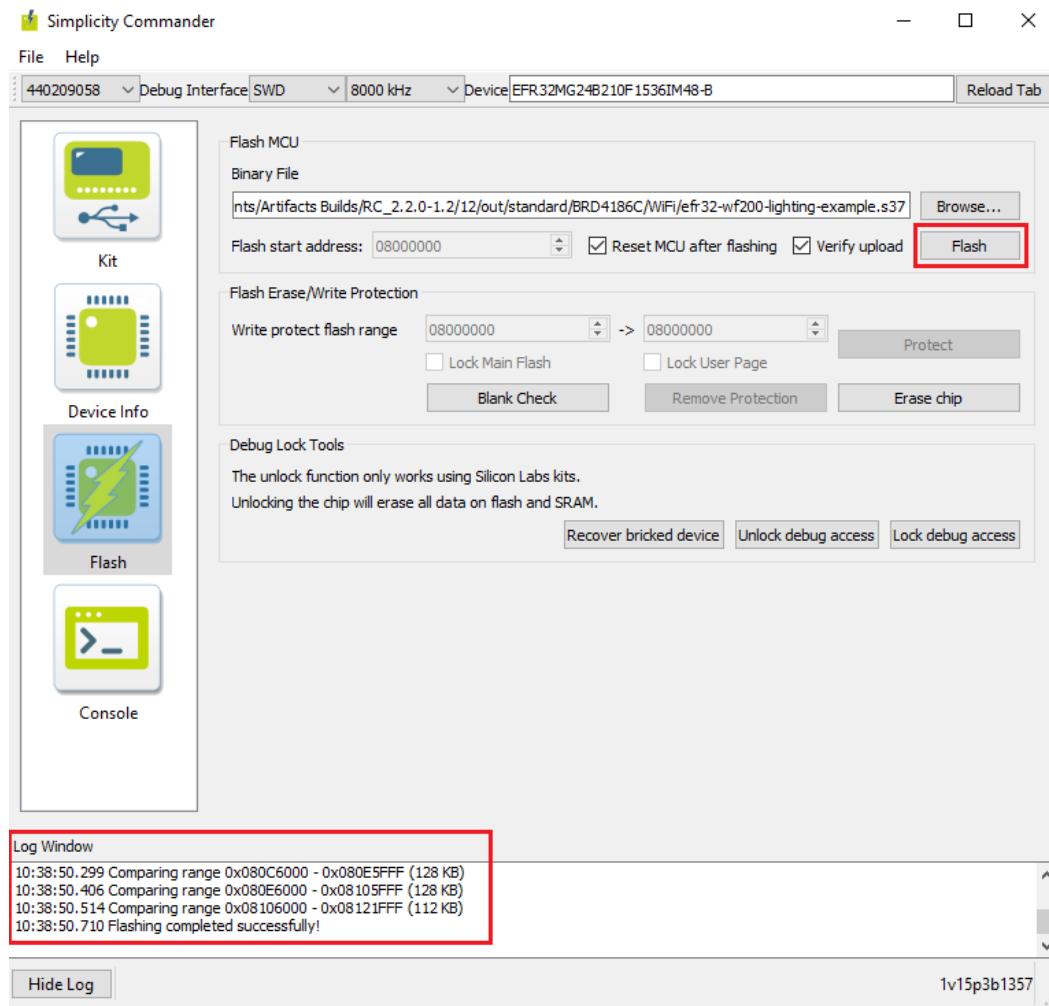


4. In the navigation pane, go to the **Flash** section.

5. Your board will be displayed. Click **Browse** next to the **Binary File** field and locate the binary.



6. Click **Flash**. The binary will be flashed and the Log Window will display a "Flashing completed Successfully" message.



Flashing the SiWx917 SOC Matter Binary using Simplicity Commander

SiWx917 SoC device support is available in the latest [Simplicity Commander](#). The SiWx917 SOC board will support .rps only file to flash. Follow these steps to create and flash .rps file using .s37.

1. Locate Simplicity Commander in your PC/Laptop where it is installed through command prompt(cmd).

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3603]
(c) Microsoft Corporation. All Rights Reserved.

C:\Users\Kirankha\Downloads\SimplicityCommander-Microsoft\SimplicityCommander-Microsoft\Comender_min32_x64_1v16p1b1501\Simplicity_Commander>
```

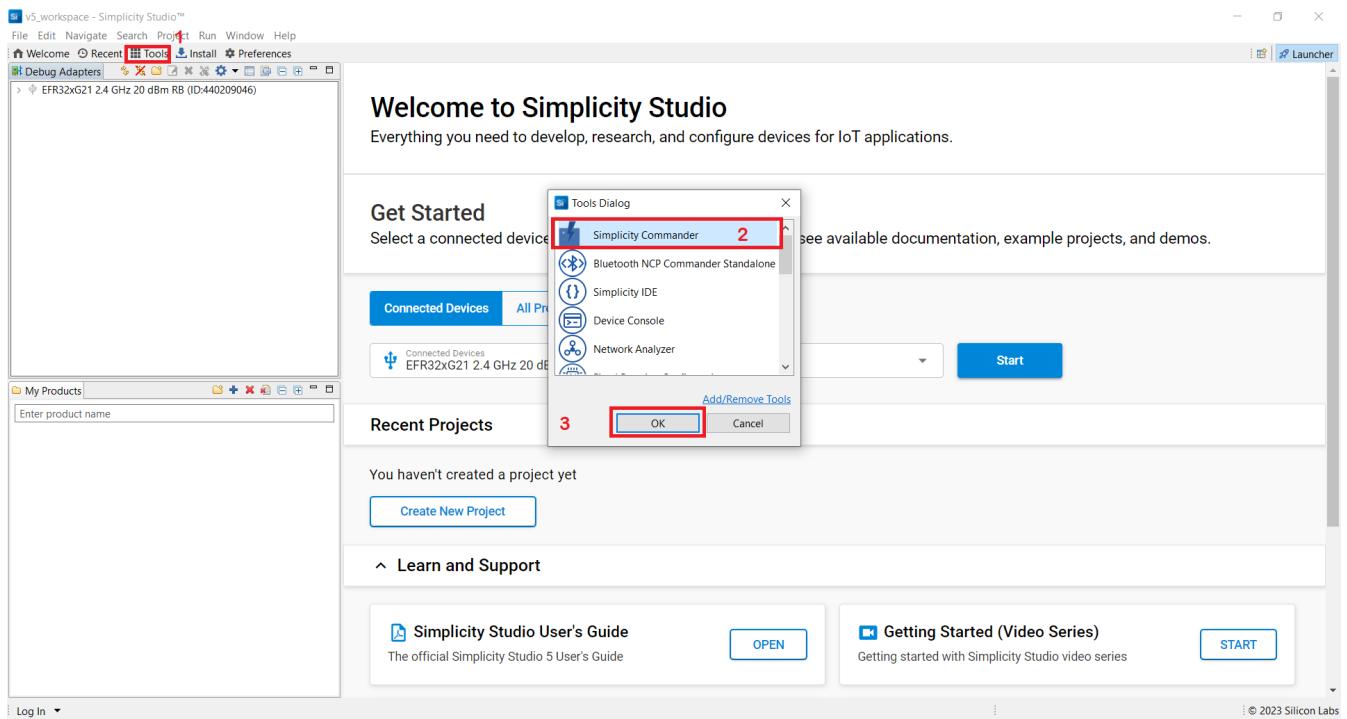
2. Copy and paste the built .s37 binary file to the Simplicity commander path.
3. Convert .s37 binary to .rps using the command below using commander terminal.

```
comander rps convert <file_name.rps> --app <file_name.s37>
```

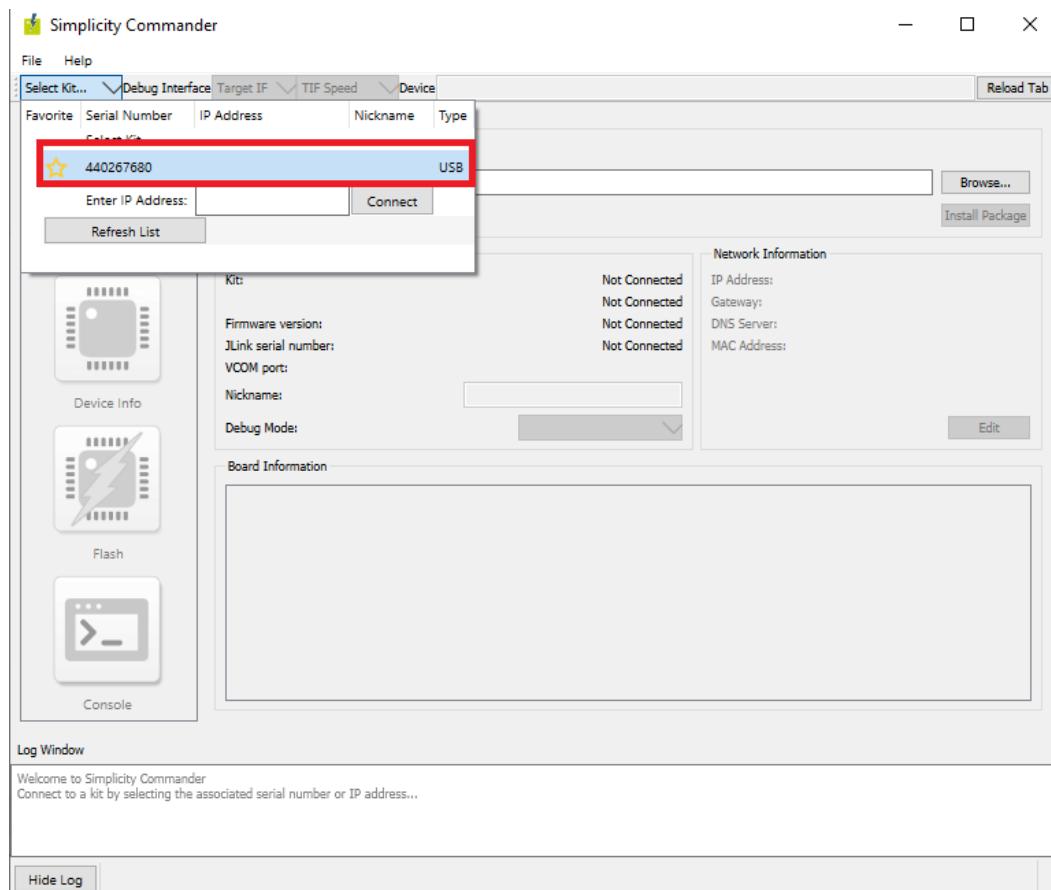
4. Flash to the device using command or follow the next steps to flash through Commander Software.

```
comander rps load <file-name>.rps
```

5. In the Simplicity Studio home page, click **Tools**.
6. In the Tools dialog, select Simplicity Commander and click OK.

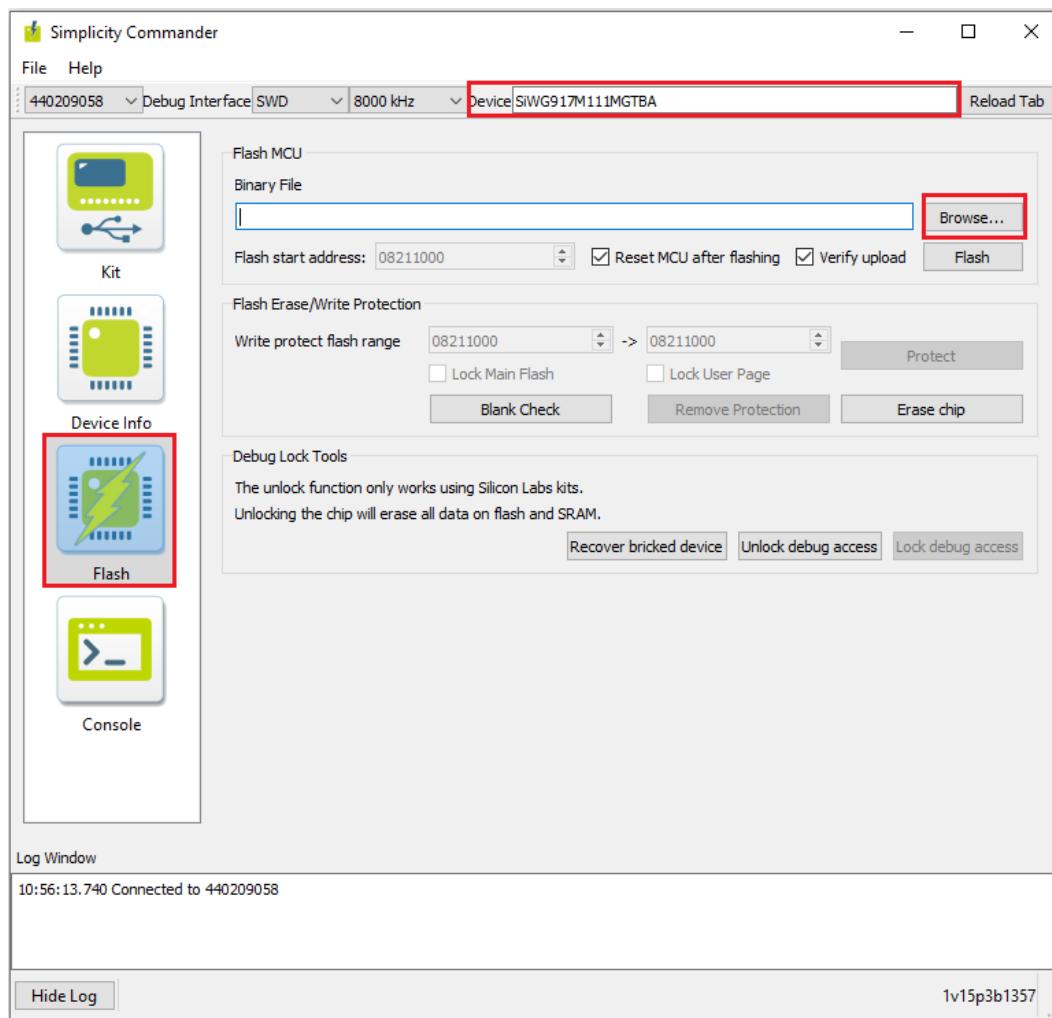


7. In the Simplicity Commander window, click **Select Kit** and choose your radio board.

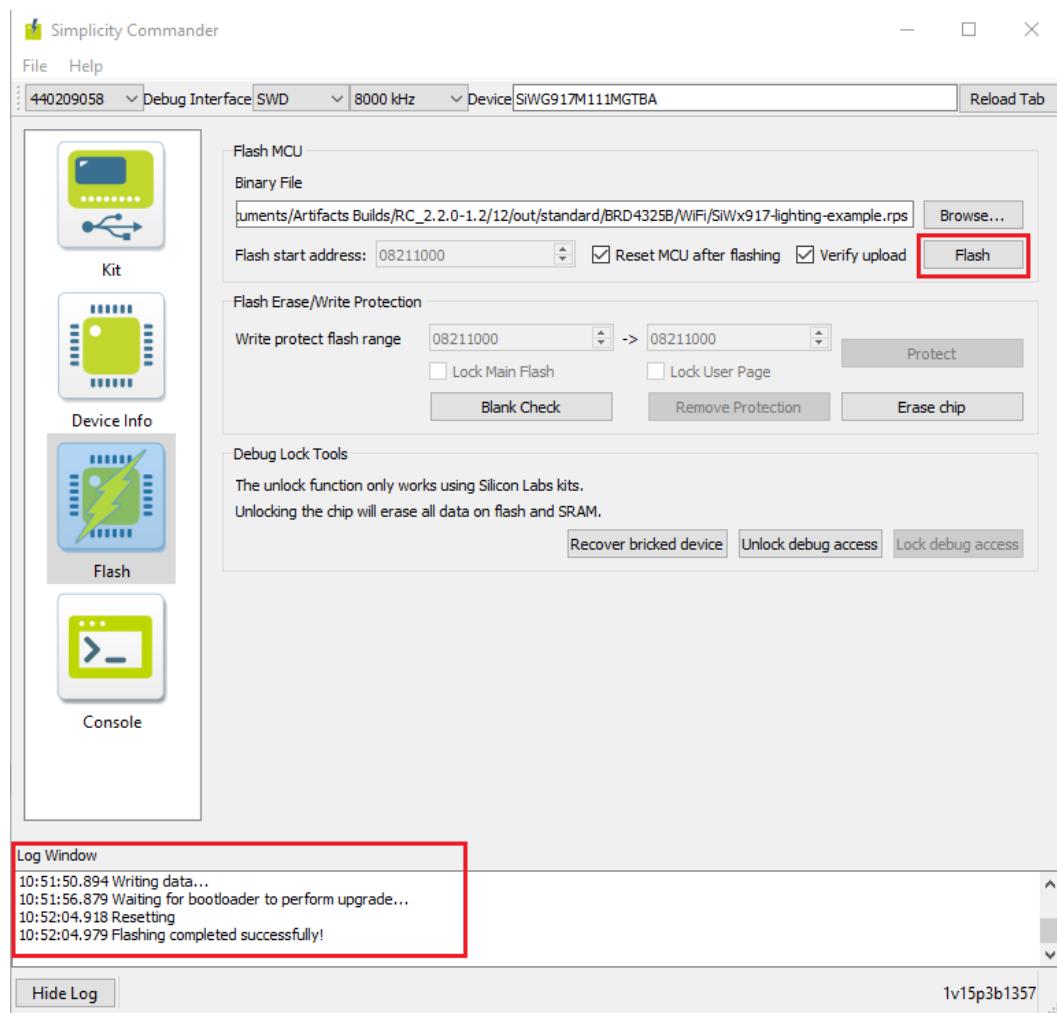


8. In the navigation pane, go to the **Flash** section.

9. Above beside "Reload tab" board will be displayed, click **Browse** next to the **Binary File** field and locate binary.



10. Click **Flash**. The binary will be flashed and the Log Window will display a "Flashing completed Successfully" message.

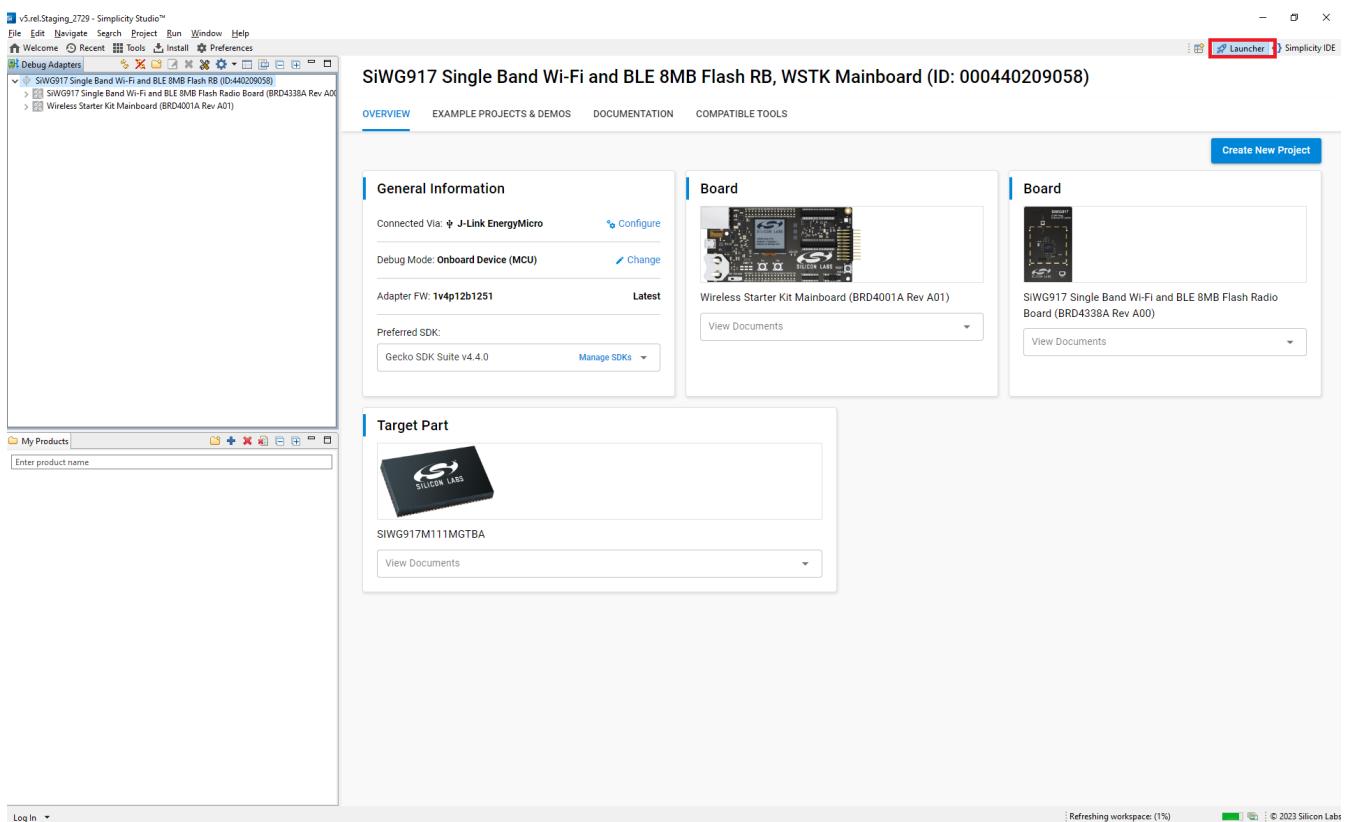


Build an SoC Application Using Studio

Building the 917 SoC Matter Accessory Devices (MADs) using Simplicity Studio

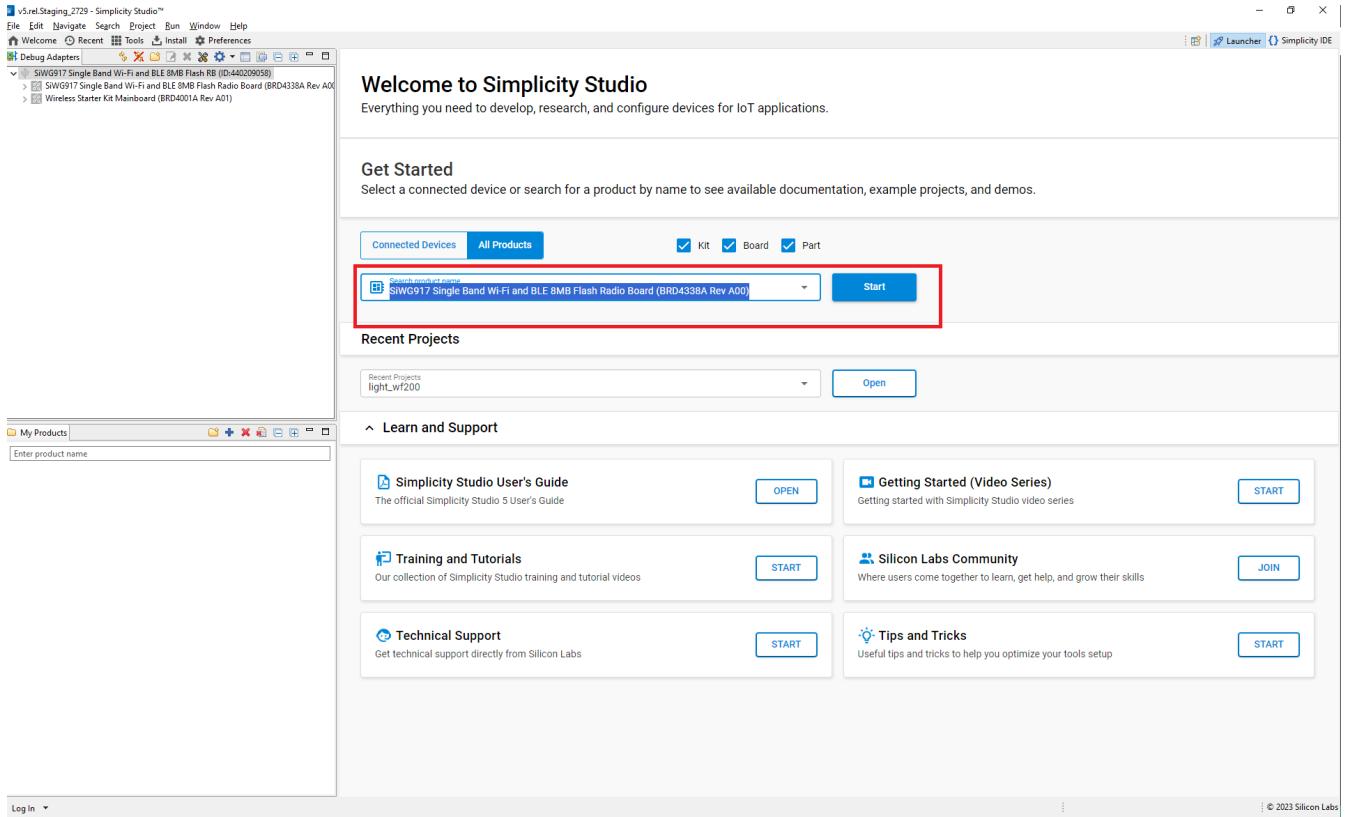
In Simplicity Studio 5, create the Light MAD:

1. [Download](#) and Install Simplicity Studio 5.
2. To install the software packages for Simplicity Studio, refer to the [Software Package Installation section](#).
3. Switch to the Launcher view (if not already in it).

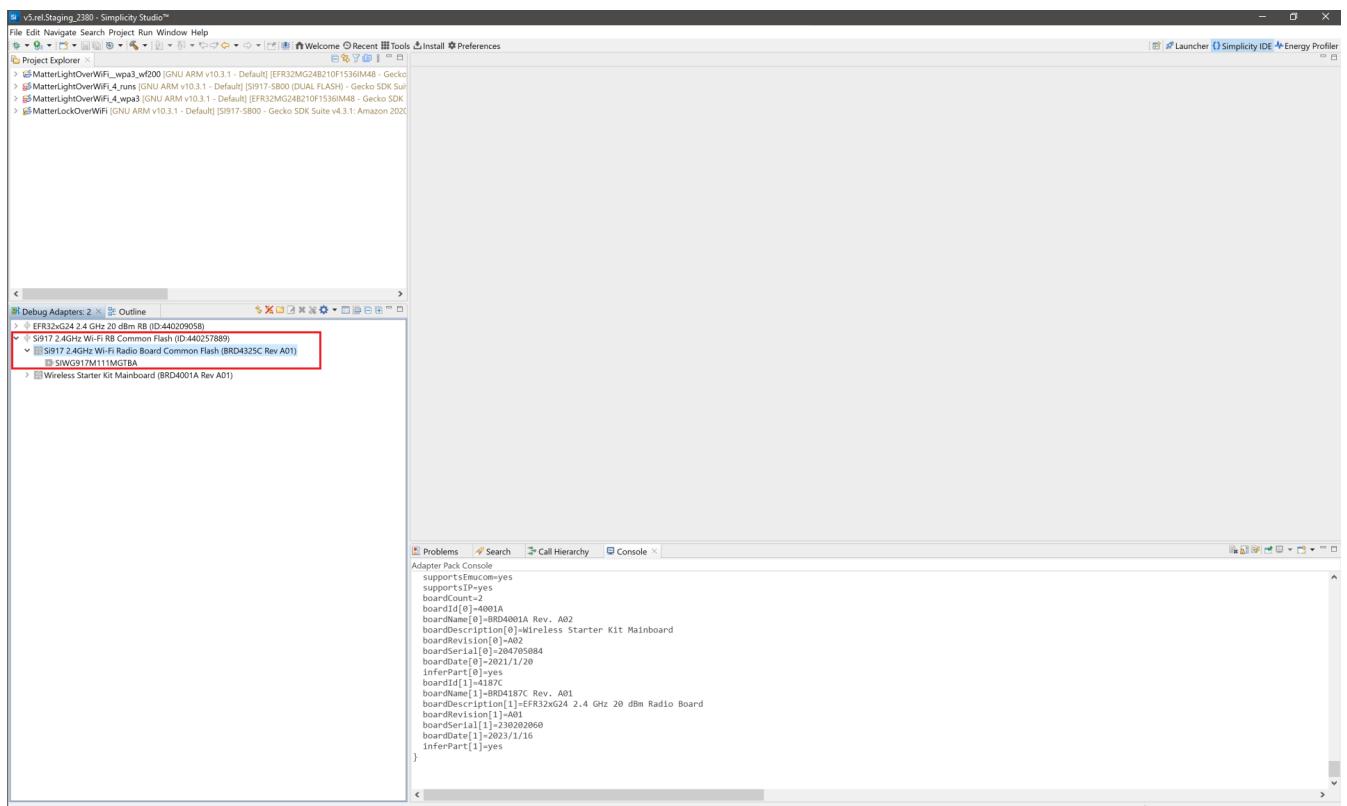


4. Go to All Products in the **Launcher** tab and select a compatible board from the supported SiWx917 SOC dev boards.

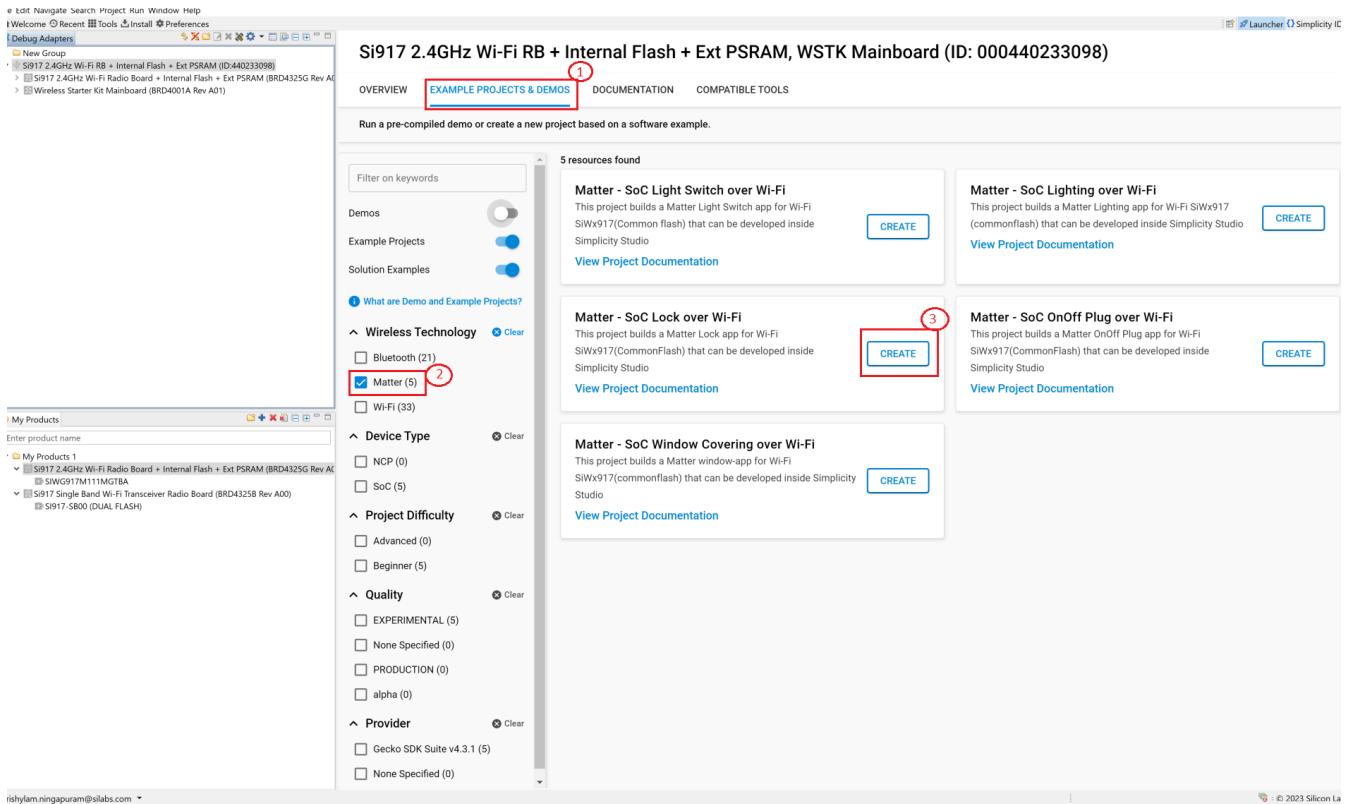
BRD4338A (Common Flash)



5. Once it shows up in the **Debug Adapters** view, select it.



6. Open the Example Projects and Demos tab, select the Matter filter and enter **Wi-Fi** in Filter on keywords and click **CREATE**.



Si917 2.4GHz Wi-Fi RB + Internal Flash + Ext PSRAM, WSTK Mainboard (ID: 000440233098)

OVERVIEW EXAMPLE PROJECTS & DEMOS DOCUMENTATION COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos Example Projects Solution Examples

What are Demo and Example Projects?

Wireless Technology

- Bluetooth (21)
- Matter (5) 3
- Wi-Fi (33)

Device Type

- NCP (0)
- SoC (5)

Project Difficulty

- Advanced (0)
- Beginner (5)

Quality

- EXPERIMENTAL (5)
- None Specified (0)
- PRODUCTION (0)
- alpha (0)

Provider

- Gecko SDK Suite v4.3.1 (5)
- None Specified (0)

5 resources found

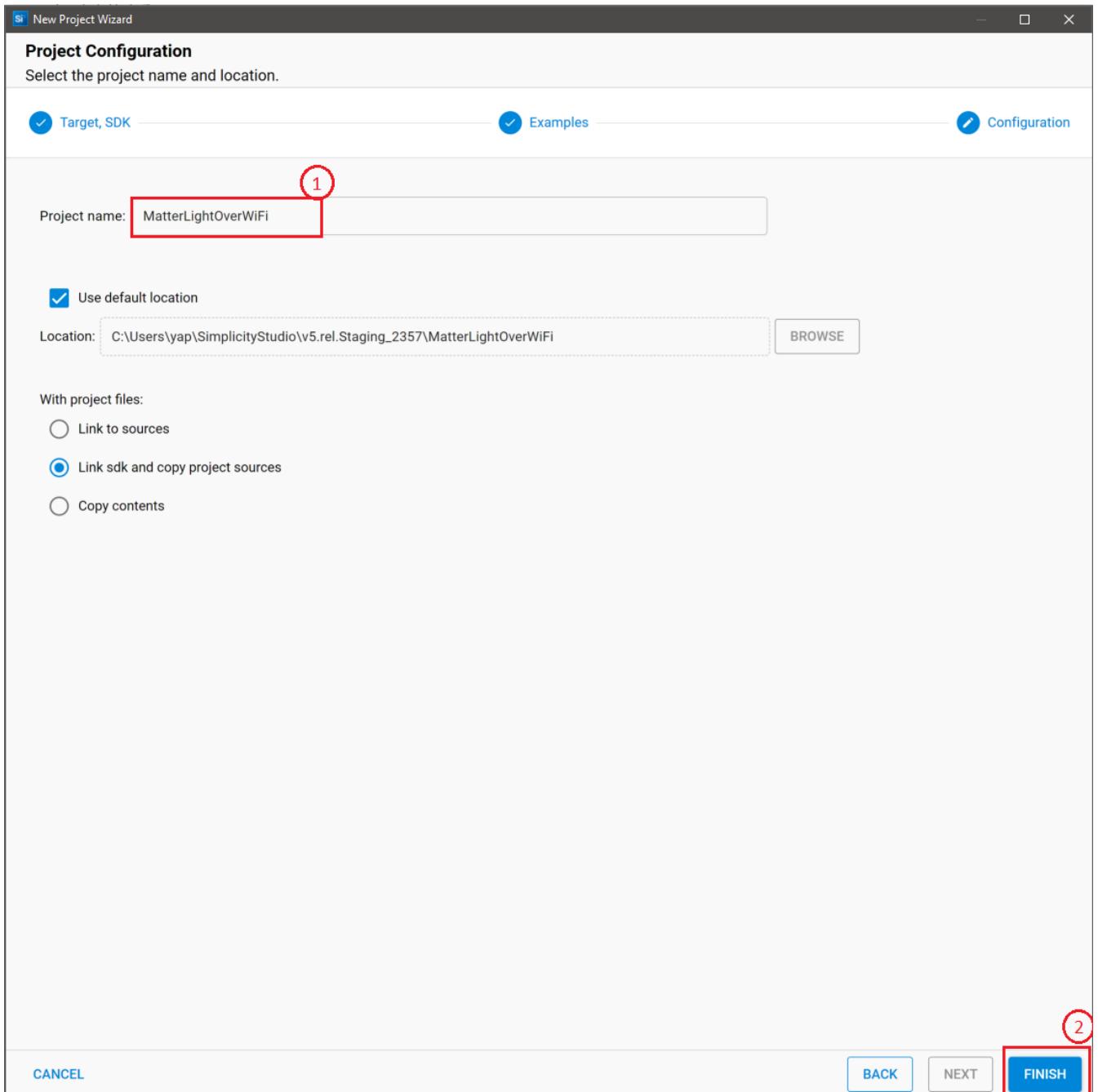
Matter - SoC Light Switch over Wi-Fi
This project builds a Matter Light Switch app for Wi-Fi SiWx917(Common flash) that can be developed inside Simplicity Studio
[View Project Documentation](#) CREATE

Matter - SoC Lock over Wi-Fi
This project builds a Matter Lock app for Wi-Fi SiWx917(CommonFlash) that can be developed inside Simplicity Studio
[View Project Documentation](#) CREATE

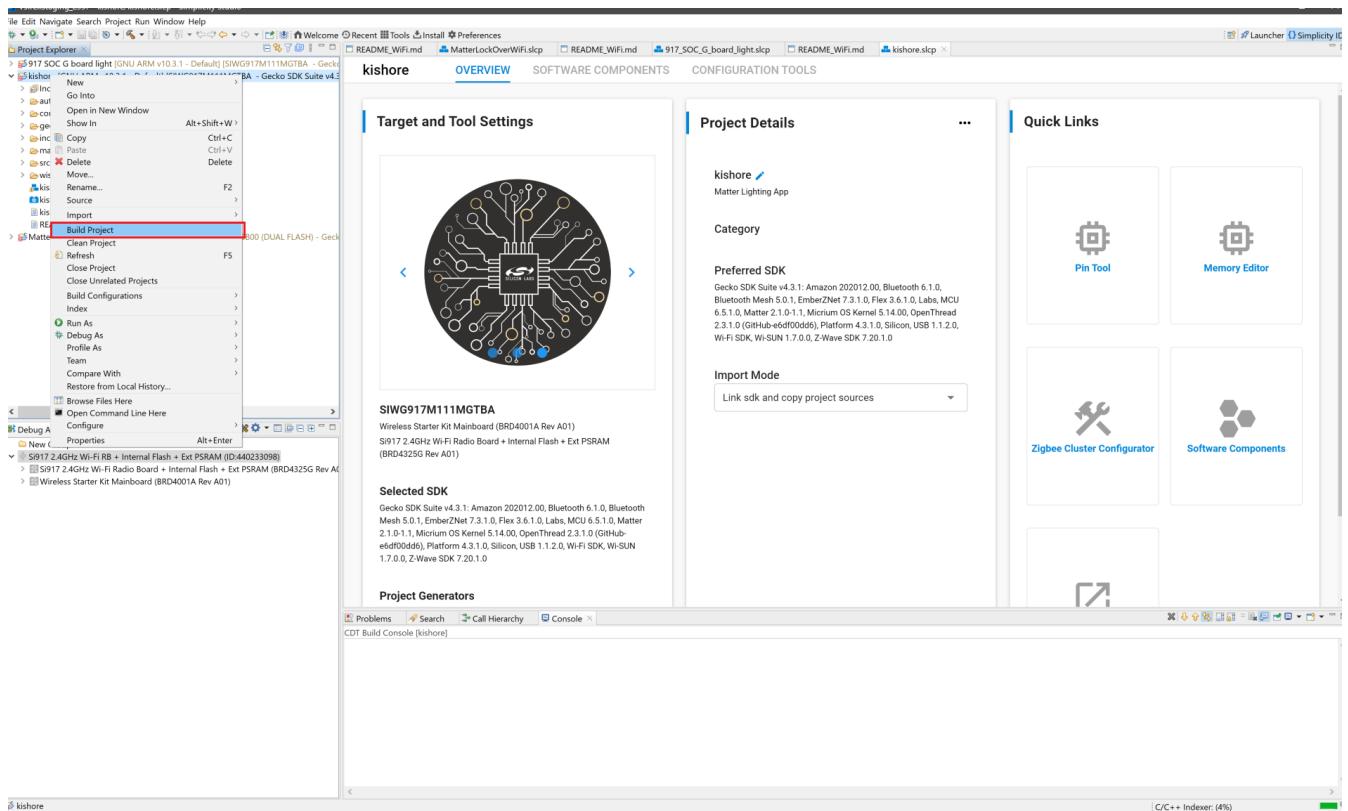
Matter - SoC Window Covering over Wi-Fi
This project builds a Matter window-app for Wi-Fi SiWx917(commonflash) that can be developed inside Simplicity Studio
[View Project Documentation](#) CREATE

© 2023 Silicon Lab

7. Rename the Project Name if you wish, and click **Finish**.

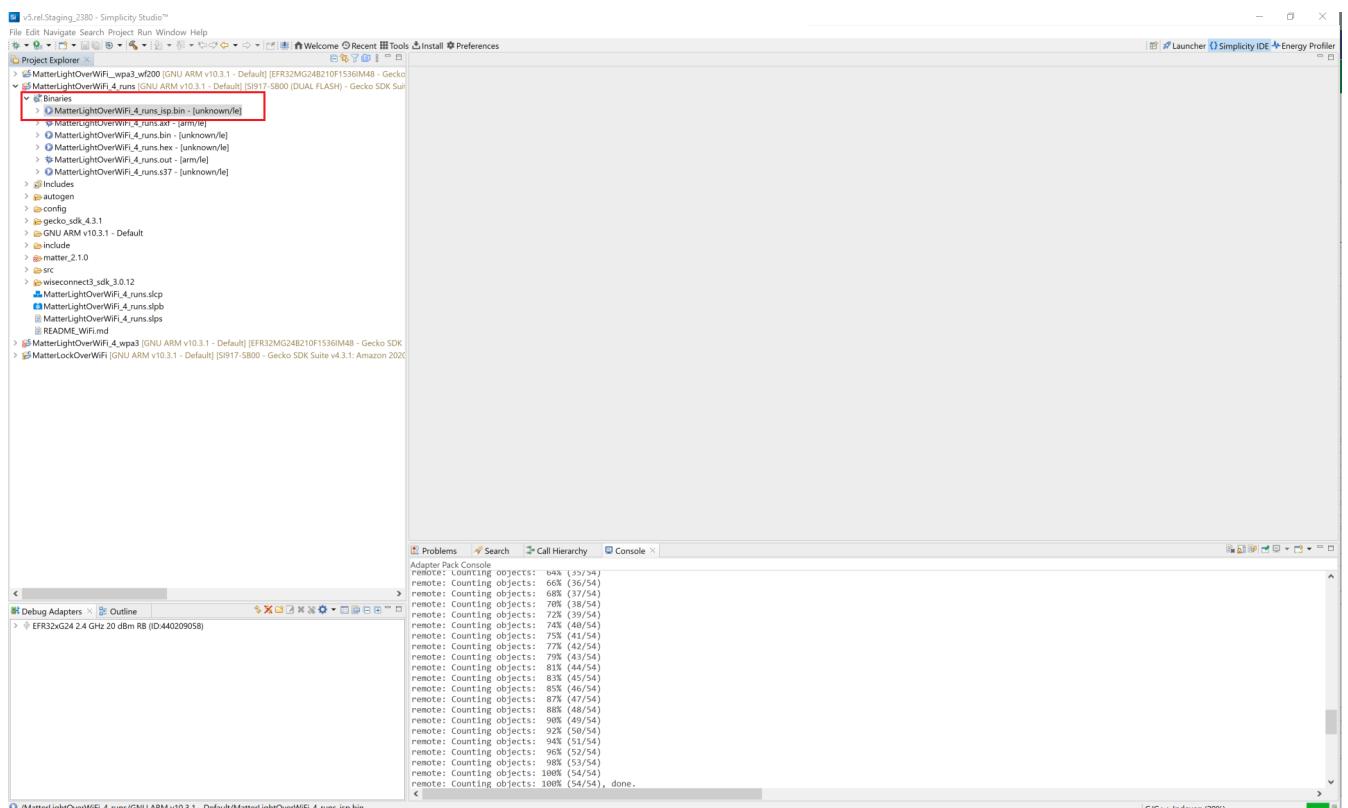


8. Once the project is created, right-click on the project and select **Build Project** in the **Project Explorer** tab.



9. To flash the application, connect the compatible dev board to the machine or PC if not yet done.

10. Once the project is compiled successfully, go to the Project Explorer view and select the binary to be flashed.



Build an SoC Application Using Studio



The screenshot shows the Simplicity Studio interface for building a MatterLightOverWiFi application. The Project Explorer panel lists files and folders for the project. The Target and Tool Settings panel shows a circular schematic of the SiWX917 SoC. The Project Details panel shows the project name as MatterLightOverWiFi and its category as Matter Lighting App. The Quick Links panel has links for Software Components and Pin Tool. The bottom console window displays the build log:

```
C:\Users\yap\Downloads\SimplicityStudio-5 (1)\SimplicityStudio_v5\developer\adapter_packs\commander\commander.exe" postbuild "C:\Users\yap\SimplicityStudio\v5.rel.Staging_2357\MatterLightOverWiFi\MatterLightOverWiFi.slpb...
Parsing file C:\Users\yap\SimplicityStudio\v5.rel.Staging_2357\MatterLightOverWiFi\MatterLightOverWiFi.slpb...
Running task RPS create...
Running task convert...
Running task convert...
Running task convert...
Running task convert...
DONE

15:07:25 Build Finished. 0 errors, 879 warnings. (took 13m:2s.943ms)
```

Note: SiWX917 SoC device will support both .isp.bin and .rps file format to flash.

12. The Flash programmer window will open. Click the Program button to start the flashing.

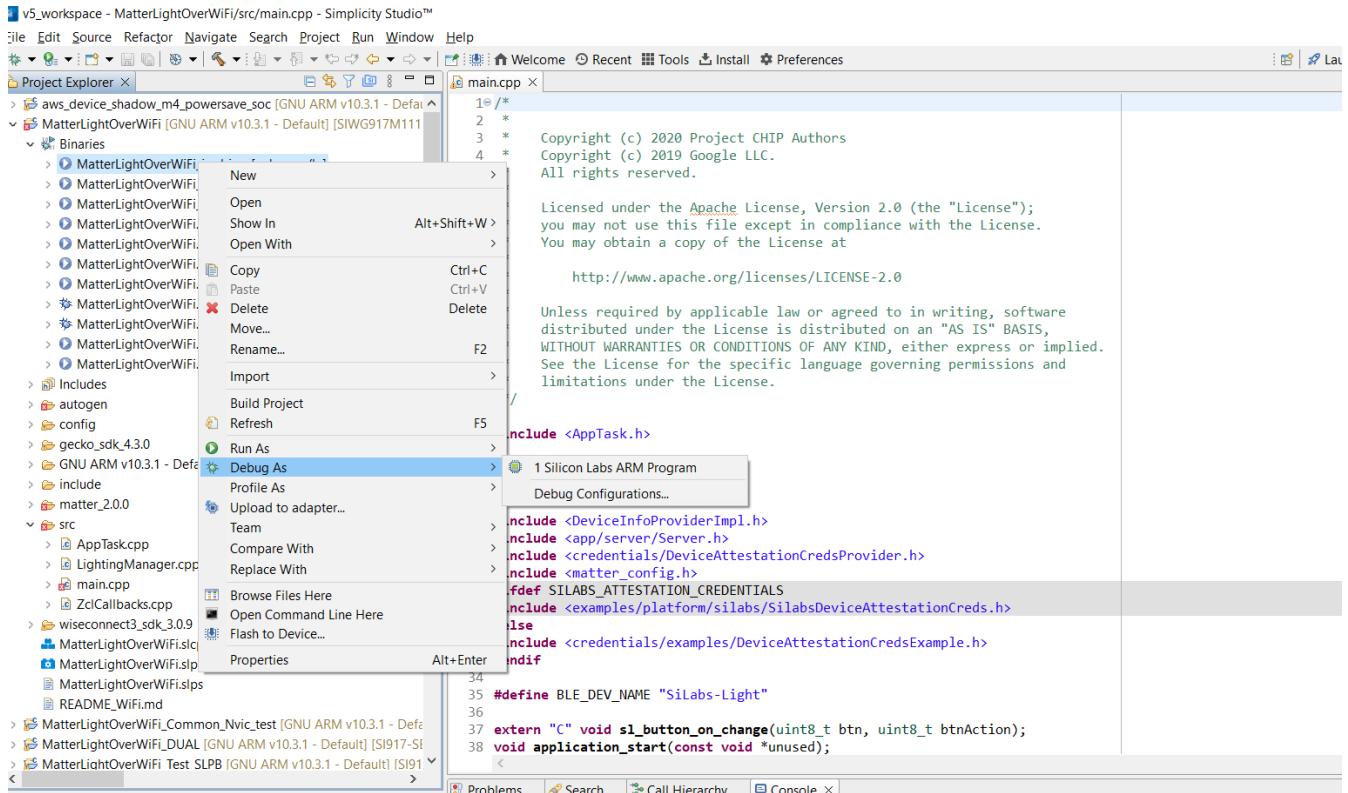
The screenshot shows the Simplicity Studio interface with the Flash Programmer dialog open. The dialog shows the target board as Wireless Starter Kit Mainboard (EFR32xG24 2.4 GHz 10 dBm Radio Board) and the adapter as J-Link Silicon Labs (44025163). The File Type dropdown is set to hex. The File field contains the path to the .isp bin file. The Flash Erase/Write Protection section has the Select flash range option selected, with memory ranges 0x08000000 to 0x08180000. The Program button is highlighted in red. The bottom console window shows the build log:

```
"C:\Users\yap\Downloads\SimplicityStudio-5 (1)\SimplicityStudio_v5\developer\adapter_packs\commander\commander.exe" postbuild "C:\Users\yap\SimplicityStudio\v5.rel.Staging_2357\MatterLightOverWiFi\MatterLightOverWiFi.slpb...
Parsing file C:\Users\yap\SimplicityStudio\v5.rel.Staging_2357\MatterLightOverWiFi\MatterLightOverWiFi.slpb...
Running task RPS create...
Running task convert...
Running task convert...
Running task convert...
Running task convert...
DONE

15:07:25 Build Finished. 0 errors, 879 warnings.
```

Note: Output of the SiWX917 SoC application will be displayed on the J-Link RTT Viewer.

13. In order to debug Matter Application, right-click the selected `_isp.bin` binary and click on **Debug As**.

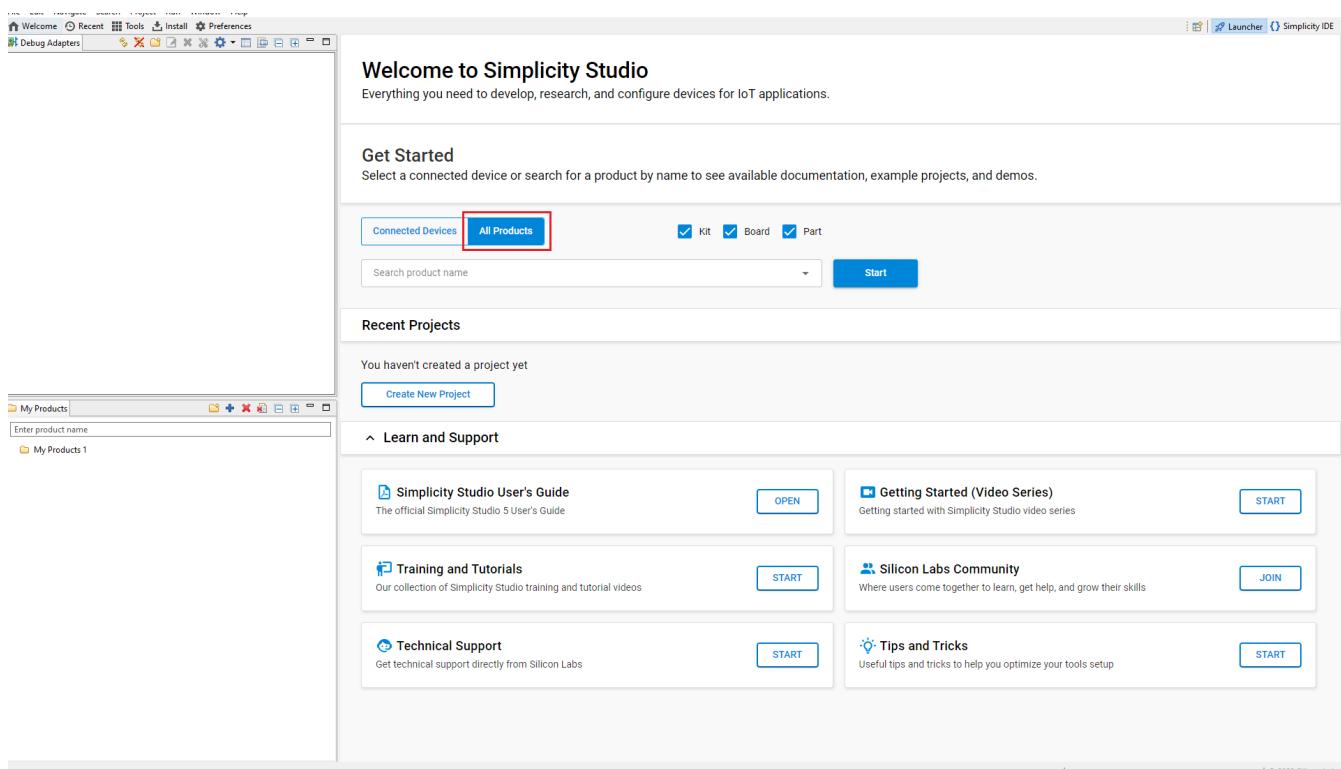


Build an NCP Application Using Studio

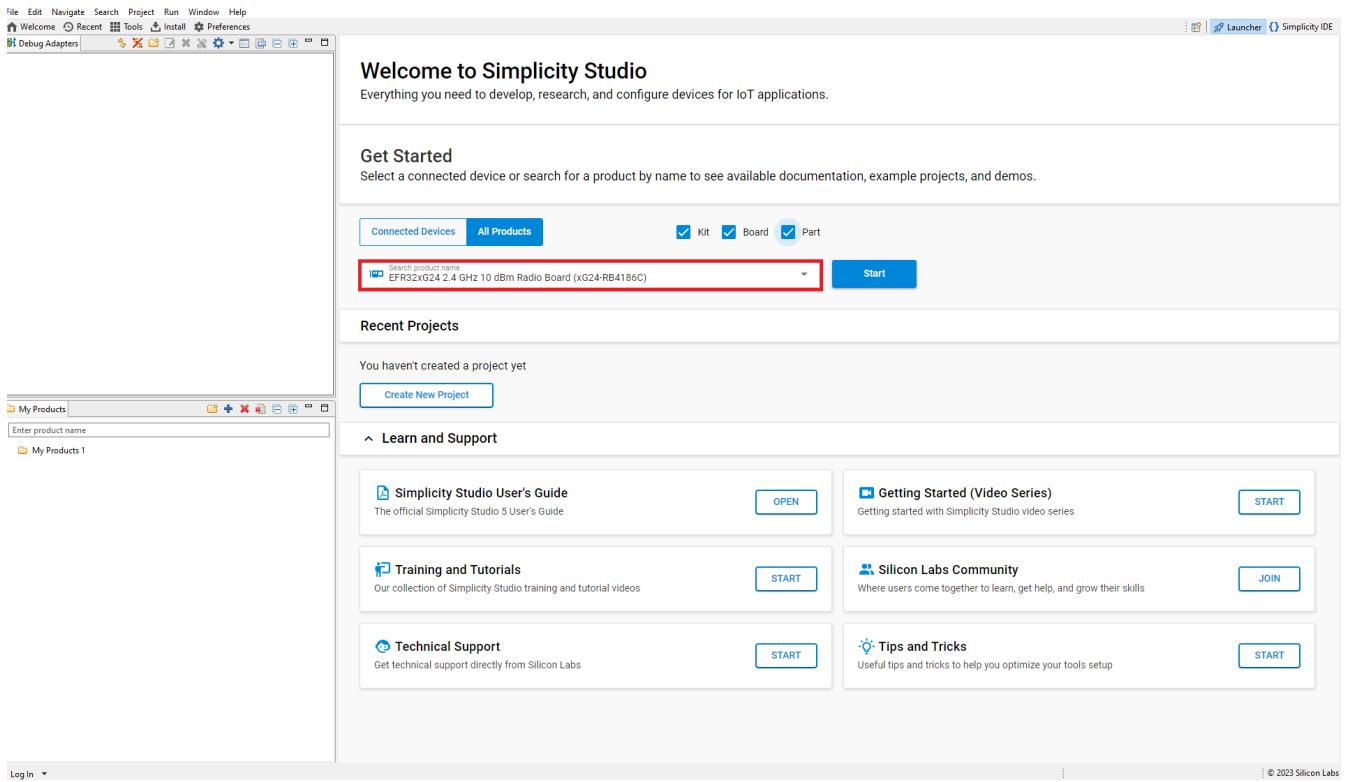
Create a Project for an EFR32 Application

This page provides a detailed description on how to create an Wi-Fi NCP project for EFR32 boards.

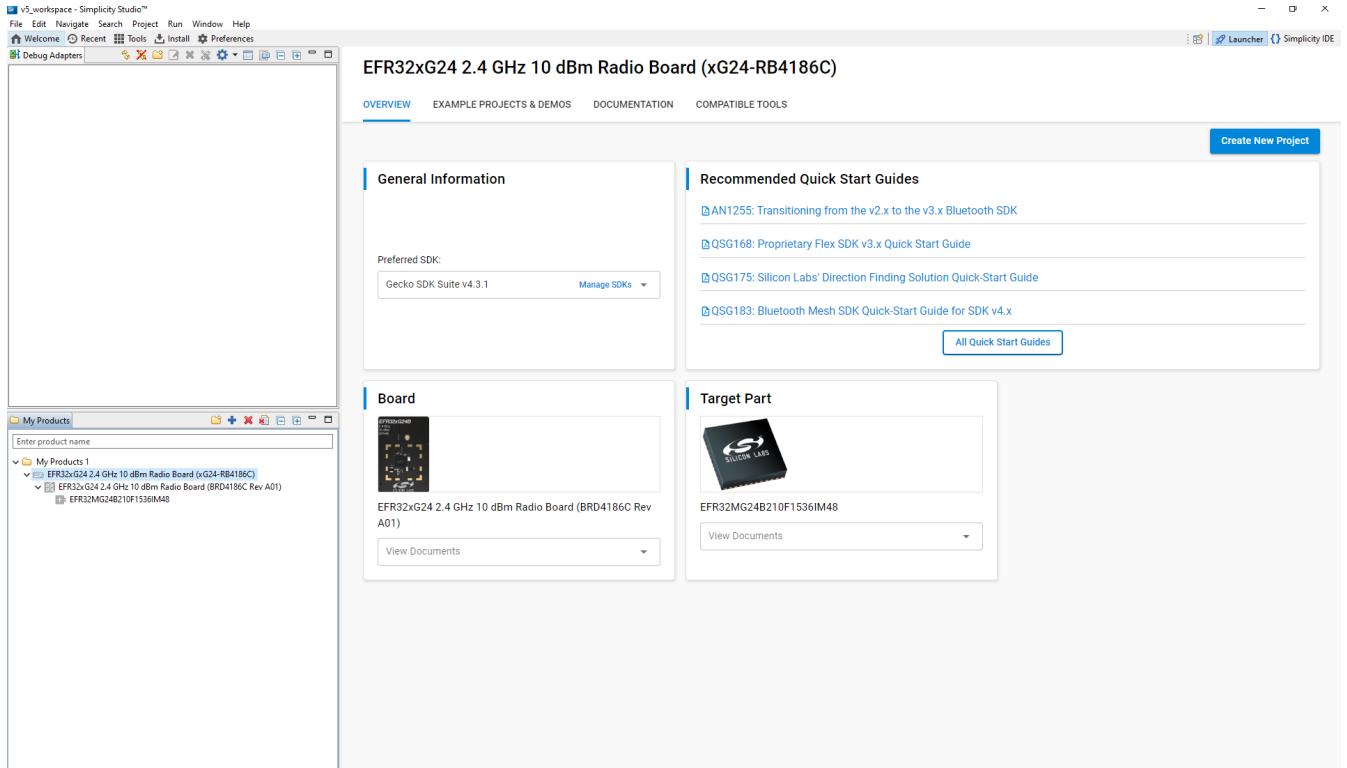
1. [Download](#) and Install Simplicity Studio.
2. To install the software packages for Simplicity Studio, refer to [Software Package Installation](#).
3. Log in to Simplicity Studio and connect the EFR32 WSTK board to the computer.
4. Go to the **All Products** section.



5. Search and select the radio board from the displayed list and select **Start**.

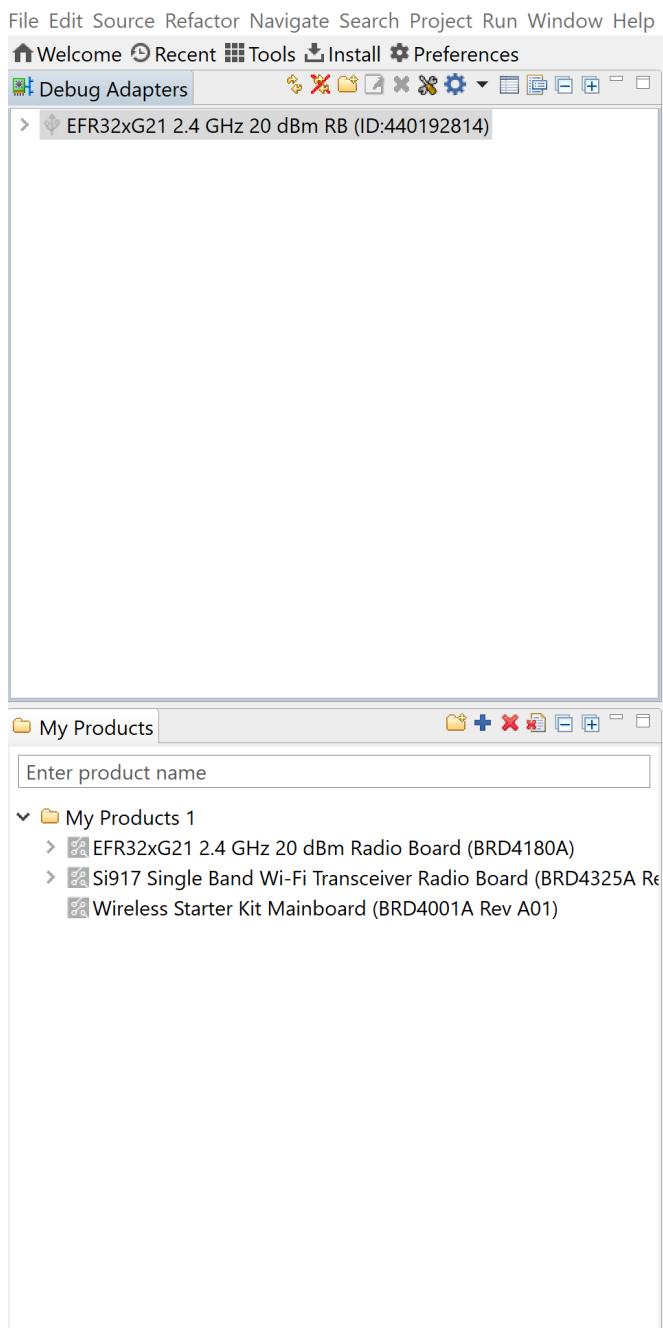


6. The Launcher page will display the selected radio board's details.



7. Verify the following in the General Information section:

- o The Debug Mode is Onboard Device (MCU).
- o The Preferred SDK is the version you selected earlier.



The screenshot shows the Silicon Labs Studio interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar below has icons for Welcome, Recent, Tools, Install, Preferences, and various debug adapter options. A dropdown menu shows "EFR32xG21 2.4 GHz 20 dBm RB (ID:440192814)". The main workspace displays the title "EFR32xG21 2.4 GHz 20 dBm RE". Below the title are tabs for OVERVIEW, EXAMPLE PROJECTS & DEMOS, and DOCUMENTATION. The OVERVIEW tab is selected. On the left, there's a sidebar titled "My Products" with a search bar and a list of products under "My Products 1": EFR32xG21 2.4 GHz 20 dBm Radio Board (BRD4180A), Si917 Single Band Wi-Fi Transceiver Radio Board (BRD4325A Rev A01), and Wireless Starter Kit Mainboard (BRD4001A Rev A01).

The right side of the interface contains two sections: "General Information" and "Board".

General Information:
Connected Via: [J-Link Silicon Labs](#)
[Configure](#)
Debug Mode: **Onboard Device (MCU)** [Change](#)
Adapter FW: **1v4p6b1171** [Latest](#)
Secure FW: **1.2.1** [Update to 1.2.11](#) | [Changelog](#)
Preferred SDK:
Gecko SDK Suite v4.3.1 [Manage SDKs](#) ▾

Board:


8. Open the Example Projects and Demos tab, select a project, and click Create Project.

EFR32xG24 2.4 GHz 10 dBm Radio Board (BRD4186C Rev A00)

OVERVIEW EXAMPLE PROJECTS & DEMOS DOCUMENTATION COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos

Example Projects

Solution Examples

What are Demo and Example Projects?

Wireless Technology Clear

- Bluetooth (51)
- Bluetooth Mesh (20)
- Connect (9) 2
- Matter (45) 1
- RAIL (22)
- Thread (17)
- Zigbee (26)

Device Type Clear

- Host (1)
- NCP (0)
- RCP (0)
- Soc (30)

Ecosystem Clear

- Amazon (0)

MCU Clear

- 32-bit MCU (0)
- Bootloader (0)

Matter - SoC Light Switch over Wi-Fi
This project builds a Matter Light Switch for WI-FI WF200 that can be developed inside Simplicity Studio

[View Project Documentation](#) [CREATE](#)

Matter - SoC Lighting over Thread
This is a Matter Lighting Application over Thread for BRD4186C

[View Project Documentation](#) [RUN](#) [CREATE](#)

Matter - SoC Lighting over Wi-Fi
This project builds a Matter Lighting app for WI-FI RS9116 and 917NCP that can be developed inside Simplicity Studio

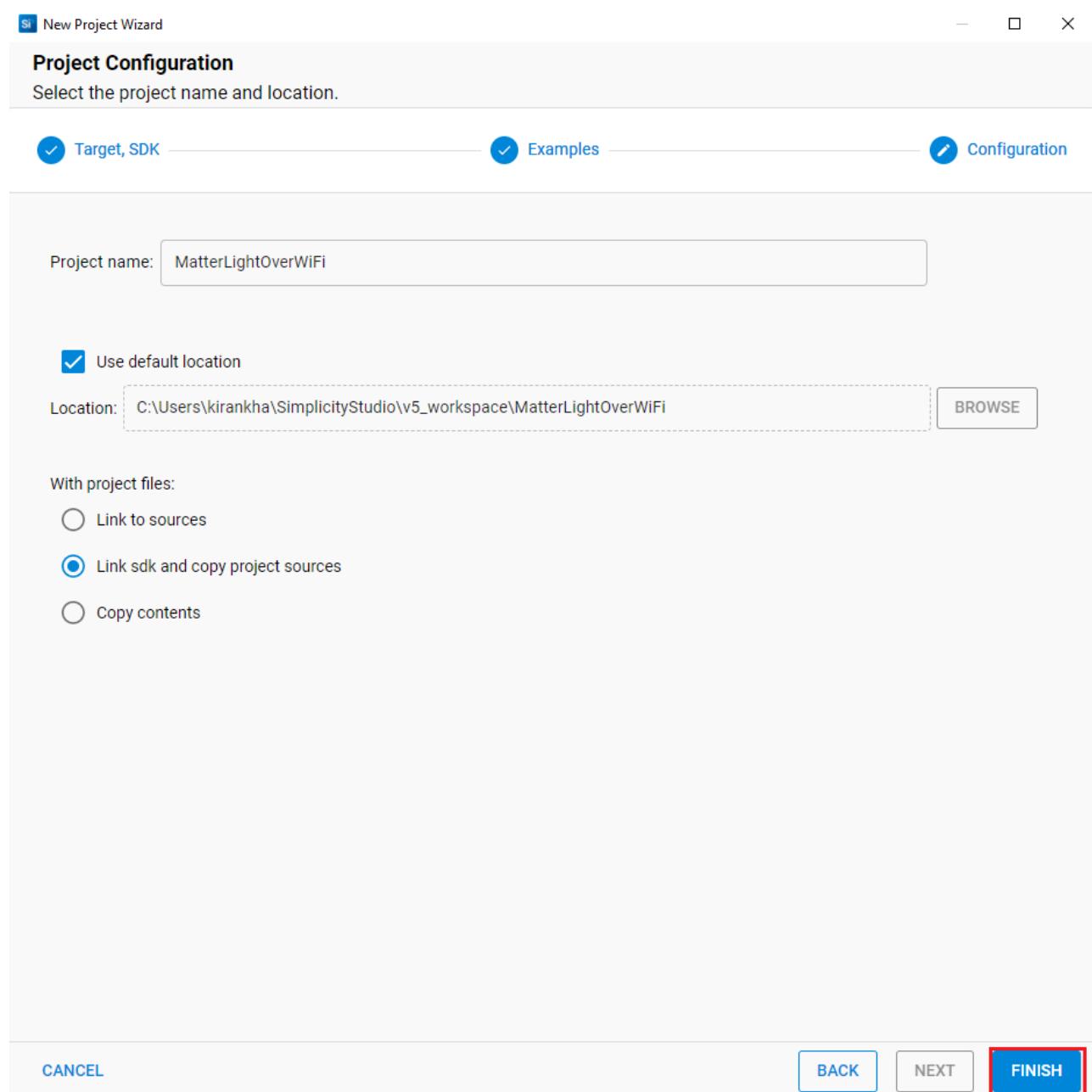
[View Project Documentation](#) [CREATE](#)

Matter - SoC Lock over Thread
This project builds a Matter Lock that can be developed inside Simplicity Studio

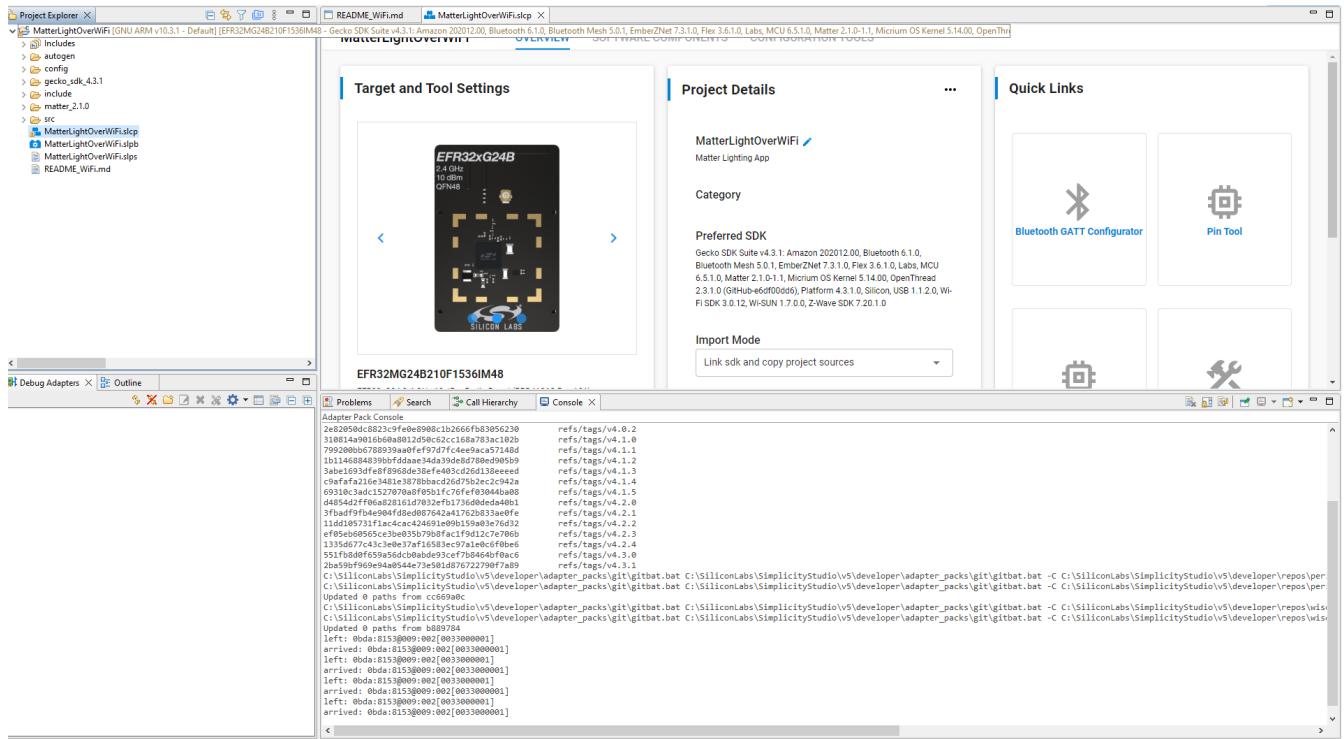
[View Project Documentation](#) [CREATE](#)

Matter - SoC Lock over Wi-Fi

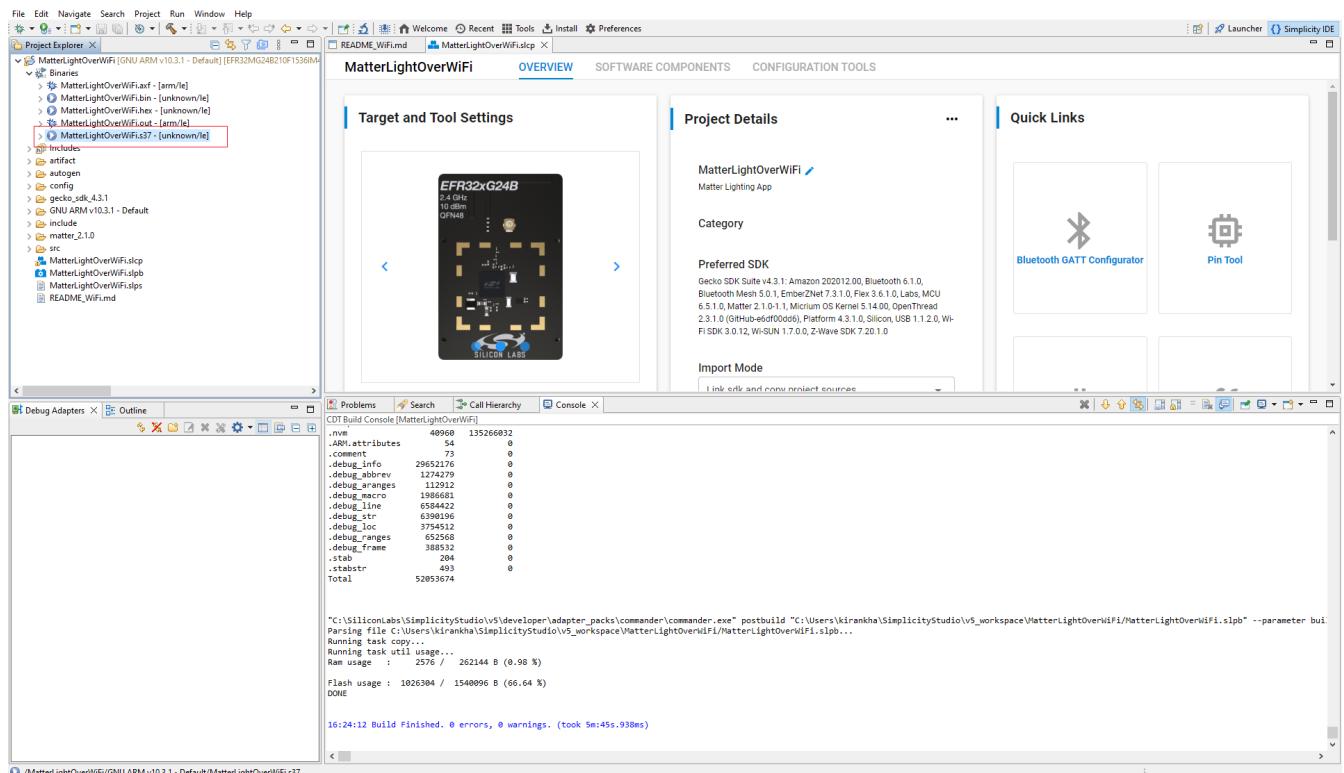
9. In the New Project Wizard window, click **Finish**.



10. Once the project is created, right-click the project and select **Build Project** in the Project Explorer tab.

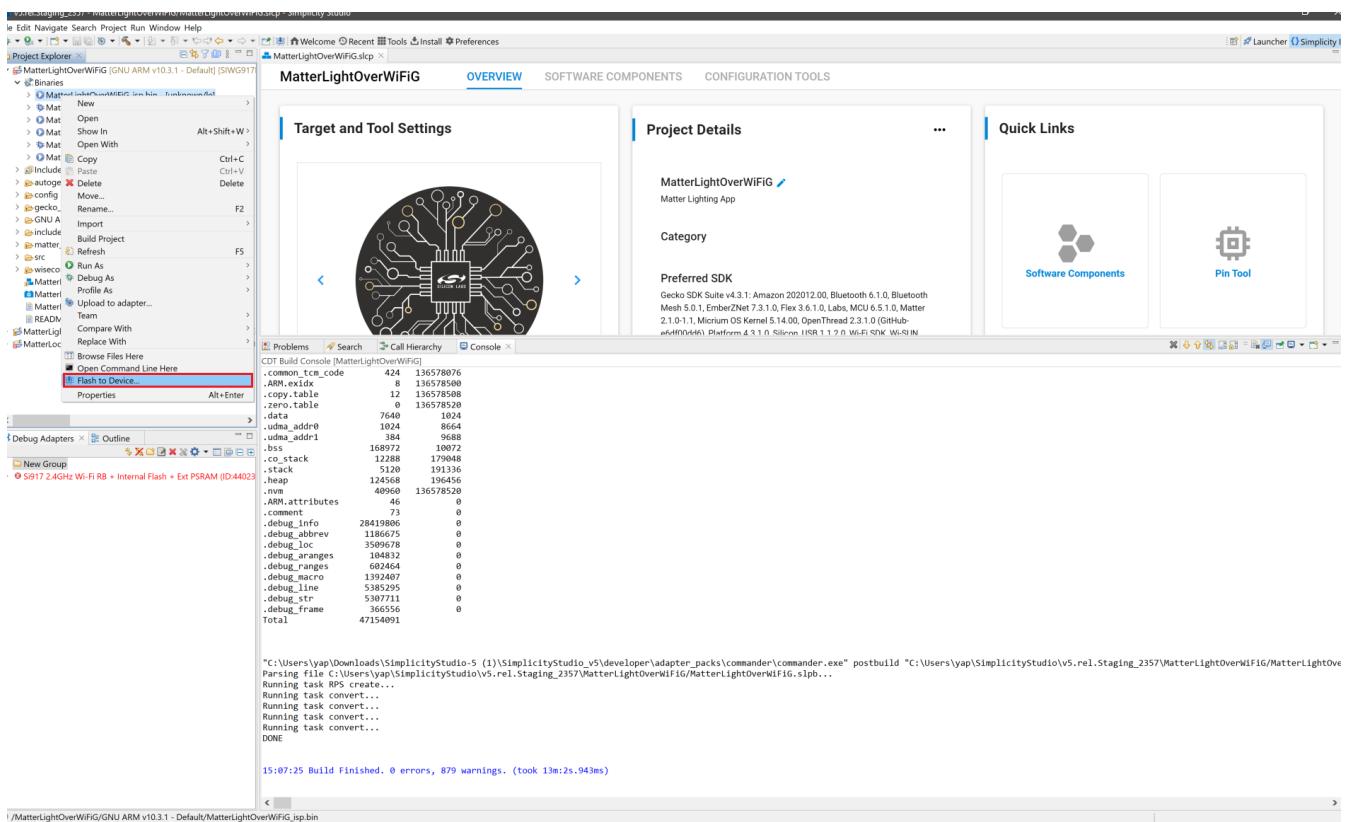


11. Once the project is compiled successfully, go to the Project Explorer view and expand the binaries folder to flash the binary.

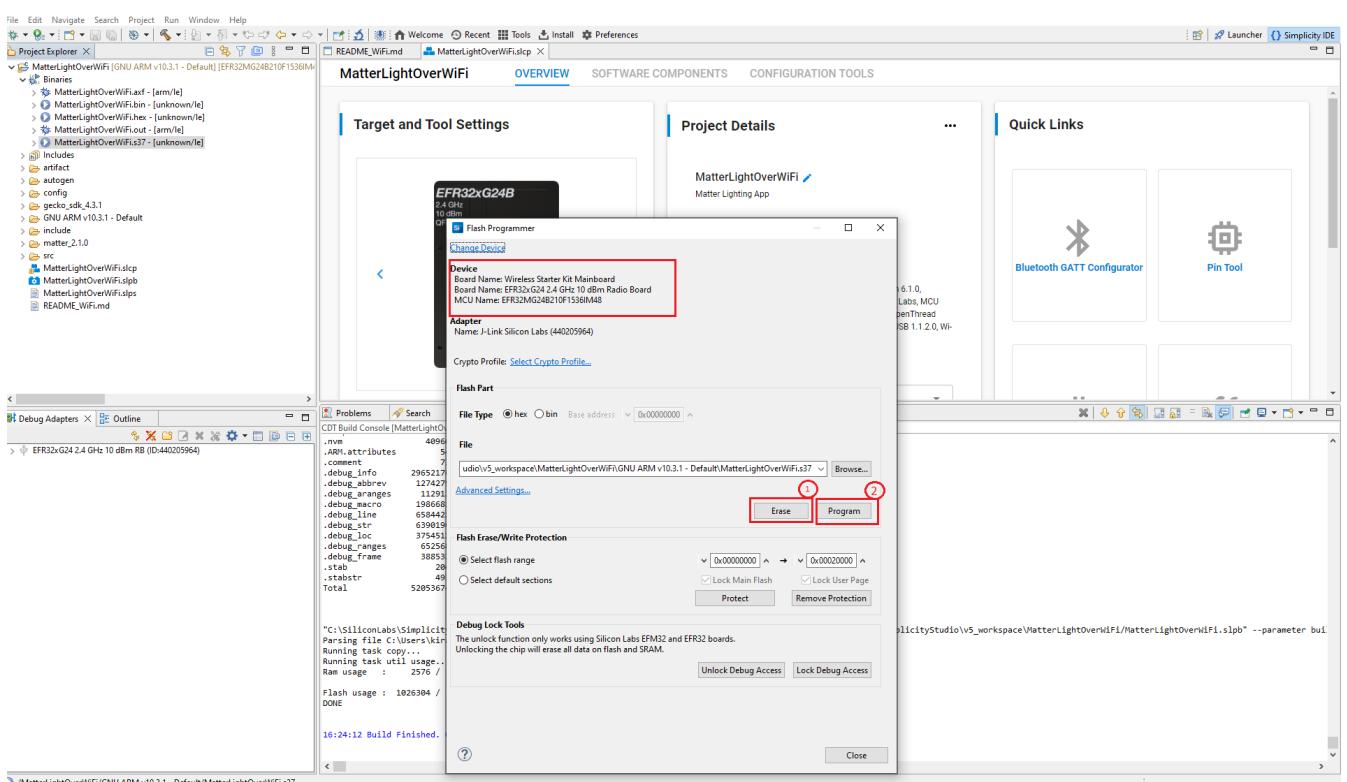


12. Right-click the selected 's37' binary and click **flash to device**.

Build an NCP Application Using Studio



13. The Flash programmer window opens. Click **Erase** and then **Program** to start flashing.



Note: Output of the EFR32 NCP Host application will be displayed on the J-Link RTT Viewer.

Flash a Binary

Flashing the Matter Binaries Using Simplicity Commander

To flash the application for EFR32 and SiWx917 SOC Board Simplicity Commander software will be used.

Before flashing the application for EFR32 Boards, flash **bootloader images** as per board variants:

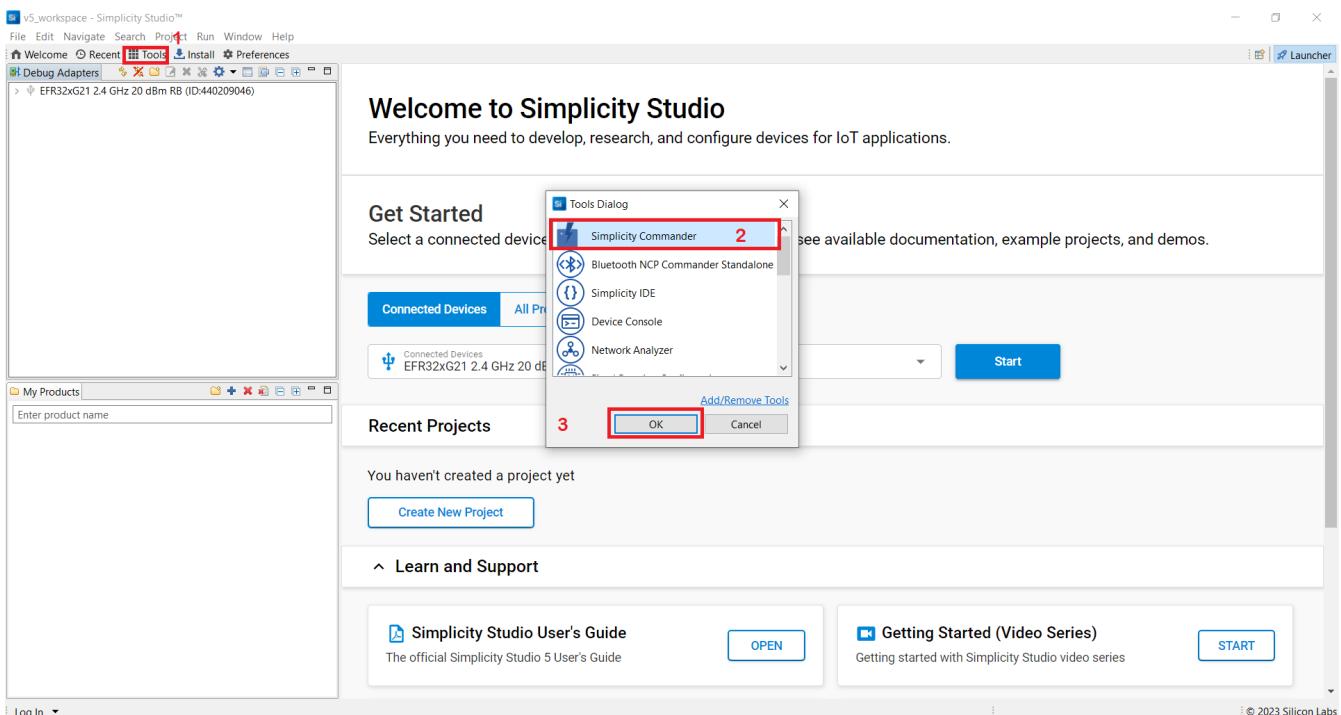
- **BRD4186C Board**
 - For MG24 + RS9116 :- Internal Bootloader (bootloader-storage-internal-single-512k-BRD4186C-gsdk4.1)
 - For MG24 + WF200 :- External Bootloader (bootloader-storage-spiflash-single-1024k-BRD4186C-gsdk4.1)
- **BRD4187C Board**
 - For MG24 + RS9116 :- Internal Bootloader (bootloader-storage-internal-single-512k-BRD4187C-gsdk4.1)
 - For MG24 + WF200 :- External Bootloader (bootloader-storage-spiflash-single-1024k-BRD4187C-gsdk4.1)

Bootloader binaries are available in the respective path of codebase `third_party/silabs/matter_support/matter/efr32/bootloader_binaries` folder. Silicon Labs recommends always flashing the latest bootloader binaries from the codebase.

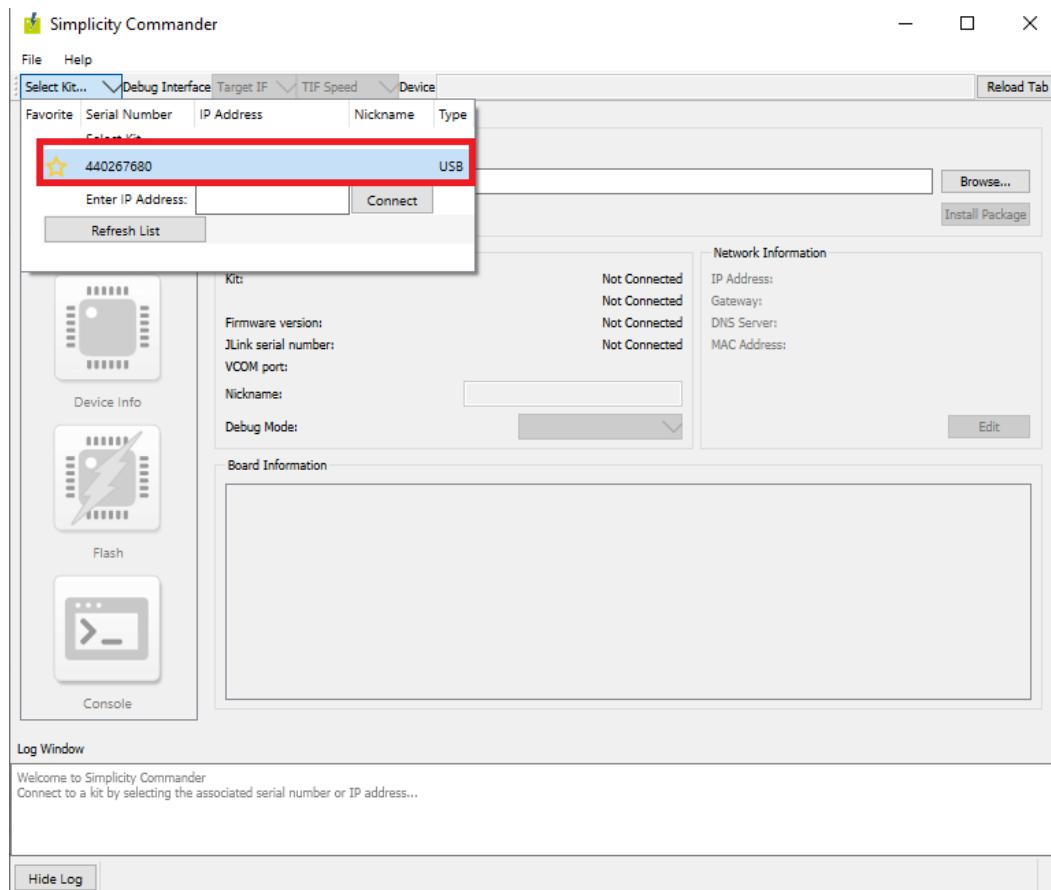
Note: Bootloader binaries are flashed using Simplicity Commander only. It supports EFR32 Boards only.

Flashing the Bootloader Binaries for EFR32 Board using Simplicity Commander

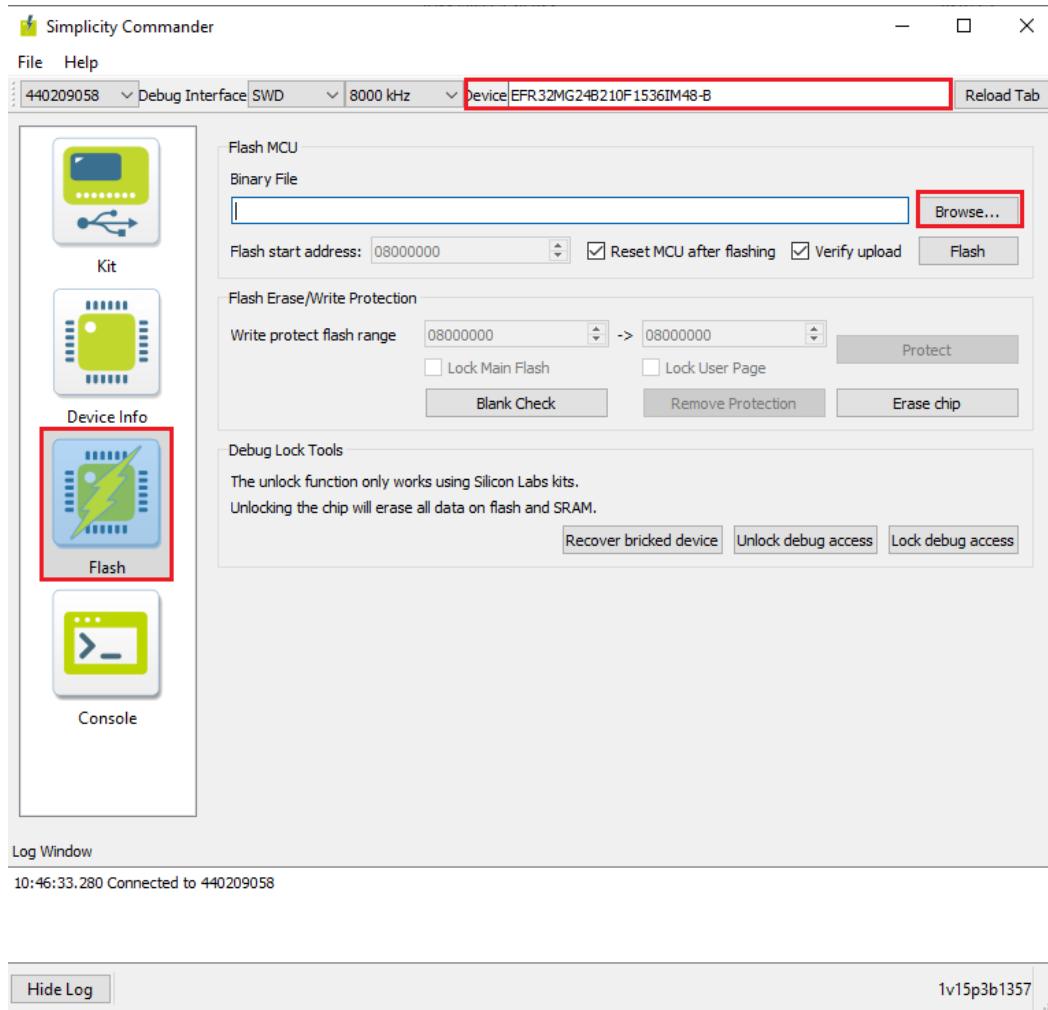
1. In the Simplicity Studio home page, click Tools.
2. In the Tools dialog, select Simplicity Commander and click OK.



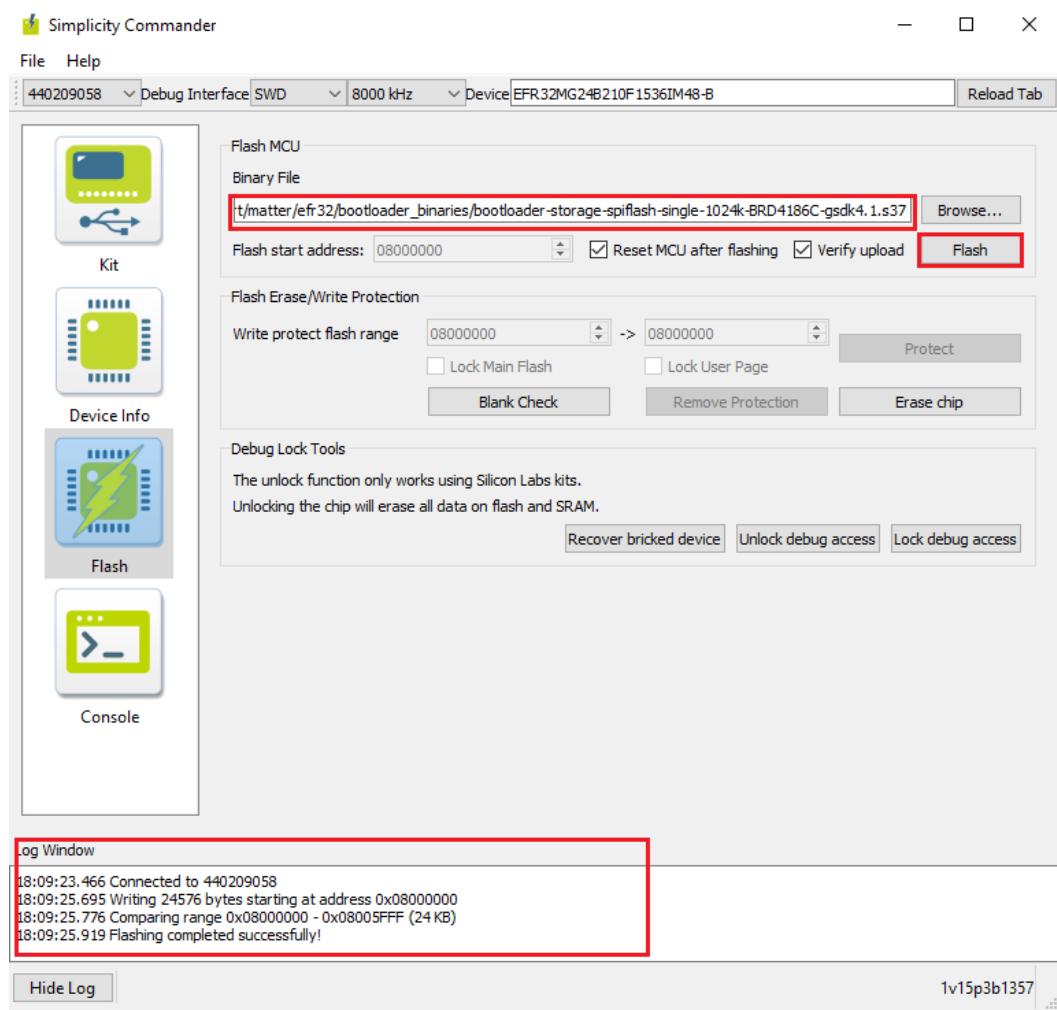
3. In the Simplicity Commander window, click **Select Kit** and choose your radio board.



4. In the navigation pane, go to the Flash section.
5. Above beside "Reload tab" board will be displayed, click Browse next to the Binary File field and locate bootloader binary.

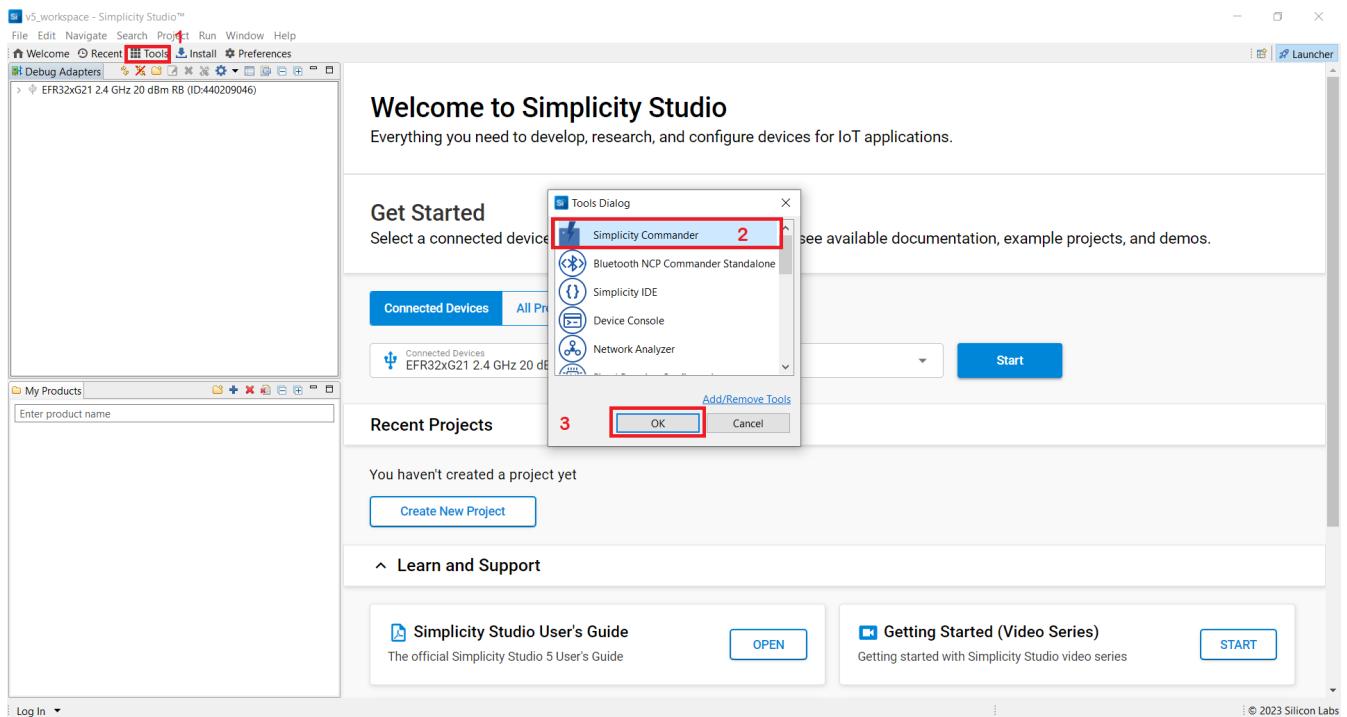


6. Click Flash, the bootloader will be flashed and the Log Window will display a "Flashing completed Successfully" message.

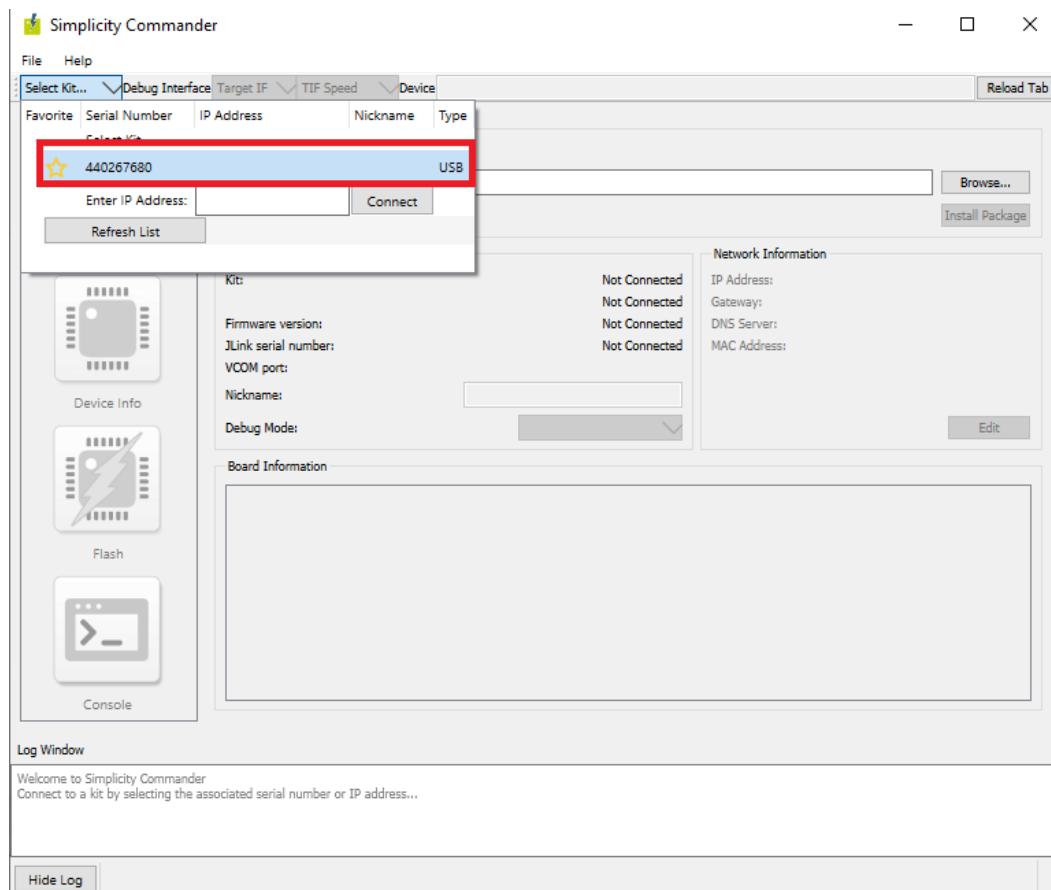


Flashing the EFR32 Matter Binary using Simplicity Commander

1. In the Simplicity Studio home page, click Tools.
2. In the Tools dialog, select Simplicity Commander and click OK.

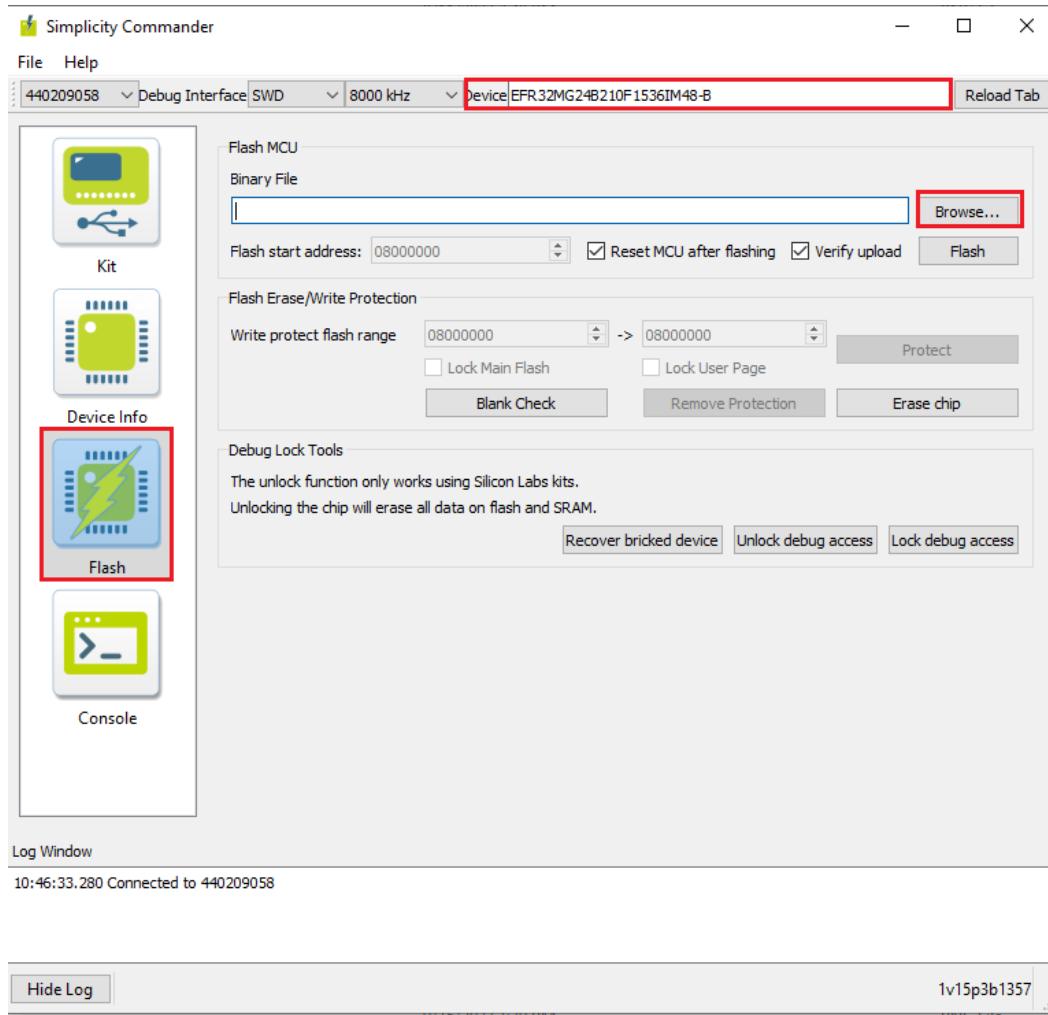


3. In the Simplicity Commander window, click Select Kit and choose your radio board.

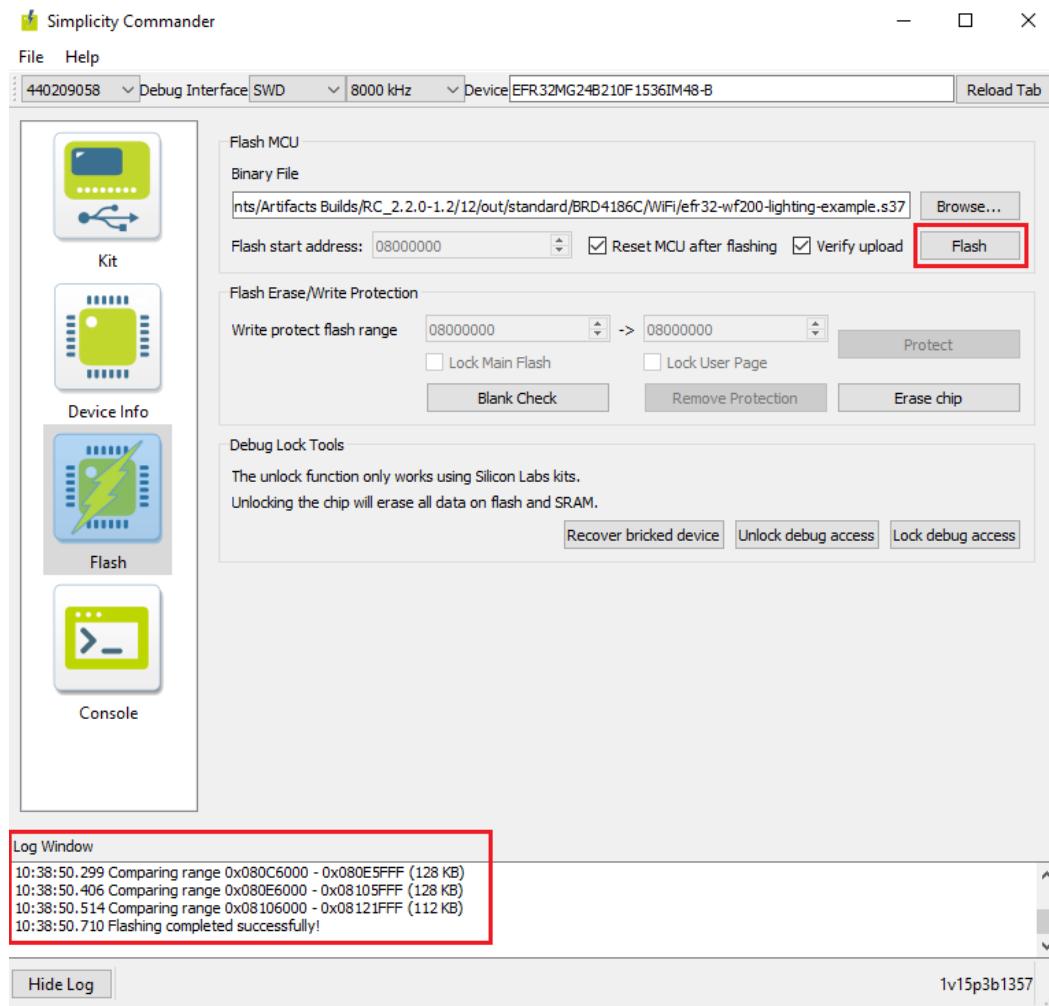


4. In the navigation pane, go to the **Flash** section.

5. Your board will be displayed. Click **Browse** next to the **Binary File** field and locate the binary.



6. Click **Flash**. The binary will be flashed and the Log Window will display a "Flashing completed Successfully" message.



Flashing the SiWx917 SOC Matter Binary using Simplicity Commander

SiWx917 SoC device support is available in the latest [Simplicity Commander](#). The SiWx917 SOC board will support .rps only file to flash. Follow these steps to create and flash .rps file using .s37.

1. Locate Simplicity Commander in your PC/Laptop where it is installed through command prompt(cmd).

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3603]
(C) Microsoft Corporation. All Rights Reserved.

C:\Users\Kirankha\Downloads\SimplicityCommander-Microsoft\SimplicityCommander-Microsoft\Comender_min32_x64_1v16p1b1501\Simplicity_Commander>
```

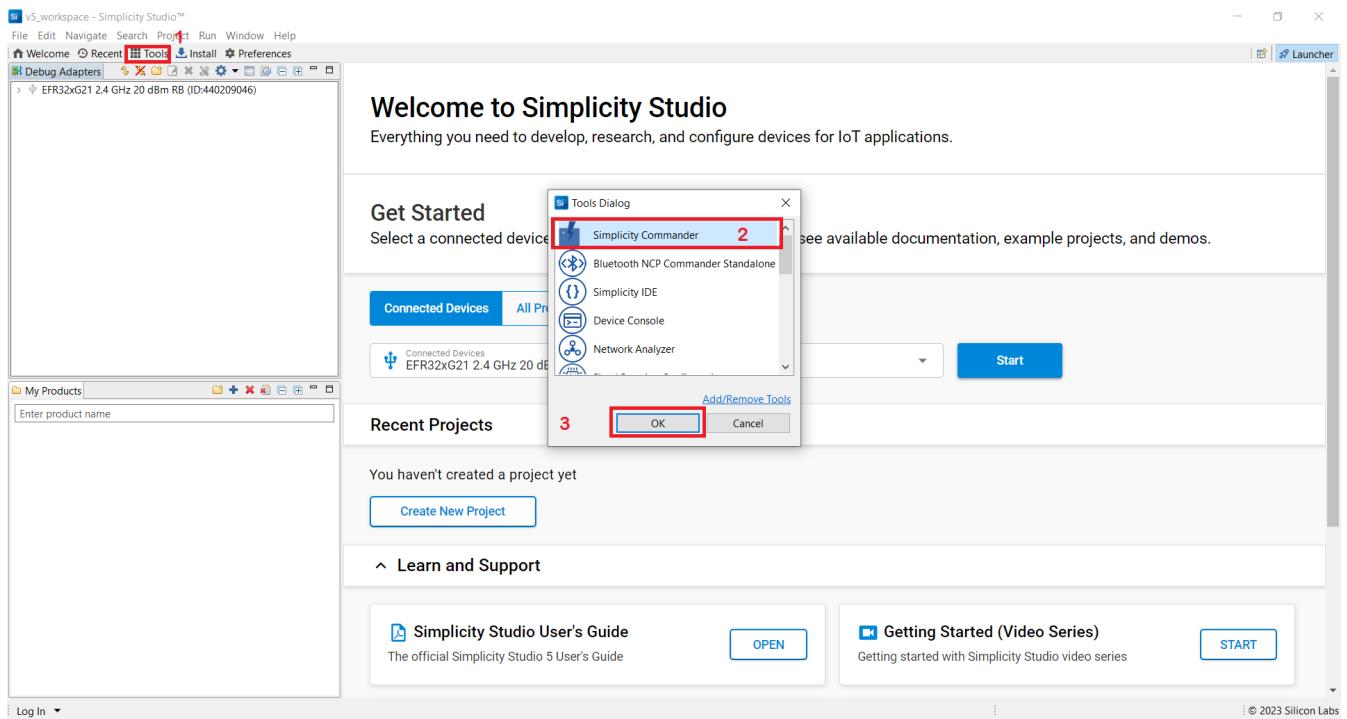
2. Copy and paste the built .s37 binary file to the Simplicity commander path.
3. Convert .s37 binary to .rps using the command below using commander terminal.

```
comander rps convert <file_name.rps> --app <file_name.s37>
```

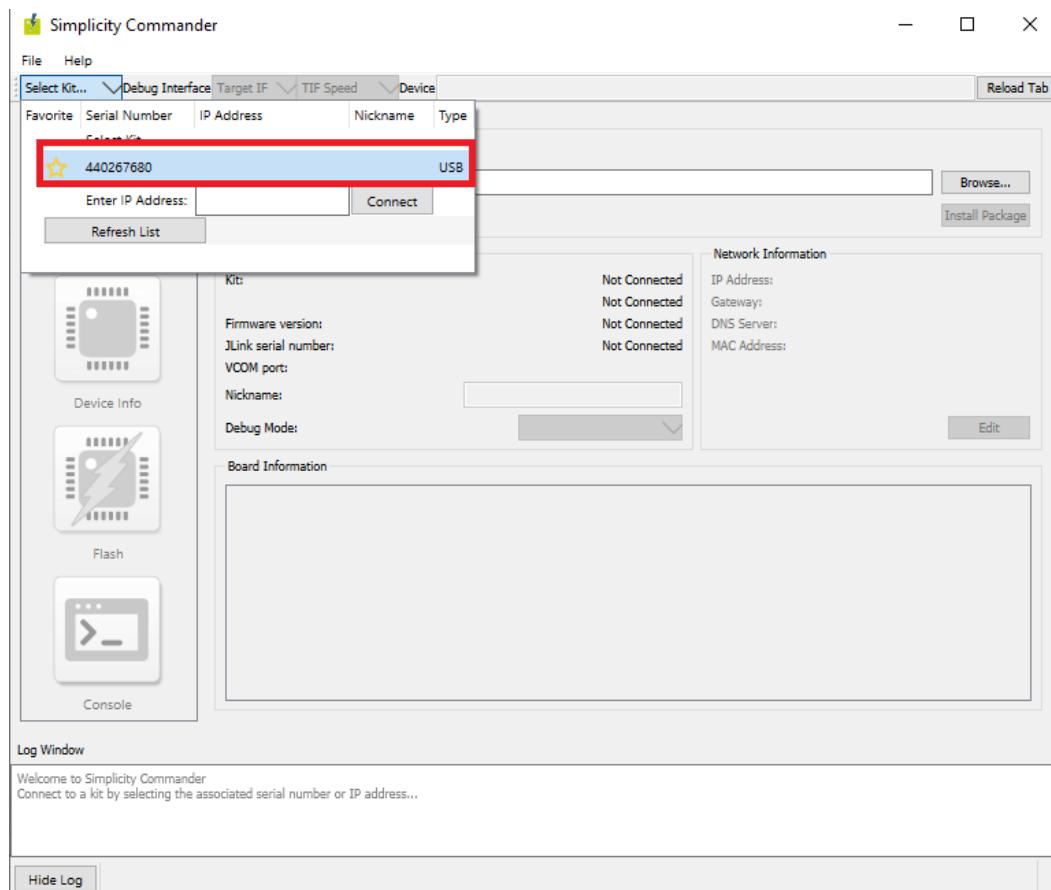
4. Flash to the device using command or follow the next steps to flash through Commander Software.

```
comander rps load <file-name>.rps
```

5. In the Simplicity Studio home page, click **Tools**.
6. In the Tools dialog, select Simplicity Commander and click OK.

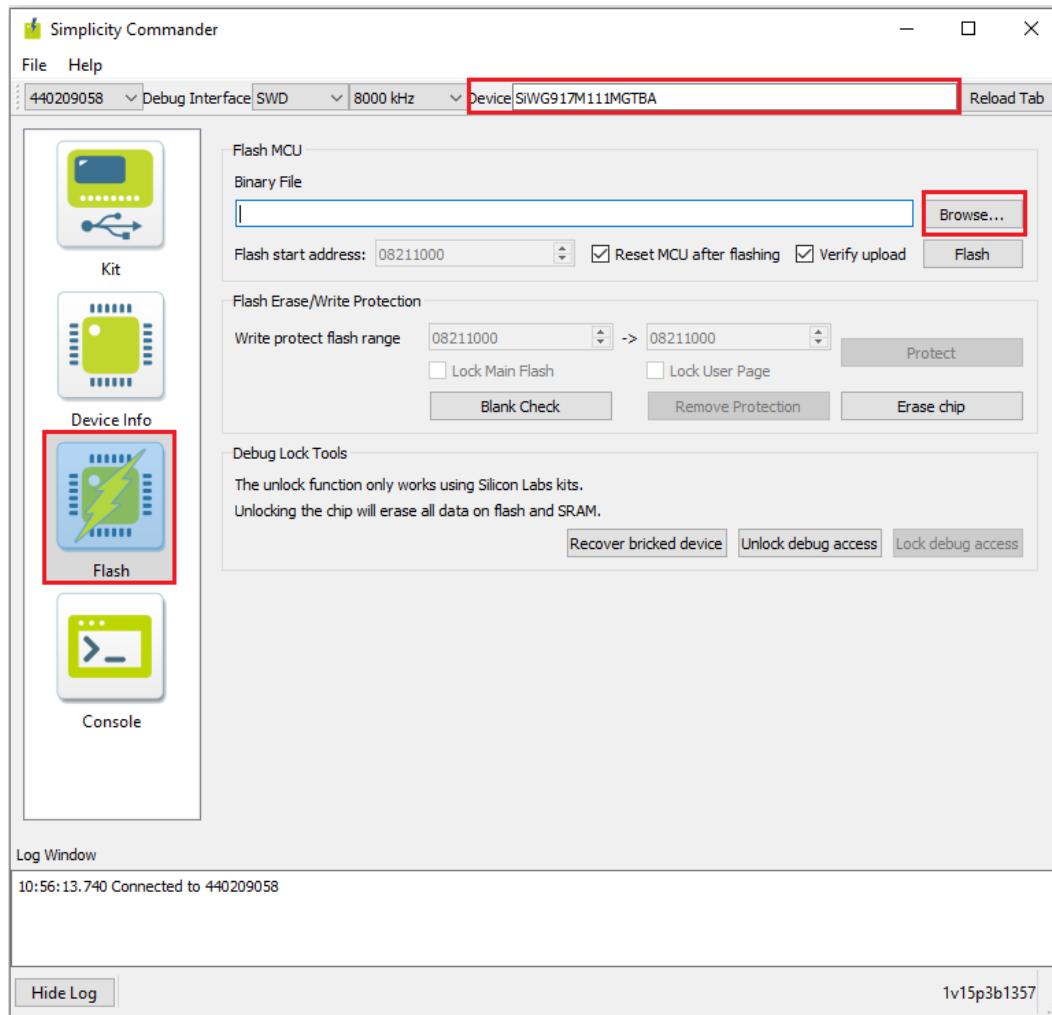


7. In the Simplicity Commander window, click **Select Kit** and choose your radio board.

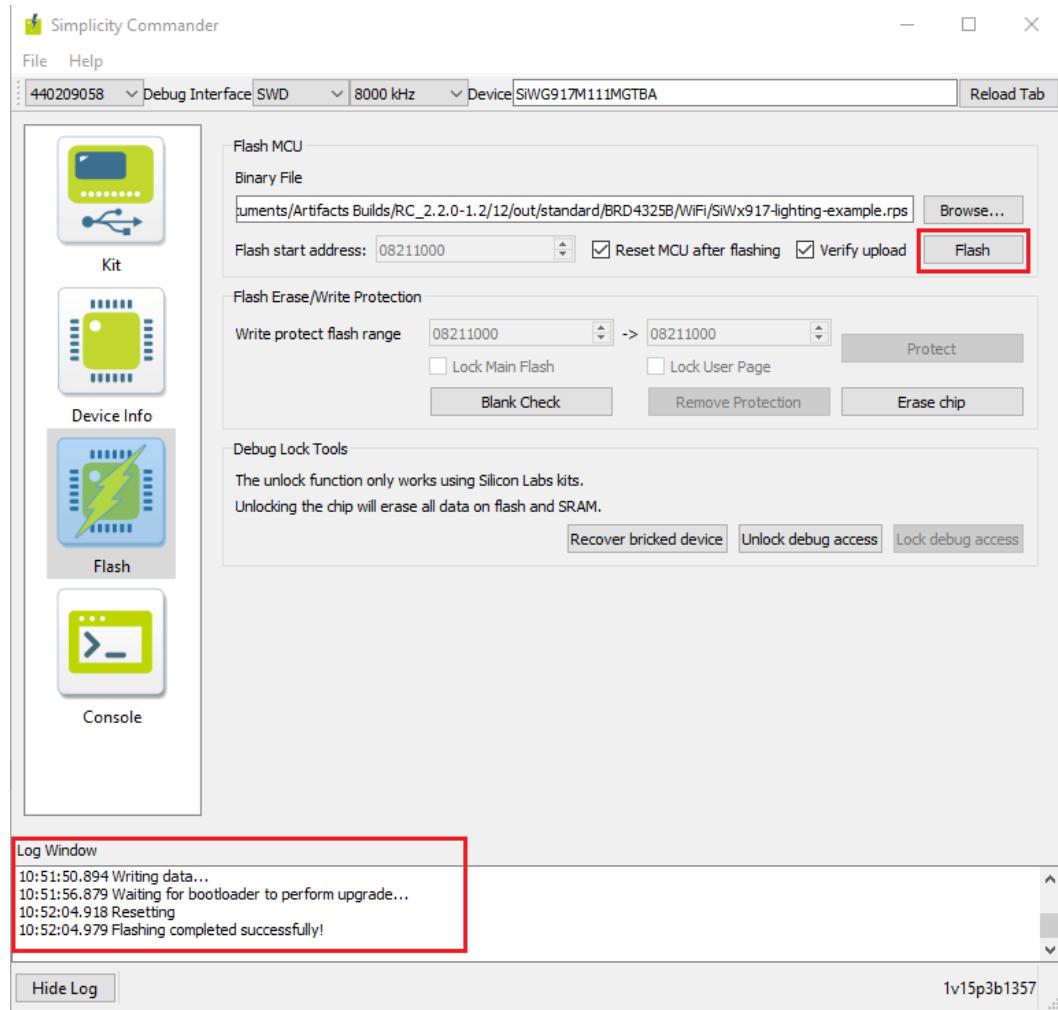


8. In the navigation pane, go to the **Flash** section.

9. Above beside "Reload tab" board will be displayed, click **Browse** next to the **Binary File** field and locate binary.



10. Click **Flash**. The binary will be flashed and the Log Window will display a "Flashing completed Successfully" message.



Set Up the Raspberry Pi

Using the Pre-Built Raspberry Pi "Matter Hub" Image

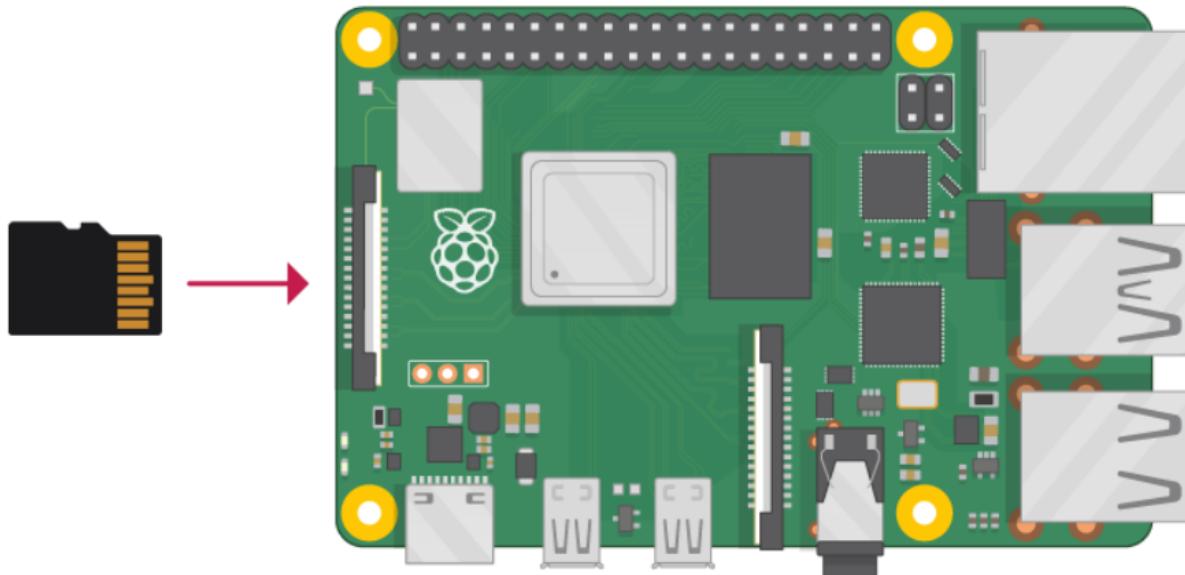
When using a Raspberry Pi as a controller in your Matter network, you have two options.

- Building the Raspberry Pi Environment "from scratch" using a Raspberry Pi 4.
- Using the Silicon Labs Pre-Built Raspberry Pi image available on the [Matter Artifacts page](#).

Building Environment using Raspberry Pi 4

Flash the Ubuntu OS Onto the SD Card

1. Insert the flashed SD card (directly or using a card reader) into the laptop/PC that will run the Raspberry Pi Imager tool.
2. Launch the Raspberry Pi 4 Imager.
3. Flash the Pi image using any one of the following procedure:
 - Click Choose OS > Other General-purpose OS > Ubuntu >Ubuntu xx.xx 64-bit server OS.
Note: Flash the latest version of Ubnutu Server (64-bit server OS for arm64 architecture).
 - Download the Matter Hub Raspberry Pi Image provided on the [Matter Artifacts page](#), then click Choose OS > Use custom, and then select the Matter Hub Raspberry Pi Image that you downloaded.
4. Click **Storage** and select the **SD card detect**.
5. This Raspberry Pi 4's console can be accessed in multiple ways. In [this guide](#) Raspberry Pi 4 is being accessed using Putty.
6. Enter the details like User name, Password, SSID, and its password to connect to the network. Click **Save**.
7. Click **Write** and then **Yes** when you are asked for permission to erase data on the SD card. It will then start flashing the OS onto the SD card.
8. When it is done, click **Continue**.
9. Remove the SD card from the reader and insert it into the Raspberry Pi as shown below:



On powering up the board, the red and green lights should start blinking.

Start Using the Raspberry Pi

1. Power-up the RPi4B. Once it is booted up, check the Raspberry Pi's IP address. Refer to [Finding Raspberry Pi IP address](#) in the References section to get the IP address, or enter the Hostname directly in PuTTY.
2. Once you find the IP address, launch PuTTY, select **Session**, enter the IP address of the Raspberry Pi, and click **Open**.
3. Enter the username and password given at the time of flashing and click **Enter**.
Note: If you do not provide the username and password while flashing, then by default: Username: ubuntu Password: ubuntu
4. Switch to root mode and navigate to path "/home/ubuntu/connectedhomeip/out/standalone" to find the chip-tool. On the pre-built Matter Hub image, the chip-tool will be ready and working. Keep the PuTTY session open for the further steps.
5. If you are building the chip-tool from scratch, update the latest packages by running following commands in the terminal:

```
$ sudo apt update  
$ sudo apt install
```

6. Install the required packages using the following commands:

```
$ sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev
```

If you see any popups between installs, you can select **OK** or **Continue**.

Build Environment

1. Installing prerequisites on Raspberry Pi 4. Follow the instructions in [the Project CHIP GitHub Site](#), in the section "Installing prerequisites on Raspberry Pi 4".
2. To build the environment, follow the Software setup and Compiling chip-tool steps in [Software setup](#).

Bluetooth Setup

Because Bluetooth LE (BLE) is used for commissioning on Matter, make sure BLE is up and running on Raspberry Pi. Raspberry Pi internally has some issues with BLE that may cause it to crash.

```
$ sudo systemctl status bluetooth.service
```

To stop BLE if it is already running:

```
$ sudo systemctl stop bluetooth.service
```

To restart the Bluetooth service, first enable it:

```
$ sudo systemctl enable bluetooth.service
```

When you check the status of the Bluetooth service, it will be inactive because it has been enabled but not restarted:

```
$ sudo systemctl status bluetooth.service
```

Restart the service:

```
$ sudo systemctl restart bluetooth.service
```

Now the status of the service should be active and running:

```
$ sudo systemctl status bluetooth.service
```

Run an Application

Running a Matter over Wi-Fi Application

In order to run a Matter over Wi-Fi application you must first create a Matter network using the chip-tool and then control the Matter device from the chip-tool.

Creating the Matter Network

This procedure uses the chip-tool installed on the Matter Hub. The commissioning procedure does the following:

- Chip-tool scans BLE and locates the Silicon Labs device that uses the specified discriminator.
- Establishes operational certificates.
- Sends the Wi-Fi SSID and Passkey.
- The Silicon Labs device will join the Wi-Fi network and get an IP address. It then starts providing mDNS records on IPv4 and IPv6.
- Future communications (tests) will then happen over Wi-Fi.

Commissioning can be done using chip-tool running either on Linux or Raspberry Pi

1. Get the SSID and PSK of the Wi-Fi network (WPA2 - Security) you are connected to.
2. Go to `$MATTER_WORKDIR/matter` directory and run the following:

```
$ out/standalone/chip-tool pairing ble-wifi <node_id> <ssid> <password> <pin_code> <discriminator>
```

In this command:

- node_id is the user-defined ID of the node being commissioned.
- ssid and password are credentials.
- pin_code and discriminator are device-specific keys.

Note: You can find these values in the logging terminal of the device (for instance UART) when the device boots up. For example:

```
I: 254 [DL]Device Configuration:  
I: 257 [DL] Serial Number: TEST_SN  
I: 260 [DL] Vendor Id: 65521 (0xFFFF1)  
I: 263 [DL] Product Id: 32768 (0x8000)  
I: 267 [DL] Hardware Version: 1  
I: 270 [DL] Setup Pin Code: 20202021  
I: 273 [DL] Setup Discriminator: 3840 (0xF00)  
I: 278 [DL] Manufacturing Date: (not set)  
I: 281 [DL] Device Type: 65535 (0xFFFF)
```

The node ID used here is 1122. This will be used in future commands. '\$SSID' is a placeholder for your Wi-Fi SSID and '\$PSK' is a placeholder for the password of your Wi-Fi network. '20202021' is the Setup Pin Code used to authenticate the device. '3840' is the Setup Discriminator used to discern between multiple commissionable device advertisements.

If there are any failures, run the following command and then re-run the chip-tool command:

```
$ rm -rf /tmp/chip_*
```

If you are having difficulty getting the chip-tool to commission the device successfully, it may be because you have more than one network interface available to the chip-tool. The device on which you are running the chip-tool must be on the same Wi-Fi network. For instance, if you have an Ethernet connection as well as a Wi-Fi connection, you need to unplug the Ethernet connection and try running the chip-tool.

Controlling the Matter Accessory Device

1. In a PuTTY session to the Matter hub, use the chip-tool to test the Matter light device.
 1. Control the light status of the light MAD Using `./chip-tool onoff on nodeID 1`. You can also use `chip-tool toggle nodeID 1`.
 2. For dev board with buttons available, you can use BTN1 to toggle the light status locally.

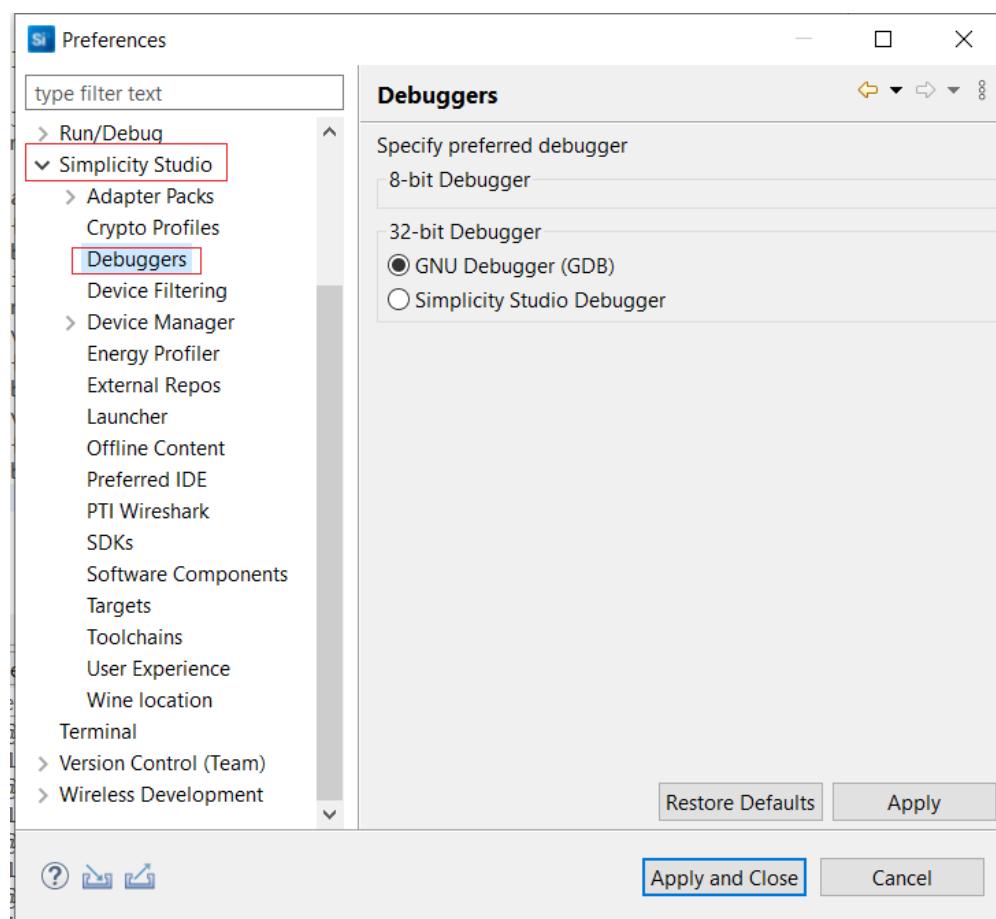
Factory Reset the Device

As the device remembers the Access Point credentials given for commissioning, if you want to run the demo multiple times, do a factory reset by pressing the BTN0 on WSTK for about 6-7 seconds. The LED0 will flash 3 times.

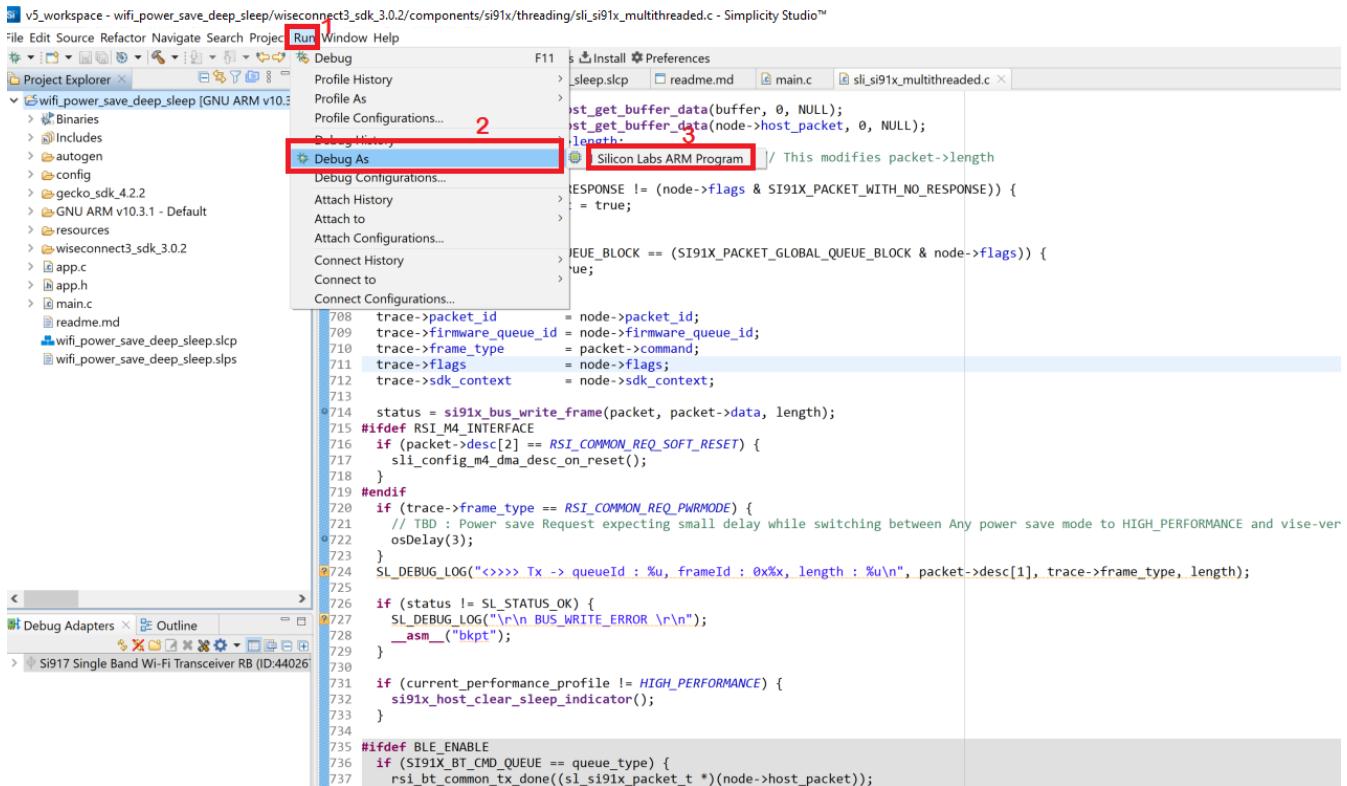
Debug an Application

Debug the Application

1. In the Project Explorer pane, select the project name.
2. To Enable GNU Debugger Select **Preferences** in Launcher Tab.
3. Expand **Simplicity Studio** Tab and click on **Debuggers**. Select **GNU Debugger** and Click on **Apply and Close**.



4. From the menu bar, select Run > Debug As > 1 Silicon Labs ARM Program.



The screenshot shows the Simplicity Studio interface with the following details:

- Project Explorer:** Shows the project structure for "wifi_power_save_deep_sleep" with files like main.c, app.c, app.h, and various configuration and resource files.
- Debug As:** A dropdown menu is open under the "Debug" menu, with "Silicon Labs ARM Program" highlighted (step 3).
- Code Editor:** Displays the source code for "sli_si91x_multithreaded.c". The code includes comments indicating modifications to packet length and handling of RSI_COMMON_REQ_SOFT_RESET.
- Debug Adapters:** Shows a connection to "Si917 Single Band Wi-Fi Transceiver RB (ID:44026)".

5. Studio will switch to debug mode and halt execution at the main() function in your application.
6. Add a break point in the desired location of the code and click the Resume button (having an icon with a rectangular bar and play button).
7. Execution will halt at the break point.
8. Use the following debug functions to direct the execution of the code:
 - o Step In button (having an icon with an arrow pointing between two dots).
 - o Step Over button (having an icon with an arrow going over a dot).
 - o Step Out button (having an icon with an arrow pointing out from between two dots).

The screenshot shows the Simplicity Studio IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help, Welcome, Recent, Tools, Install, Preferences, and a toolbar with icons for file operations. The left sidebar displays the Project Explorer with a tree view of files: Silicon Labs ARM MCU: Si917-SB00, wifi_power_save_deep_sleep.auf, and main() at main.c:30 0x8212320. The main workspace shows the source code for 'main.c' with syntax highlighting for comments, keywords, and identifiers. The code implements a deep sleep management routine using Silicon Labs' system and power manager APIs. The right side features a table for managing variables (Name, Type, Value) and a status bar at the bottom with tabs for Console, Memory, Problems, Executables, and Debugger Console.

```
12 * www.silabs.com/about-us/legal/master-software-license-agreement. This
13 * software is distributed to you in Source Code format and is governed by the
14 * sections of the MSLA applicable to Source Code.
15 *
16 ****
17 #include "sl_component_catalog.h"
18 #include "sl_system_init.h"
19 #include "app.h"
20 #if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
21 #include "sl_power_manager.h"
22 #endif
23 #if defined(SL_CATALOG_KERNEL_PRESENT)
24 #include "sl_system_kernel.h"
25 #else // SL_CATALOG_KERNEL_PRESENT
26 #include "sl_system_process_action.h"
27 #endif // SL_CATALOG_KERNEL_PRESENT
28
29 int main(void)
30 {
31     // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
32     // Note that if the kernel is present, processing task(s) will be created by
33     // this call.
34     sl_system_init();
35
36     // Initialize the application. For example, create periodic timer(s) or
37     // task(s) if the kernel is present.
38     app_init();
39
40 #if defined(SL_CATALOG_KERNEL_PRESENT)
41     // Start the kernel. Task(s) created in app_init() will start running.
42     sl_system_kernel_start();
43 #else // SL_CATALOG_KERNEL_PRESENT
44     while (1) {
45         // Do not remove this call: Silicon Labs components process action routine
46         // must be called from the super loop.
47         sl_system_process_action();
48
49     // Application process.
49 }
```

9. View the standard output or enter input data as needed.

Supported Features

Matter Over Wi-Fi Supported Features

Matter Over Wi-Fi supports the following features on NCP and SoC variants.

- [Intermittently Connected Devices \(ICD\), formerly Sleepy End Devices \(SED\)](#)
- [Direct Internet Connectivity \(DIC\)](#)
- [Interoperability with Ecosystems](#)
- [Optimization of memory usage](#)
- [Reducing power consumption for ICD End Devices](#)
- [Developing a Custom Matter Device](#)
- [JLink RTT SOC Support](#)

Intermittently Connected Devices (ICD)

Matter over Wi-Fi Intermittently Connected Devices (ICD), formerly Sleepy End Devices

This page explains how Matter Wi-Fi ICDs work and how to configure a Matter Wi-Fi SED example.

Overview

Matter provides a Intermittently Connected Device (ICD) operating mode to extend the battery life of power-limited devices. This operating mode leverages native Wi-Fi functionality to enhance the power management features provided within the Matter protocol.

Wi-Fi module power saving is achieved by the Wi-Fi Station notifying the Access Point (AP) that it is entering its power save (PS) mode. Afterwards, the Wi-Fi station will shut down its RF and Wi-Fi SoC blocks to enter power saving mode.

The Access Point (AP) buffers the frames destined to a Wi-Fi station while it is in power save mode. The Access Point (AP) will send the buffered frames to the Wi-Fi station when requested to do so.

During association, the Wi-Fi Station uses the Delivery Traffic Information Map (DTIM) parameter to get from the Access Point (AP) how many beacon intervals it shall spend in sleep mode before it needs to retrieve the queued frames from the Access Point (AP).

Wi-Fi module sleep is implemented by using the PS-Poll Legacy Power Save (DTIM based) method. EFR sleep is implemented by using the power manager component (EM2).

Note: Wi-Fi module sleep is enabled after successful commissioning and EFR sleep is enabled after system bootup.

Note: Wi-Fi is implemented with DTIM-based sleep, since the operational discovery packet is a broadcast packet that will not be buffered by the Wi-Fi router.

Power Save Methods

Deep Sleep Power Save Mode for EFR32

The EFR32 will go into deep sleep (EM2) power save mode by using the power manager module. The power manager is used to transition the system to a power mode when the application is the Idle Task.

In EM2 energy mode, all high frequency clock sources are shut down. Peripherals that require a high frequency clock are unavailable or have limited functionality.

PS-Poll Legacy Power Save for Wi-Fi Module

The PS-Poll Legacy power save mode leverages the PS-Poll frame to retrieve the buffered frames from the Access Point (AP). The PS-Poll frame is a short Control Frame containing the Association Identifier (AID) value of the Wi-Fi station. In the Legacy power save mode, when the Wi-Fi station receives a beacon with its Association Identifier (AID) in the TIM element, it initiates the buffered frame delivery by transmitting a PS-POLL control frame to the Access Point (AP). The AP acknowledges the PS-Poll frame and responds with a single buffered frame.

In this mode, the Wi-Fi station stays active and retrieves a single buffered frame at a time. The AP also indicates that there are more buffered frames for the station using the More Data subfield. The Wi-Fi station continues to retrieve buffered frames using the PS-Poll frame until there are no more buffered frames and the More Data subfield is set to 0. The Wi-Fi station goes back into the sleep afterwards.

A Wi-Fi station can enter sleep mode after sending a Null frame to the AP with the power management (PM) bit set. From then on, the AP will store all packets destined to the Wi-Fi station in a per-device queue and sets the TIM field in the

beacon frame to indicate that packets destined for the Wi-Fi station have been queued.

The Wi-Fi station wakes up to receive buffered traffic for every Delivery Traffic Indication Message (DTIM) beacon. When it detects that the Traffic Indication Map (TIM) field for it has been set, it sends a PS-Poll control frame to the AP.

Delivery Traffic Indication Message (DTIM)

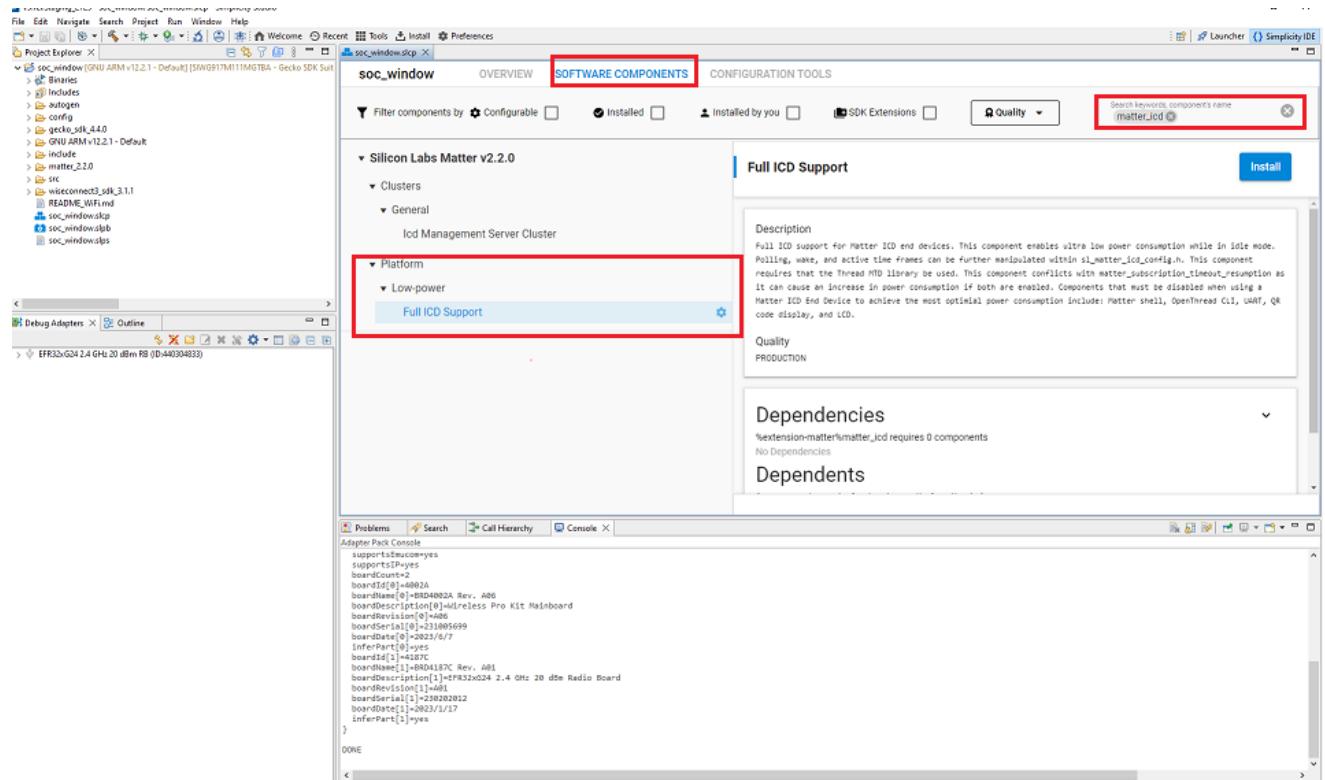
A Wi-Fi station in DTIM Power Save mode can wake at any time to transmit uplink traffic, but can only receive downlink traffic (broadcast, multicast or unicast) immediately after receiving a DTIM beacon. In order to inform the Wi-Fi station in Power Save mode that the access point has buffered downlink traffic, the access point uses the Traffic Indication Map element present in the beacon frames. The Wi-Fi station in Power Save mode wakes up to receive the DTIM beacon and checks the status of the TIM element. This element indicates whether any frames need to be retrieved from the Access Point (AP).

Note: The DTIM parameter can be configured on the access point settings.

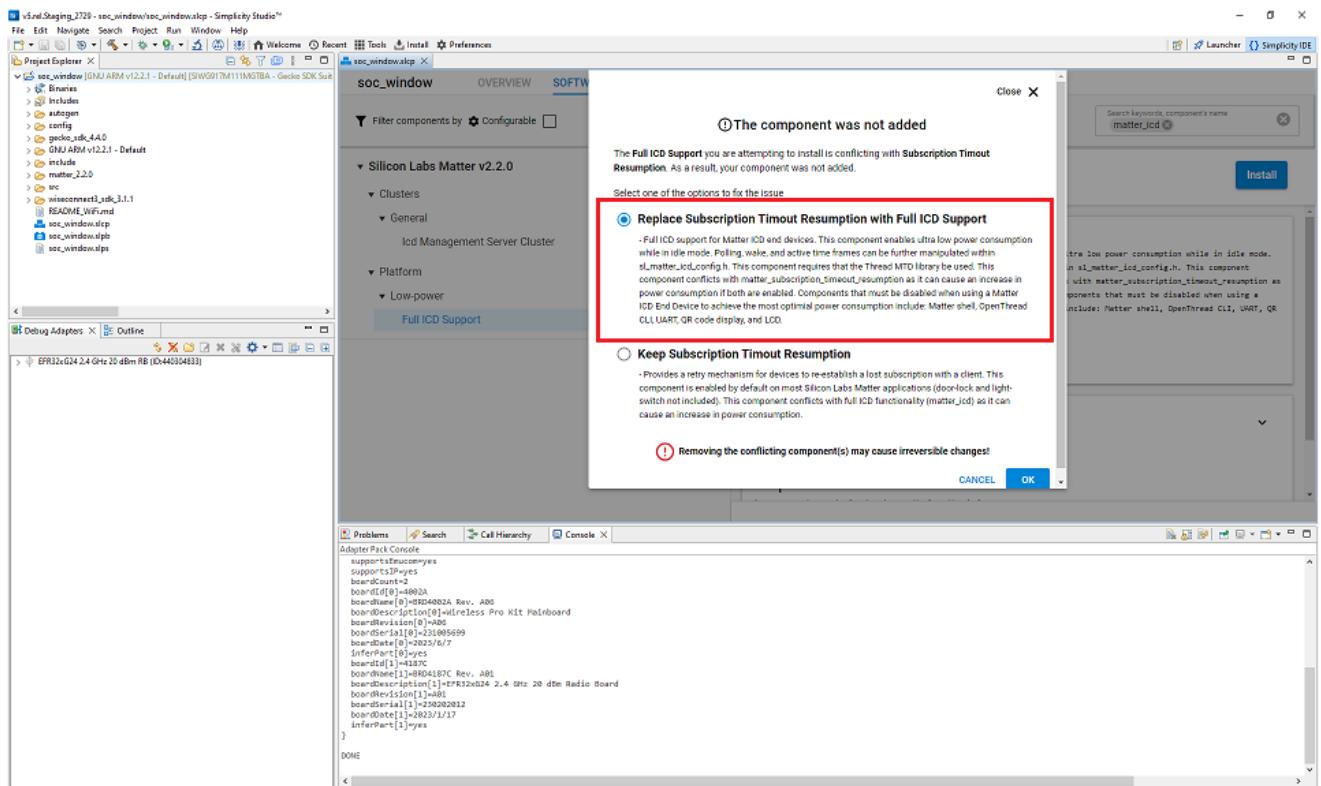
Building with ICD Functionality

To enable ICD functionality for Wi-Fi, the `matter_icd` component needs to be installed within the Software Components tab from Simplicity Studio.

- For rs9116 and WF200: `matter_icd` component is installed by default for lock-app. For thermostat and window need to install mentioned component to enable sleepy.
- For 917NCP: `matter_icd` component is installed by default for lock-app. For thermostat and window need to install mentioned component to enable sleepy.
- For SiWx917 SOC:
 - Search `matter_icd` from search bar and click **Install**.

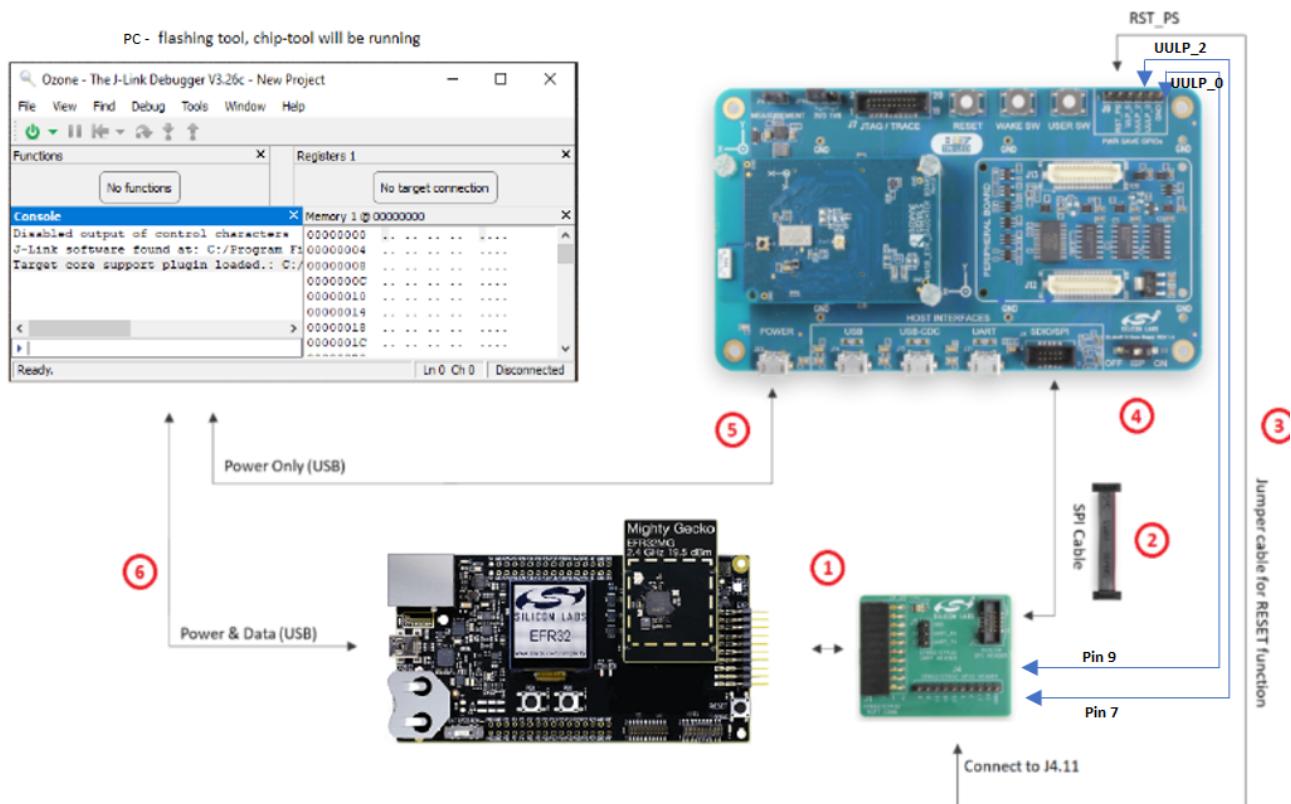


- Click on **Replace Subscription Timeout Resumption**. Sleepy support is enabled; build the project.



EFR32 + RS9116 Setup for ICDs (Sleepy Devices)

- The following GPIO pins should be connected for 9116 and Host handshakes. pin 7 and 9 to UULP_2 and UULP_0 respectively.

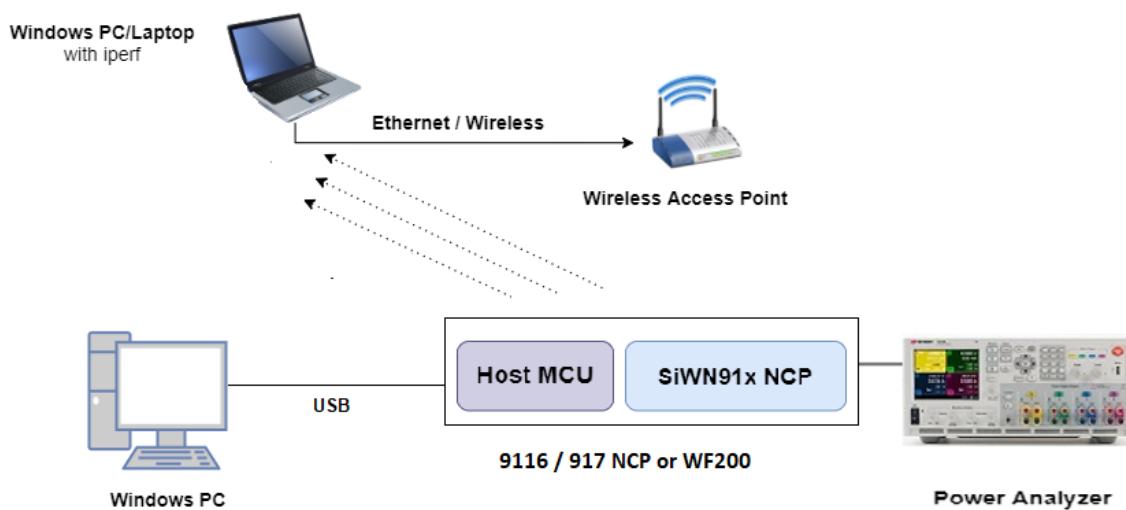


Power Measurements for Wi-Fi Devices

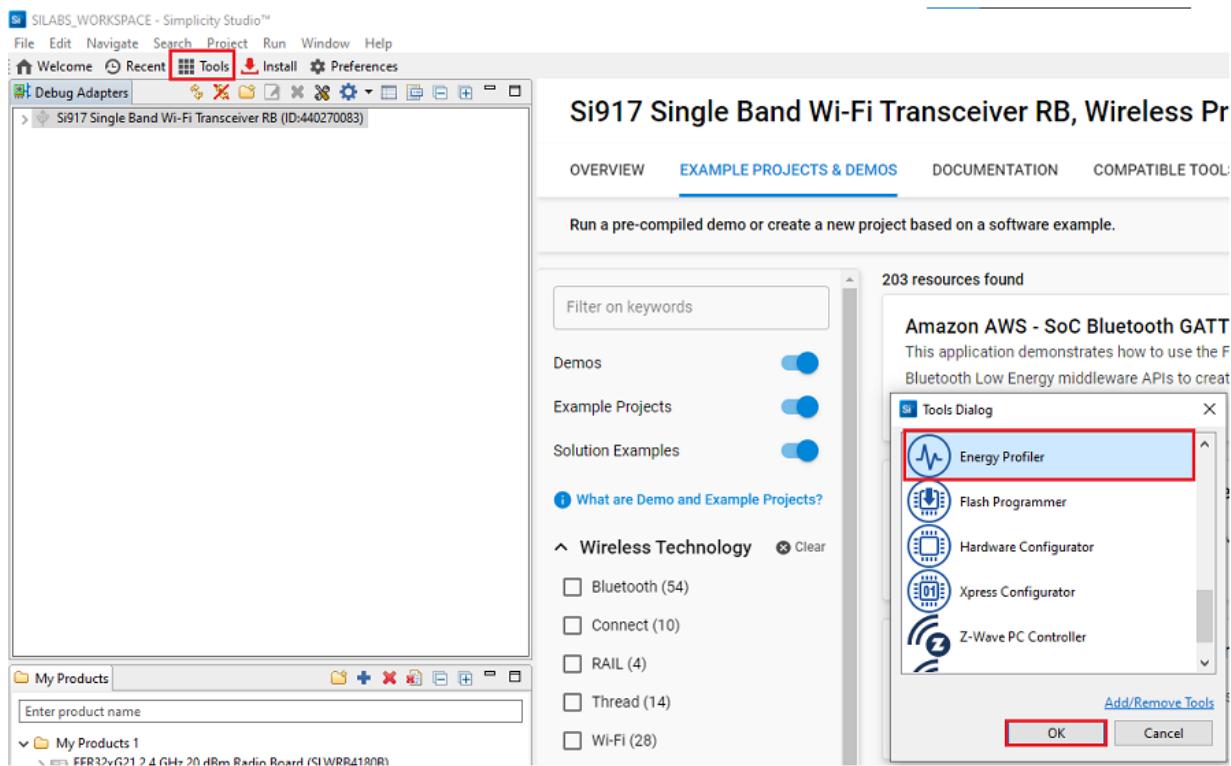
This section explains how to measure the power values for EFR Wi-Fi and SOC Wi-Fi co-processor.

Using Simplicity Studio Energy Profiler for Current Measurement

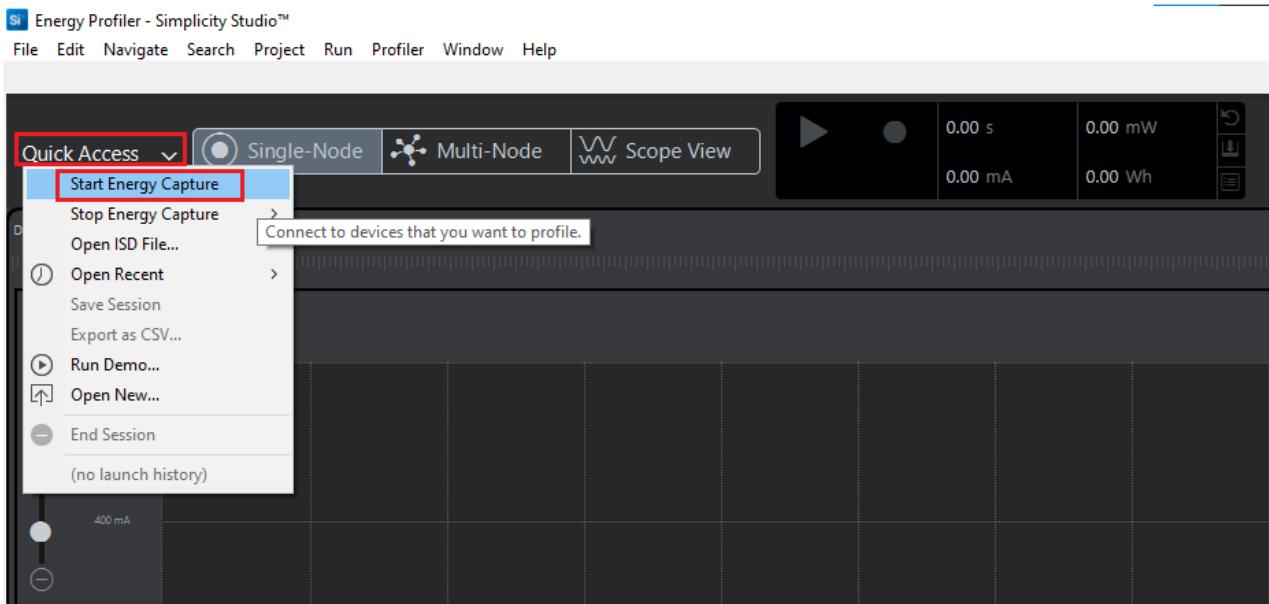
After flashing the Matter application to the module, Energy profiler or a power meter can be used for power measurements.



In Simplicity Studio, click **Tools** on the toolbar, select **Energy Profiler**, and click **OK**.



From the Quick Access or Profiler menu, select **Start Energy Capture**.



Note: A quick-start guide on the Energy Profiler user interface is in the Simplicity Studio User's Guide's [Energy Profiler User Interface section](#).

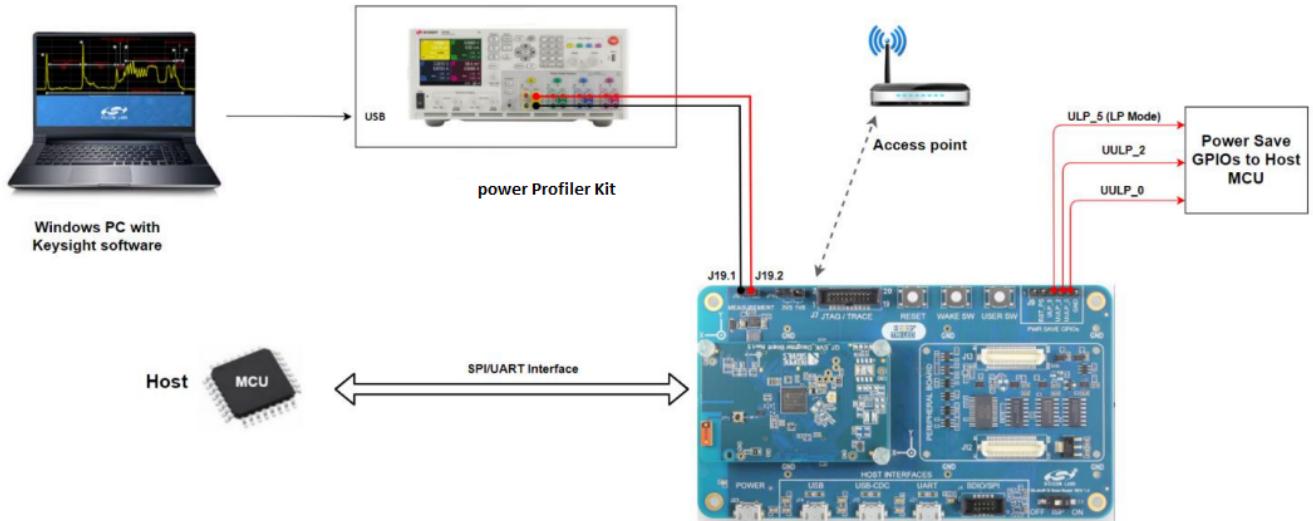
Power Consumption Measurement Using Energy Profiler for Wi-Fi Devices

Analyze the power values using Energy Profiler.

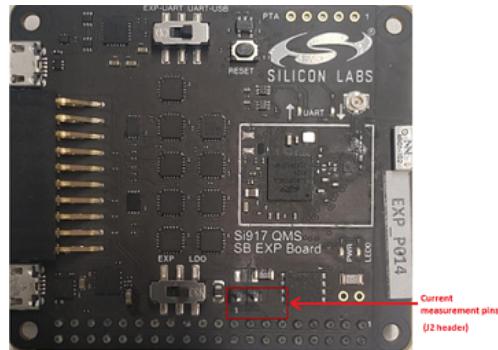


Power Consumption Measurement Using a Power Meter

Power consumption measurement pins for RS9116 Evk Boards:

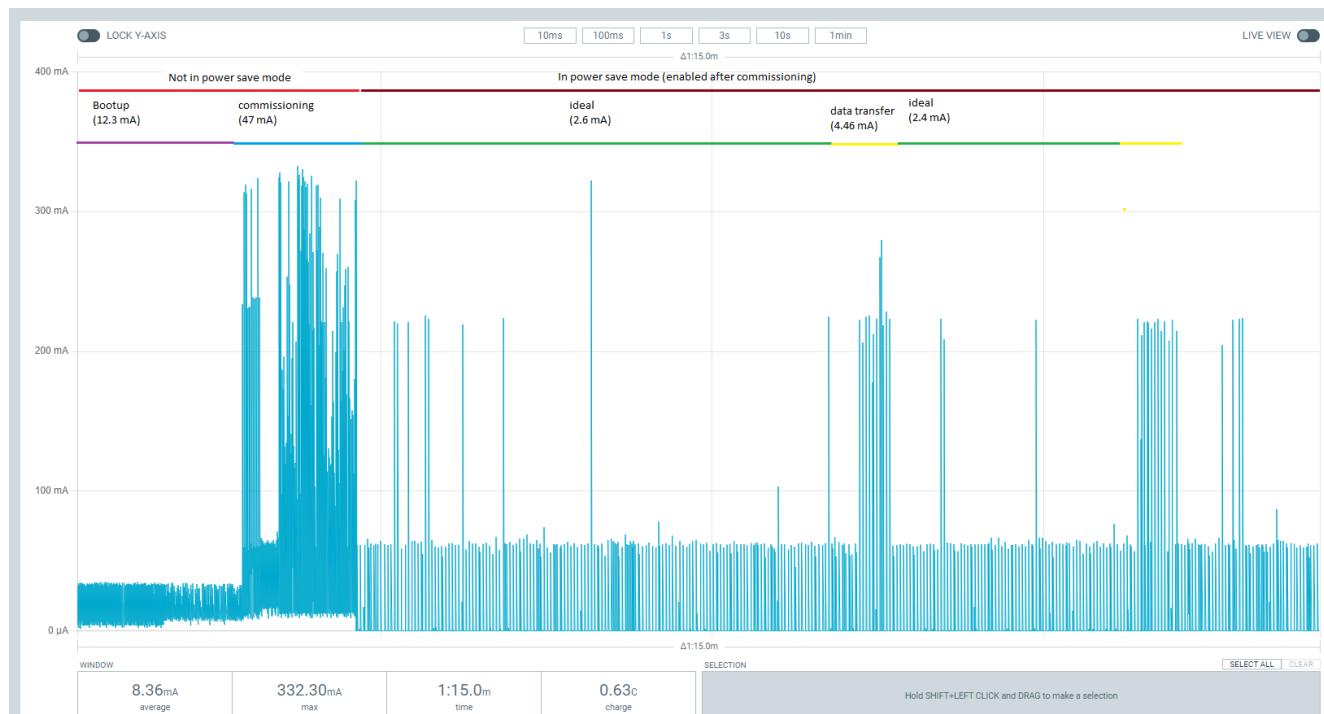


Power consumption measurement pins for EXP Boards:



The power meter's negative probe is used for pin-1 and the positive probe is used for pin-2.

Analyze the power values using the power meter.



Direct Internet Connectivity

Matter Wi-Fi Direct Internet Connectivity

See the [Matter over Wi-Fi DIC page](#).

Interoperability with Ecosystems

Matter Ecosystems

- Single Controller Configuration
- Google Ecosystem Setup
- Apple Ecosystem Setup
- Amazon Ecosystem Setup
- Samsung Ecosystem Setup
- Multi-Controller Configuration

Optimizing Memory Usage

Optimizing Memory Usage

This page provides information on optimizing memory usage for Silicon Labs devices.

How to Optimize Memory

To optimize an application's memory footprint, check the following:

1. Analyze and reduce stack usage of the application wherever possible.
2. Disable any included debug modules.
3. Turn off unused peripherals, features, and functions.

Disabling Debug Logging

Memory can be optimized by disabling debug logs. The `matter_no_debug` component disables the following from the project.

- Disables Silabs specific logging used in Matter
- Disables Hard Fault logs
- Keeps Log Level to None

To add `matter_no_debug` Component, modify the corresponding app-specific `.slcp` project file.

```
- id: matter_no_debug
from: matter
```

Optimizing ICD Power Consumption

Optimizing Power Consumption for Intermittently Connected Devices (ICD)

This page provides information on optimizing power consumption for Intermittently Connected Devices (ICD) formerly called Sleepy End Devices.

Minimal Power Consumption

Simply enabling ICD functionality does not give the application the best power consumption. By default, several features that increase power consumption are enabled in the example applications.

To achieve the most power-efficient build, the following components need to be disabled. The `matter_platform_low_power` component will do these steps for you, if installed.

- Matter Shell (`matter_shell`)
- LCD (`matter_lcd`) and Qr Code (`matter_qr_code`)

Note: `matter_shell` is not enabled by default in project file.

- Add `matter_shell` component in project file to enable the matter shell feature (for Wi-Fi non-ICD apps).
- Remove `matter_shell` while enabling ICD apps.

Flow of the Matter Wi-Fi App with LCD Configuration

- A timer starts in the Start of App Task. If there is NO activity after `defaultTimeoutMs`, the callback is triggered and the LCD will go to Sleep.
- In between, if the user presses BTN1, the LED1 will toggle as usual and the LCD screen will be enabled. After `timeoutMs` it will trigger the callback and disable the LCD.
- If the User presses BTN0, the system will check if the device is already commissioned. If it is not commissioned, the display will be enabled, it will toggle the QR Code, and the call back function will be triggered after `QRtimeoutMs`.

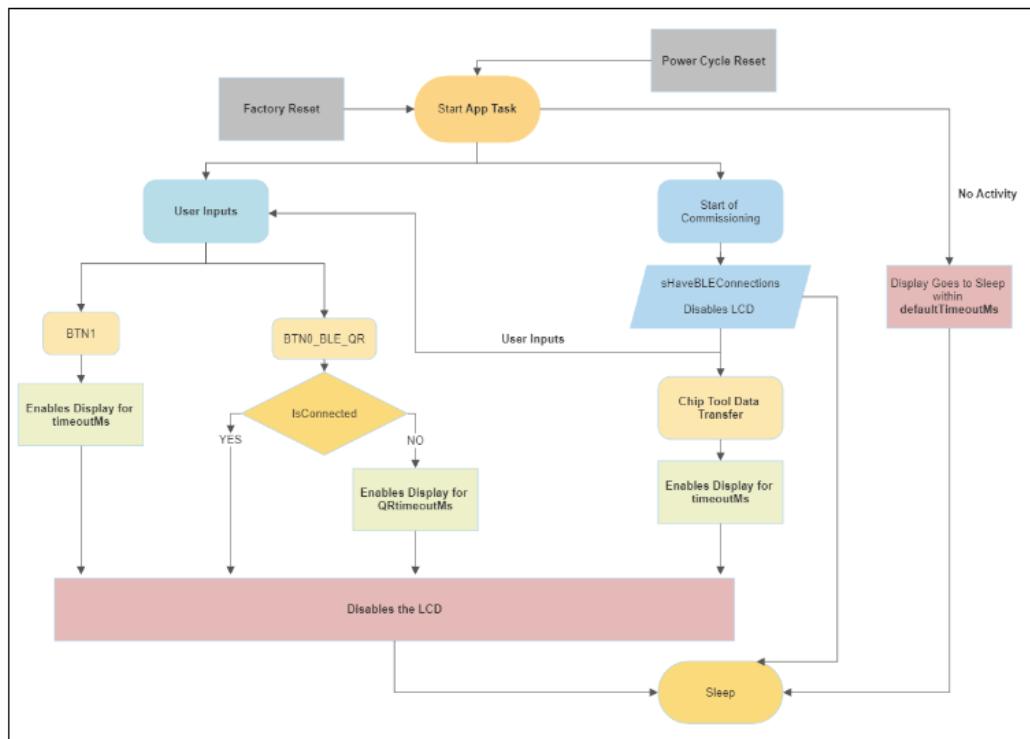
Start of Commissioning

- At the start of commissioning the display remains enabled. On pressing BTN0 the user can see the QR code to commission for a period of `QRtimeoutMs`.
- Once the commissioning process starts, the LCD screen will be disabled.

After Commissioning

- The LCD display is off during inactive transmissions.
- The LCD display active if there is any BTN press or data transfer.
- On pressing the BTNs, it will work the same way as before.
- On initiating Data Transfer, once the action is initiated, the LCD display will be enabled and disabled again after the specified time.
- On triggering Factory Reset, the LCD will be enabled for `QRtimeoutMs`, then it will be disabled.

The following diagram shows the end-to-end flow for optimizing power consumption:



Power Save Methods

For information on power saving, refer to [Power Save Methods](#).

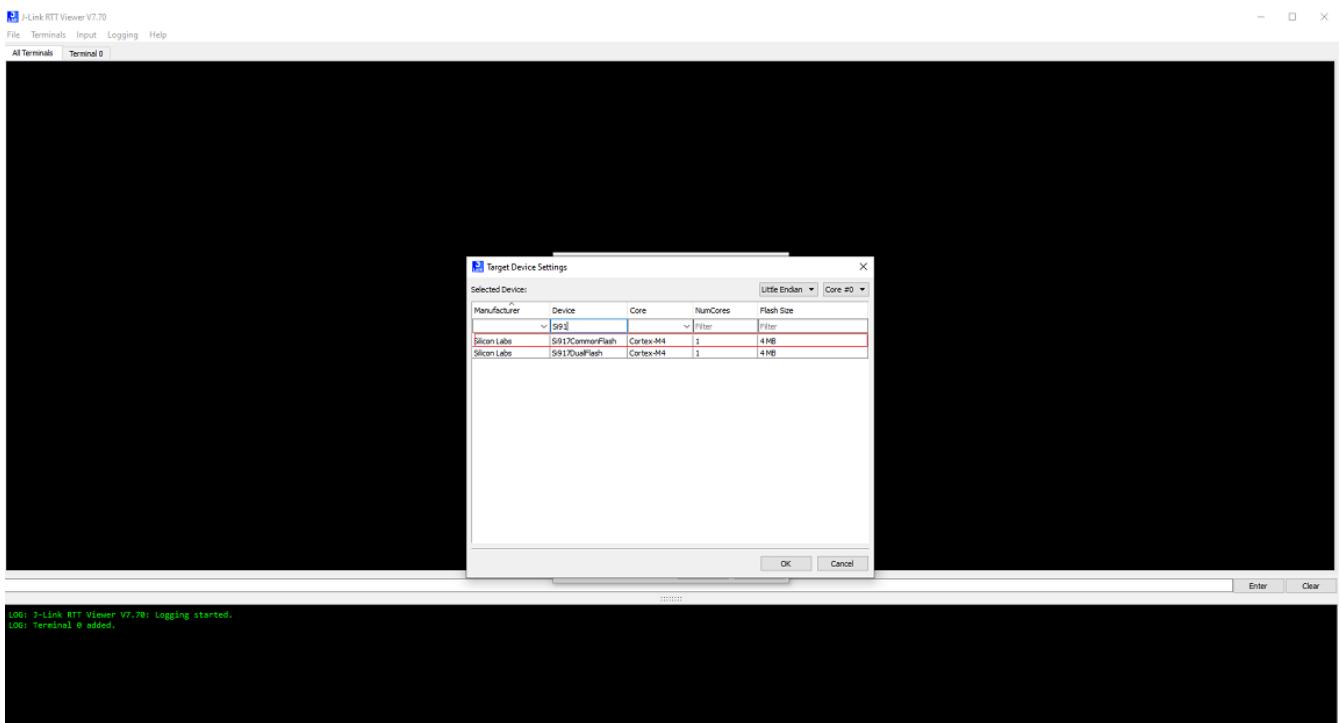
Jlink RTT Support with SOC

JLink RTT Environment Setup for a SiWx917 SoC Device

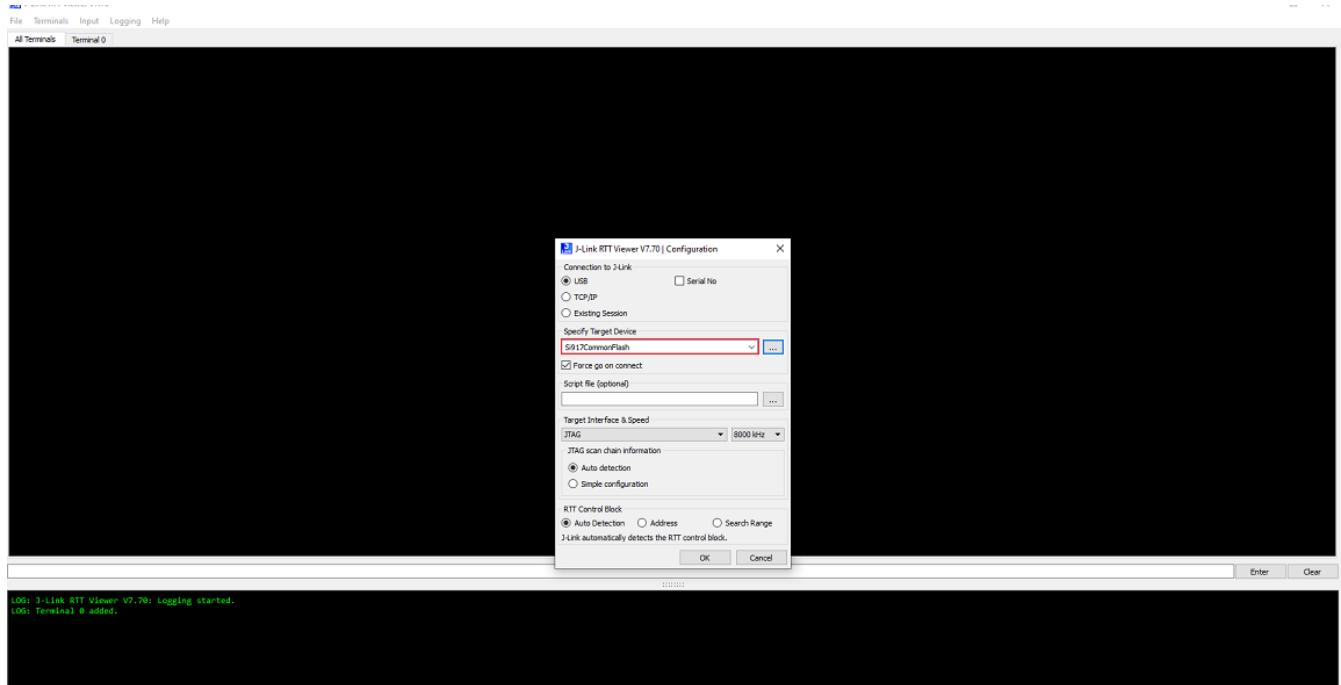
- For 917 SOC Common Flash Boards, Ozone debugging support is not enabled. To get the logs for 917 SOC, the JLink RTT tool will be used.
- Auto detection of SiWx917 SoC device in **JLink RTT** is not enabled.
- Follow the steps to configure the SiWx917 SoC with Latest **JLink RTT Logging** tool.

Steps to Configure the SiWx917 SoC on the JLink-RTT Logging

1. Update the **JLinkDevices.xml** and **.elf** files found on the [Matter Artifacts Page](#).
 - Download the **JLinkDevices.xml** file and copy it in your **JLink RTT** installation path shown in this [JLinkDevices Folder](#). If there is no **JLinkDevices** Folder, create a **JLinkDevices** folder and copy the **JlinkDevices.xml** file into it.
 - In the **JLinkDevices** folder, create a **Devices** folder and then create a sub-folder named **SiliconLabs**.
 - Download the **.elf** file (Flash driver) and copy it in the created **SiliconLabs** folder.
2. Launch **JLink RTT**. The SiWx917 Common Flash SoC device should be visible in the Device field's selection list.



3. Select **SI917COMMONFLASH** and click **OK**.



Direct Internet Connectivity

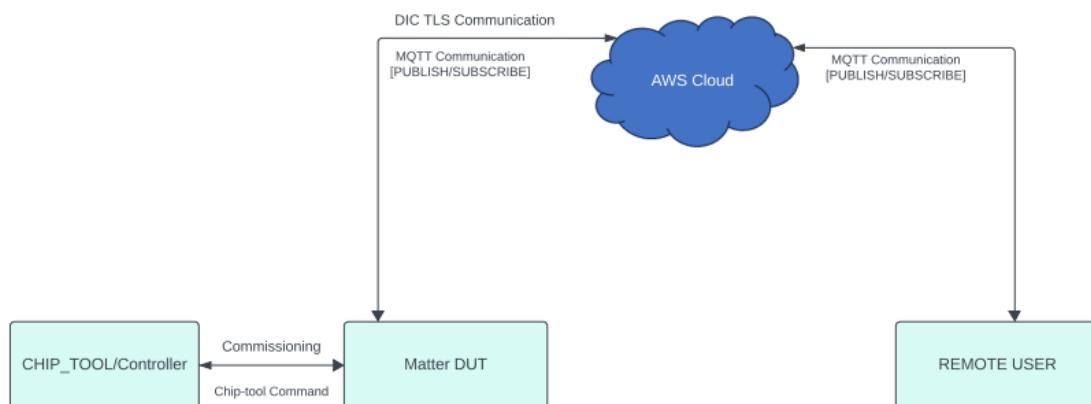
Matter Wi-Fi Direct Internet Connectivity

Direct Internet Connectivity (DIC) is a Silicon Labs-only feature to connect Matter devices to proprietary cloud solutions (eg, AWS,GCP,Apple) directly. As such, a Matter Wi-Fi device must support connecting locally on the Matter Fabric, via IPv6, and connecting to the Internet via IPv4.

- Matter devices can be controlled by chip-tool or controller and the respective status of the attribute modified will be published to the cloud.
- Remote users can install the cloud-specific application to get the notifications on the attribute status and to control the device.

DIC Feature Diagram

The following diagram illustrates the end-to-end flow for Direct Internet Connectivity.



Prerequisites

Hardware Requirements

For the hardware required for the DIC feature to run on the Silicon Labs Platform, refer to [Matter Hardware Requirements](#).

Software Requirements

To run the DIC feature, refer to [Software Requirements](#).

End-to-End Set-Up Bring Up

Message Queuing Telemetry Transport (MQTT)

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. Refer to <https://mqtt.org/> for more details.

Configuring the MQTT server

To set up and configure AWS or Mosquitto for DIC support, see the following documentation:

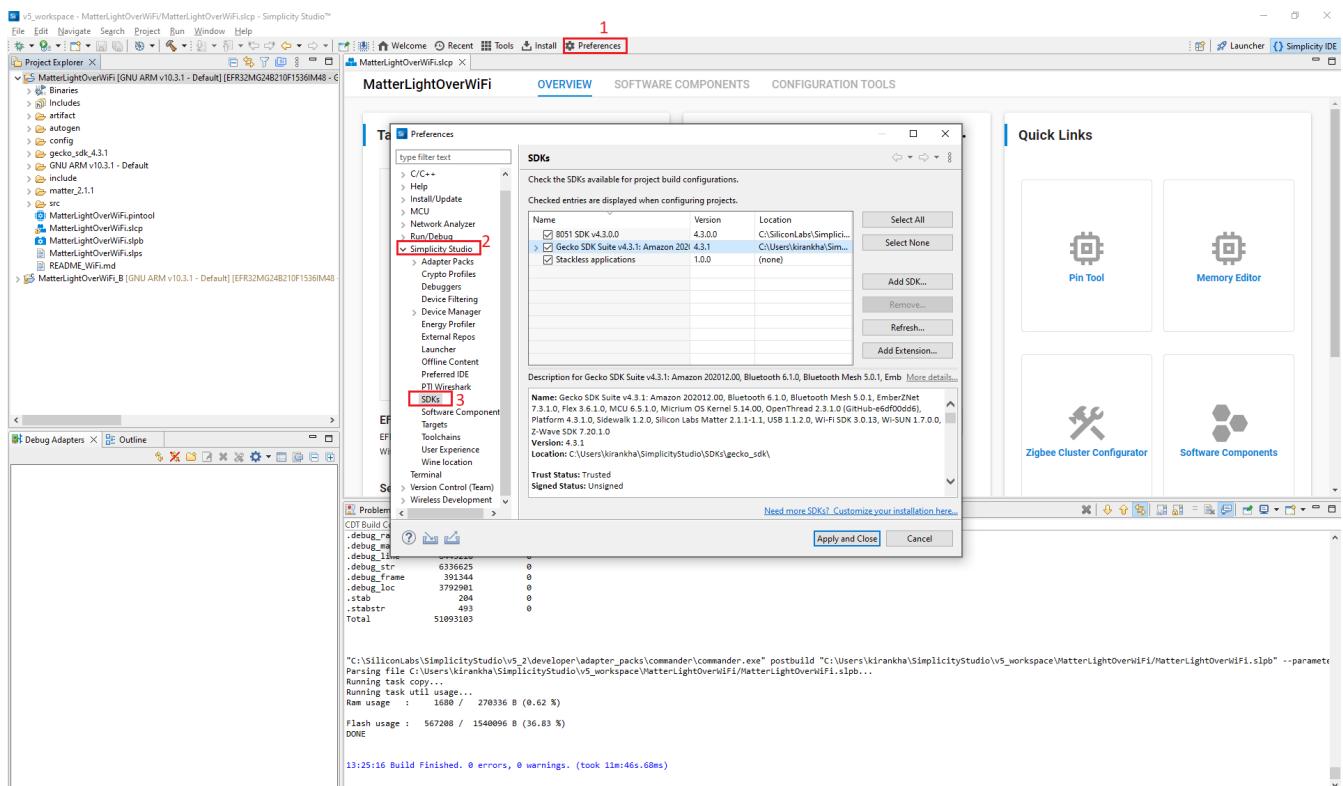
- [AWS installation](#)
- [Mosquitto installation](#)

Remote User Setup (MQTT Explorer)

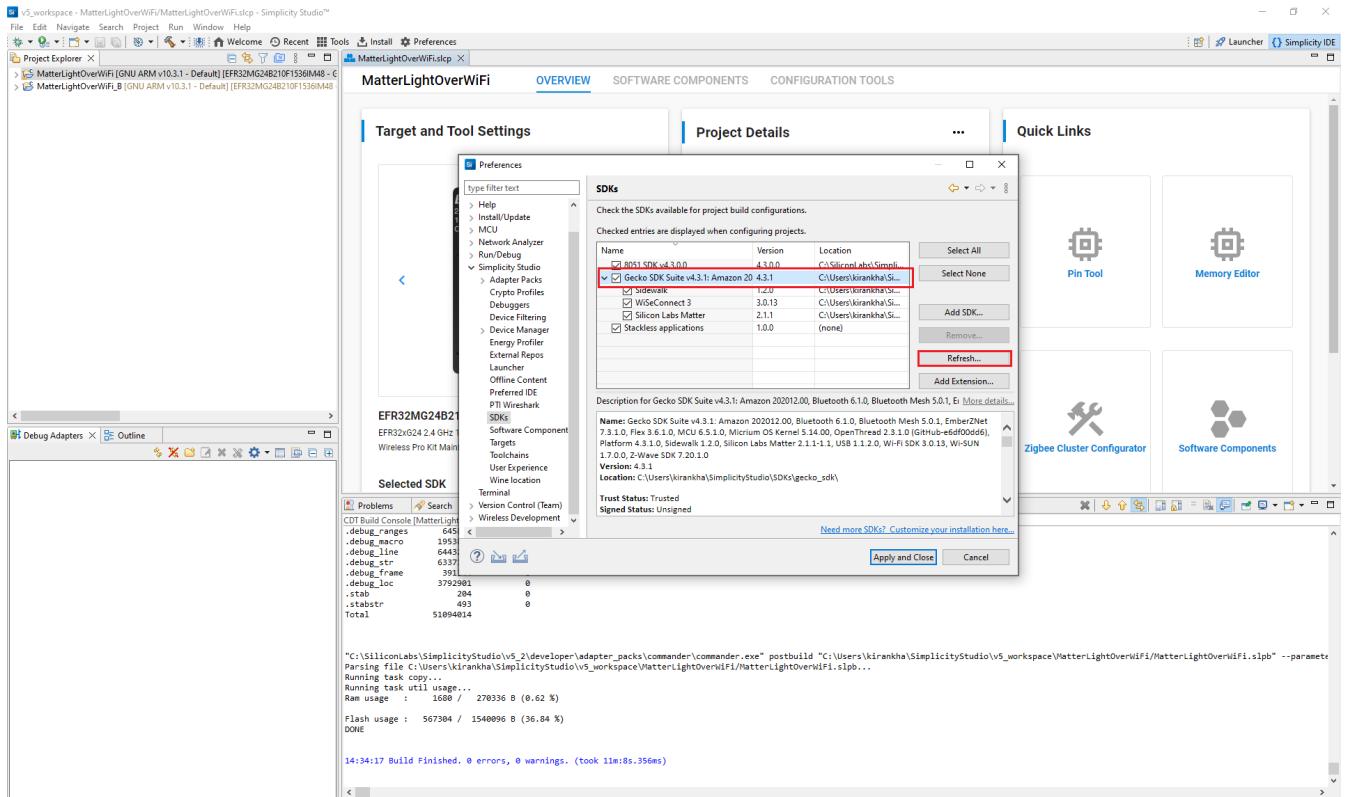
A remote user is used to check the state of a Matter device. In this context, MQTT explorer is used as a remote user. See [MQTT explorer setup and configuration](#).

Building Matter DIC Application using Simplicity Studio

1. Follow [Build DIC](#) to enable DIC feature in code.
2. After Enabling DIC feature in the Matter extension code, click **Preferences** and go to **SDKs tab** in Simplicity Studio.



3. In the **SDKs** tab, click **Gecko SDK** and click **Refresh**. It will refresh matter extension code for changes made in step 1.



4. After refreshing the Matter extension, create and build a project for the Silicon Labs Device Platform. Refer the following:

- [Creating and Building Project for NCP Board](#)
- [Creating and Building Project for SoC Board](#)

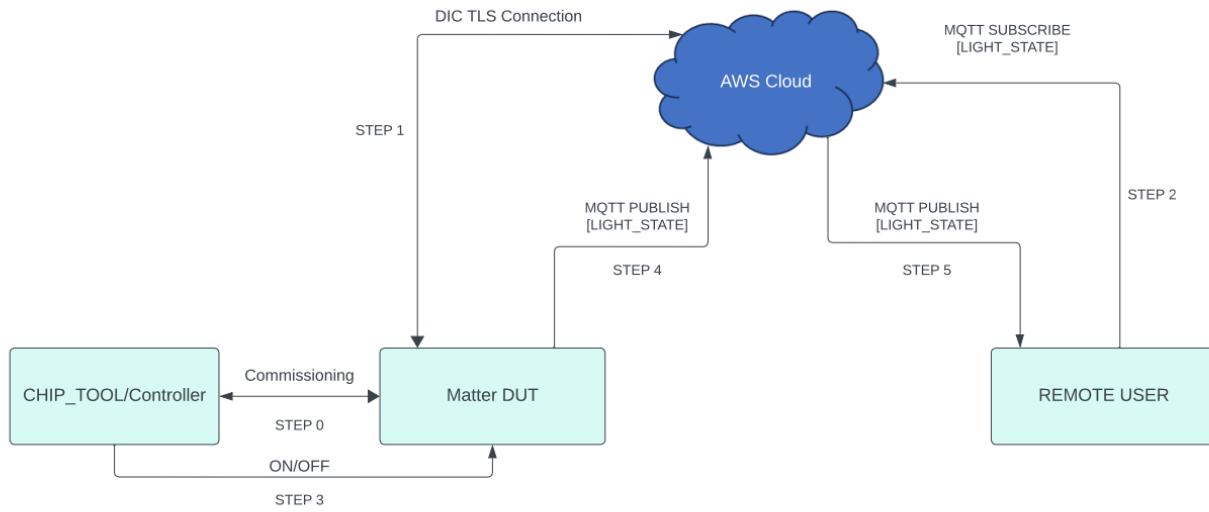
Note: Matter extension code is located at the following location:

C:\Users\system_name\SimplicityStudio\SDKs\gecko_sdk\extension .

End-to-End Test of DIC Application

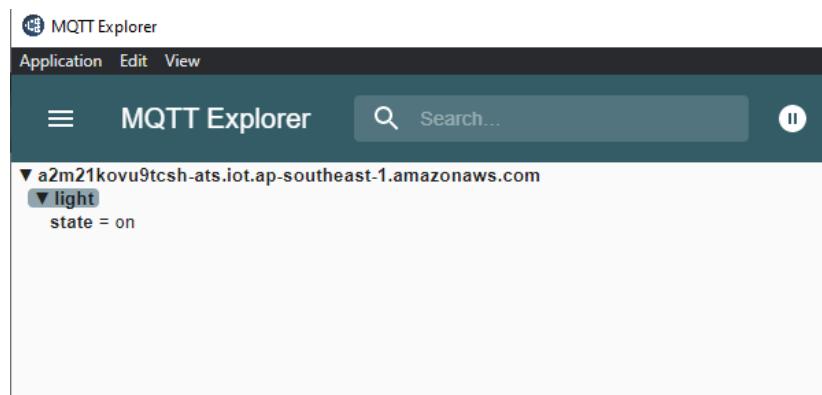
User Setup (MQTT Explorer):

- Sharing status of device to cloud
- The following diagram shows the end-to-end flow for sharing status from a Matter device to the Cloud.

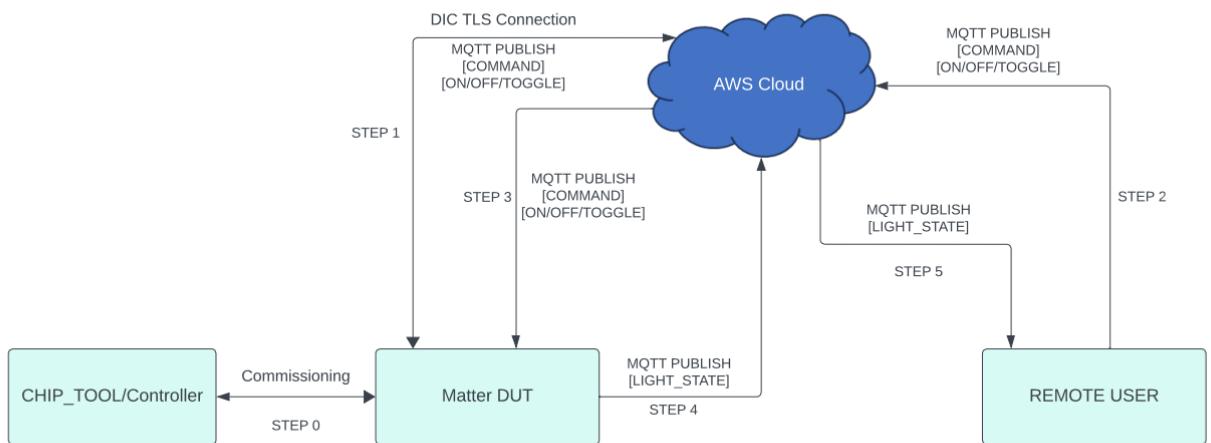


Note: For reference, Lighting App commands are given in the above image. Other application commands also can be passed.

- For the end-to-end commands to be executed from chip-tool, refer to [Running the Matter Demo Over Wi-Fi](#).
- Below are the application-specific attribute/s information or state shared to the cloud:
 - For Lighting App, On/Off Attributes
 - For Lock App, lock/unlock Attributes
 - For Windows App, lift/tilt Attributes
 - For Thermostat App, SystemMode/CurrentTemp/LocalTemperature/OccupiedCoolingSetpoint/OccupiedHeatingSetpoint Attributes
 - For On/off Plug App, On/Off Attributes
- Application status would be updated on the mqtt_explorer UI, as shown in below image.



- Control of the device through cloud interface
 - Below diagram gives end-to-end flow for Control of the matter device through cloud interface



Note: For reference, Lighting App commands given in the above image. Similarly other application commands also can be passed.

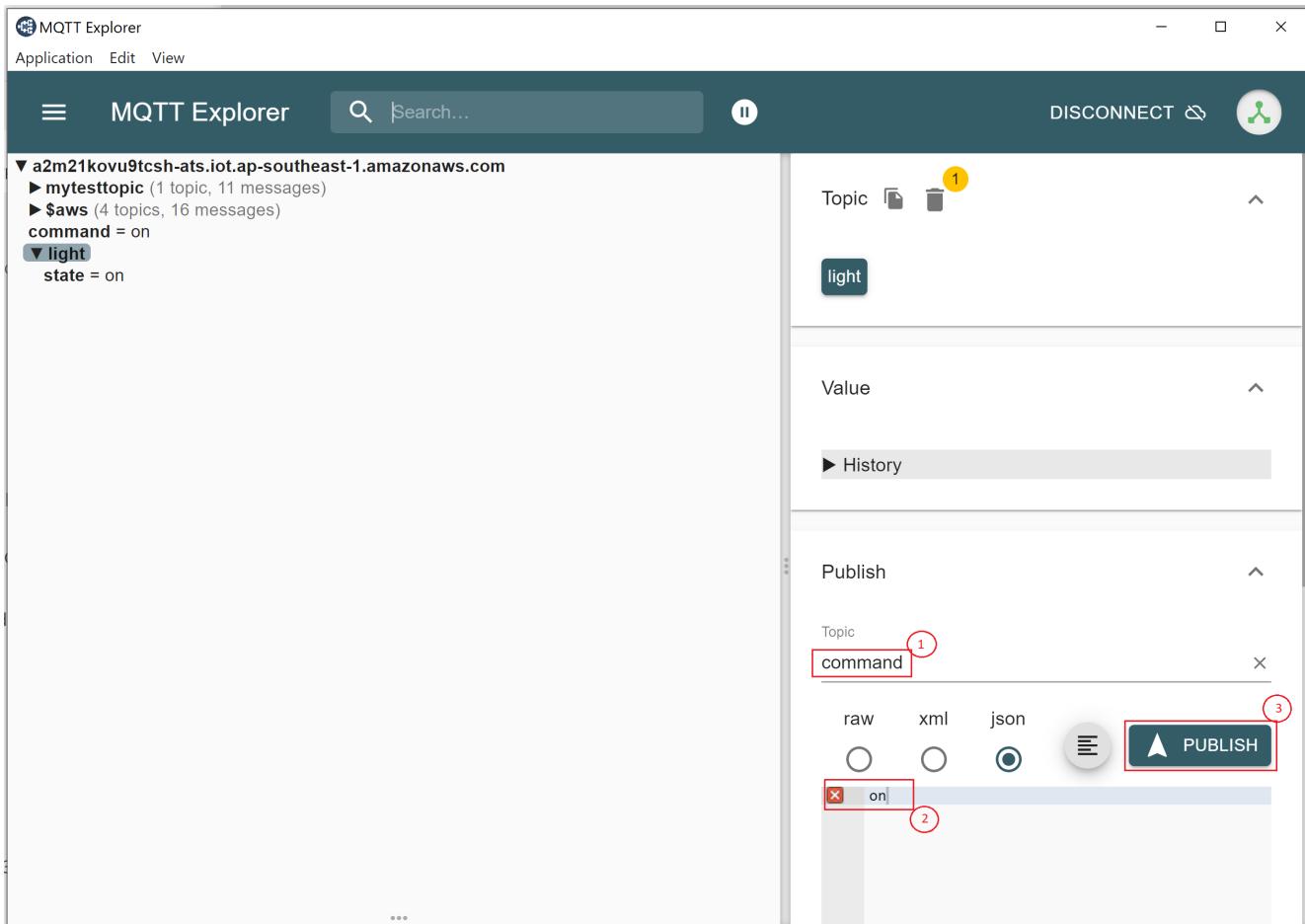
- Make sure matter device is up and commissioned successfully, refer to [Running the Matter Demo Over Wi-Fi](#)
- For controlling the device, set topic name and the commands to be executed in the mqtt_explorer for below applications.

```

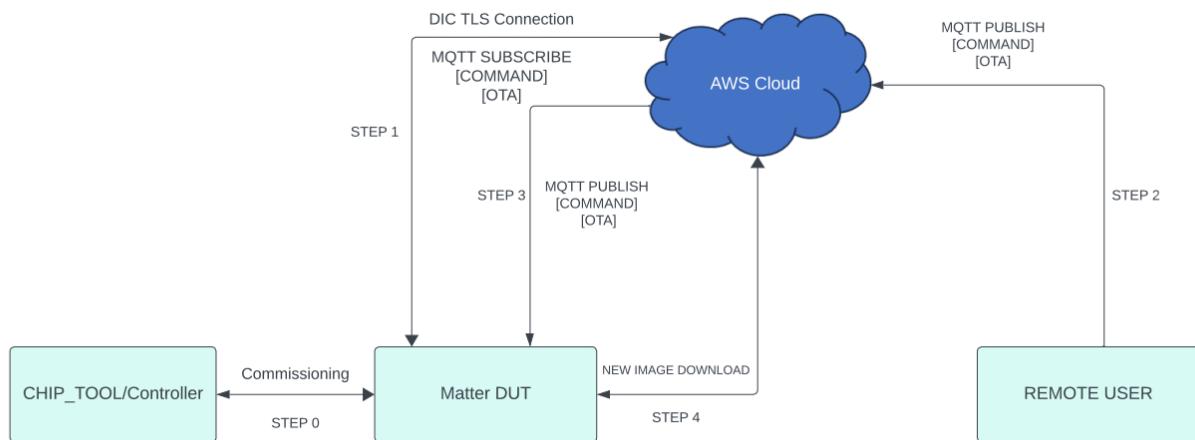
- Lighting App
- Topic: command
- Commands:
  - toggle
  - on
  - off
- Onoff-plug App
- Topic: command
- Commands:
  - toggle
  - on
  - off
- Lock App
- Topic: command
- Commands:
  - lock
  - unlock
- Thermostat App
- Topic: command
- Commands:
  - SetMode/value(value need to provide 1,2,3,4 ex:SetMode/1)
  - Heating/value(value need to provide 2500,2600 ex:HeatingSetPoint/2500)
  - Cooling/value(value need to provide 2500,2600 ex:CoolingSetPoint/2500)
- Window App
- Topic: command
- Commands:
  - Lift/value(value need to provide in range 1000 to 10000)
  - Tilt/value(value need to provide in range 1000 to 10000)

```

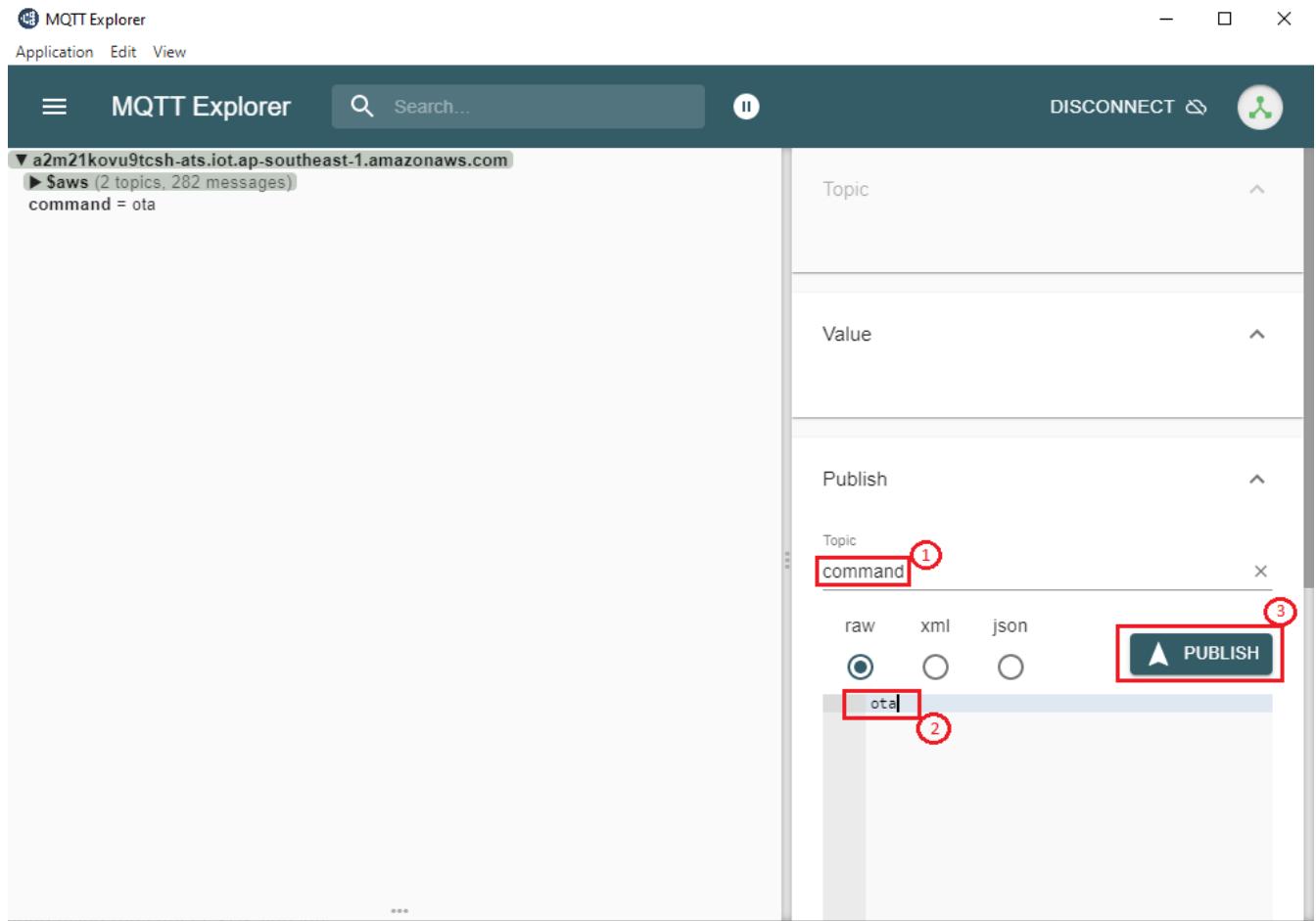
- Click Publish to execute the command.



- Download AWS OTA Image through cloud interface.
 - The diagram below provides the end to end flow of firmware upgrade feature through AWS.



- Make sure Matter device is up and commissioned successfully. Refer to [Running the Matter Demo Over Wi-Fi](#).
- Make sure device is connected to MQTT Server successfully.
- Create an AWS OTA Job on the AWS website. Refer to [How to create AWS OTA job](#).
- Trigger OTA Command through MQTT Explorer like below.
- Click **Publish** to execute the AWS OTA command.



AWS Configuration Registration

Amazon Web Services (AWS)

Amazon Web Services offers reliable, scalable, and inexpensive cloud computing services. Refer to [AWS Documentation](#) for more details.

AWS CA Certificate Registration

1. Create a CA certificate, a client certificate and a client key using the [Openssl Certificate Creation](#) documentation.
2. Open [AWS](#).
3. Login using your AWS credentials.
4. Register the CA Certificate in AWS:
 - o Go to Security -> Certificate Authorities and Register CA Certificate .
 - o Select Register CA in the Multi-account mode.
 - o Choose the CA certificate that you previously created in the Openssl Certificate Creation (CA.crt) step in the CA certificate registration, and set the CA status to Active and the Automatic certificate registration option to ON .
 - o Register the CA.
5. Go to Security -> Policies and select Create Policy . Enter the policy name (ex: DIC_POLICY) and in the policy statements select JSON and replace the contents with the JSON provided below:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

1. Once done, select **Create**.
2. Steps to generate the certificate for your Matter application to use in the dic_nvm_cert.cpp source file.
 - o Go to All Devices -> Things and select **Create Things**.
 - o Select **Create Single Thing** and click **Next**.
 - o Specify thing properties Info -> Give the thing a name (Note: Client ID) and click **Next**.
 - o Configure the device certificate - optional Info -> Use my certificate.
 - o Certificate details -> Choose **CA is registered with AWS IOT** and Select the CA that registered with AWS in Step 4.
 - o Certificate -> Choose file (Choose Client certificate generated in Openssl Certificate Creation ex: device.crt) and set the certificate status to **Active**. Click **Next**.
 - o Use the policy(ex: DIC_POLICY) created in AWS Certificate Creation.
3. Repeat Step 5 to create a new thing to use in MQTT Explorer using the certificate created for MQTT explorer (from Openssl Certificate Creation ex: explorer.crt)

Note: Thing name must be unique as it will be used as CLIENT ID.
4. Copy the contents of [AWS_CA CERT](#) and create a .pem file to use as a SERVER CERTIFICATE in MQTT Explorer.

How to create AWS OTA JOB

1. Go to AWS Amazon link <https://aws.amazon.com/>.
2. Login with Amazon Credentials.

3. Click on Services and select **IOT Core**.
4. On the side menu in Manage Section, click **Remote Actions** and click **jobs**.
5. Click **Create Job** and select Job type as a Create FreeRTOS OTA update job.
6. Enter a unique Job name without spaces.
7. In the **Devices to update** dropdown, select your Certificates which is configured above. for example:- SQA_DIC_C2, SQA_DIC_C3, DIC_2
8. Select **MQTT** as the protocol for file transfer.
9. In **File Section**, select **New/Previously/Custom** signed **gbl**(For EFR32) and **.rps**(For 917 SOC) file.
 - If the **gbl or rps** file is newly created, then select **Sign a new file for me**.
 - If the **gbl or rps** file is already uploaded to AWS, then select **Choose a previously signed file**.
 - If the **gbl or rps** file is custom modified, then select **Use my custom signed file**.
10. In **Existing code signing profile**, select **dic_ota_codesign**. Refer to [AWS Code Signing Certificate Creation](#).
11. For uploading the **gbl or rps** file, follow step 9 above. To create a **gbl** refer to [Matter OTA](#) and for **rps** file, refer to [Matter OTA](#).
12. In the File upload location in S3 select, S3 URL as ota_demo. Refer to [AWS S3 bucket Creation](#).
13. In **Path name of file on device**, give any file name (file.txt).
14. Select **ota_demo** as **IAM role** and click **Next**.
15. Click **create job**.

Note: For more details, Refer [AWS OTA prerequisites](#)

OpenSSL Certificate Creation

Openssl Certificate Creation

An SSL certificate is an important way to secure user information and protect against hackers.

Openssl Installation (In ubuntu)

1. To install openssl - `sudo apt install openssl`

Certificates Creation

The following commands are used to generate certificates:

1. To generate CA key:
 - `openssl ecparam -name prime256v1 -genkey -noout -out CA.key`
2. To generate CA certificate:
 - `openssl req -new -x509 -days 1826 -key CA.key -out CA.crt`
3. To generate Client key:
 - `openssl ecparam -name prime256v1 -genkey -noout -out device.key`
4. To generate Client certificate (ex: `device.crt` and `device.key`) using CA certificate:
 - `openssl req -new -out device.csr -key device.key`
 - `openssl x509 -req -in device.csr -CA CA.crt -CAkey CA.key -CAcreateserial -out device.crt -days 360`
5. Repeat step 3 and 4 to create an additional set of certificate to use in MQTT explorer (ex: `explorer.crt` and `explorer.key`).
(Create with different name for Identification).

Mosquitto Installation

Mosquitto

Mosquitto is a lightweight open source (EPL/EDL licensed) message broker that implements the MQTT protocol. Refer to the [Mosquitto site](#) for more details.

Set Up the Mosquitto Connection

Linux Environment

1. Install Mosquitto, using the command --> `sudo apt install Mosquitto`
2. Copy file from <https://github.com/eclipse/Mosquitto/blob/master/Mosquitto.conf> and paste in linux machine.
3. Open the Mosquitto.conf file and include password_file (Mosquitto.pwd) in "General configuration".
4. The password file contains the username and password in hashed format. To create your own username and password for Mosquitto use the following [link](#).
5. In the section "Listeners" change listener `<port no.> <ip address of linux machine>`.
6. In the same section find `#protocol mqtt` and uncomment it.
7. Follow [Openssl Certificate Creation](#) to create certificates.
8. Provide the required certificates path in the "Certificate based SSL/TLS support" and in the same section set the flag `require_certification` to `true`.
9. In "Security" section change uncomment the flag `allow_anonymous` false.
10. Now that your configuration file is set, save it and run the following command in terminal to run Mosquitto:-
 - `Mosquitto -v -c Mosquitto.conf`

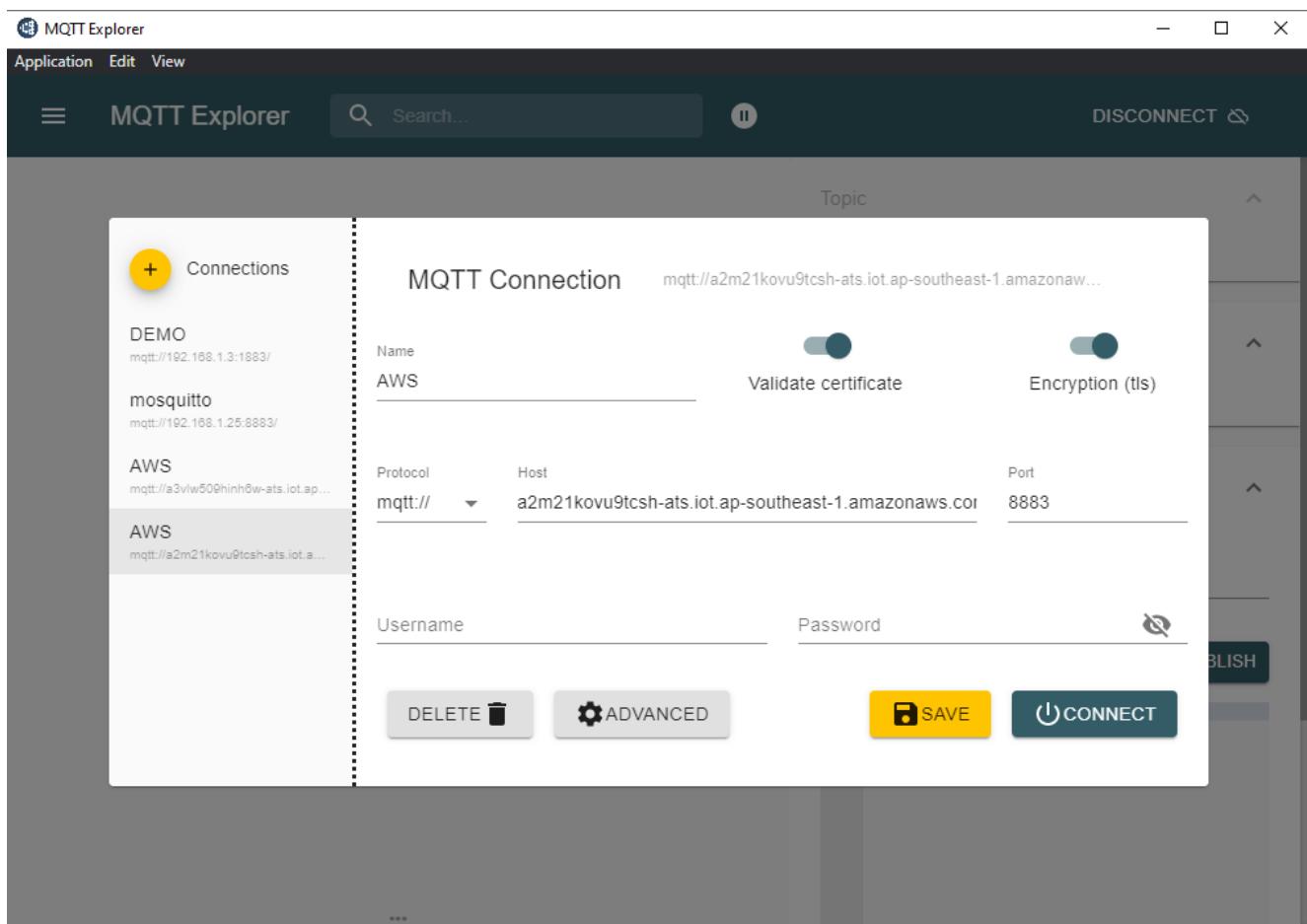
Windows Environment

1. Install Mosquitto using the [mosquitto download](#).
2. Open the Mosquitto.conf file and include password_file (Mosquitto.pwd) in "General configuration".
3. The password file contains the username and password in hashed format. To create your own username and password for Mosquitto use the following [link](#).
4. Next in section "Listeners" change listener `<port no.> <ip address of linux machine>`.
5. In the same section find `#protocol mqtt` and uncomment it.
6. Follow [Openssl Certificate Creation](#) to create certificates.
7. Provide the required certificates path in the "Certificate based SSL/TLS support" and in the same section set the flag `require_certification` to `true`.
8. In "Security" section change uncomment the flag `allow_anonymous` false.
9. Now that your configuration file is set save it and run the following command in terminal(command prompt) to run Mosquitto:-
 - `Mosquitto -v -c Mosquitto.conf`

MQTT Explorer Setup

Set Up MQTT Explorer

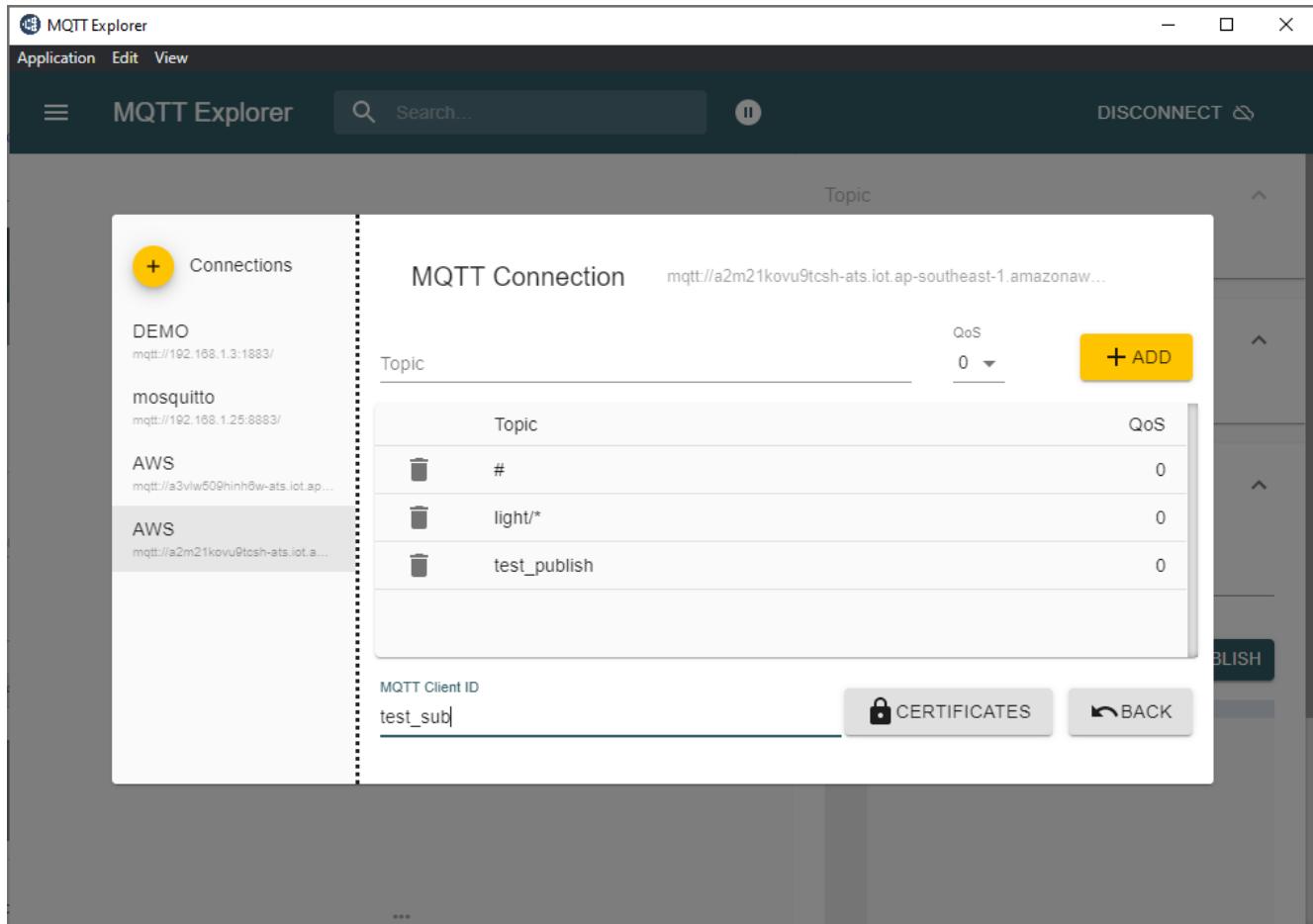
Download and install the MQTT Explorer from <https://mqtt-explorer.com/> Setting up MQTT Explorer for testing.



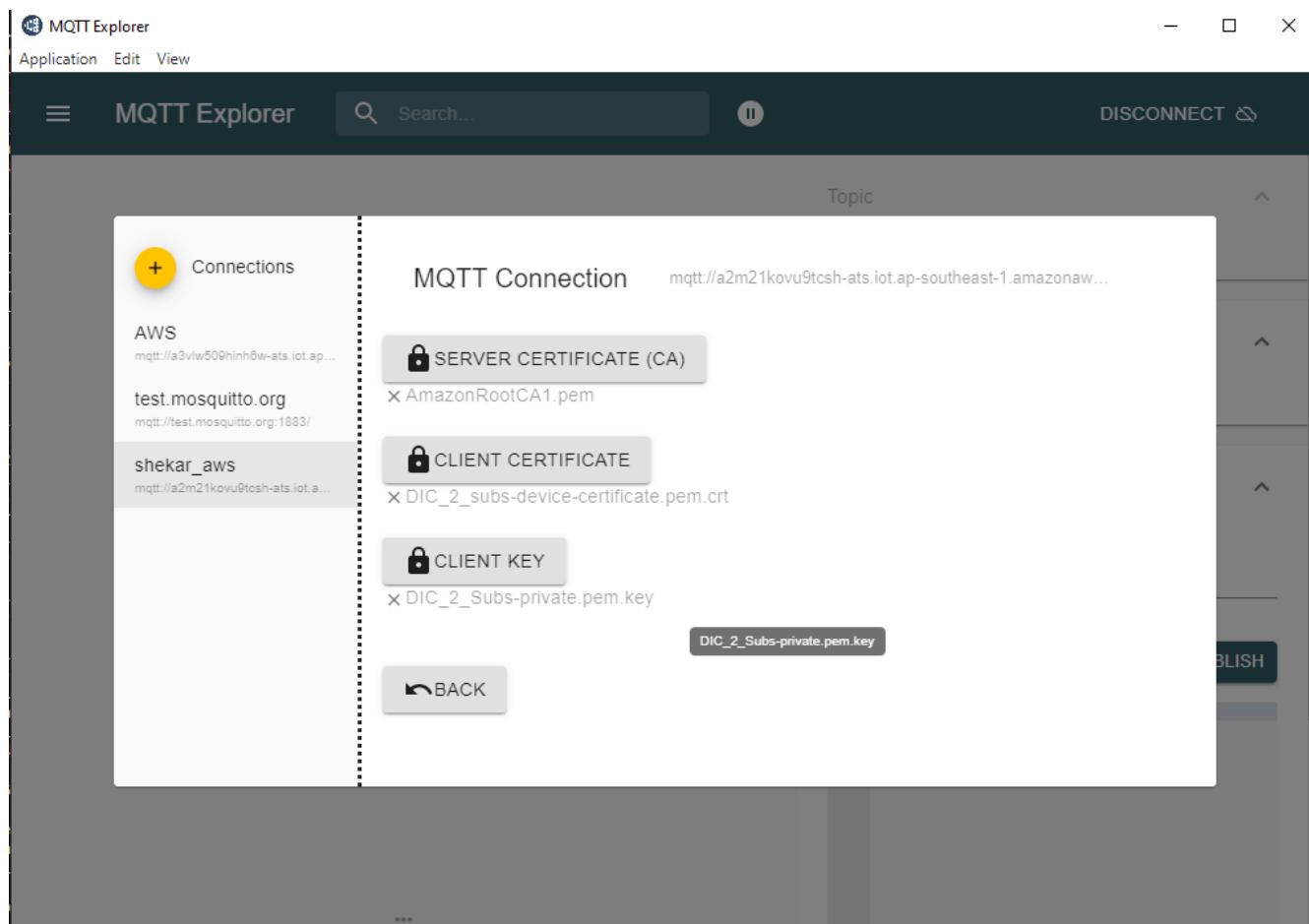
Connecting to MQTT Server

Connecting to AWS

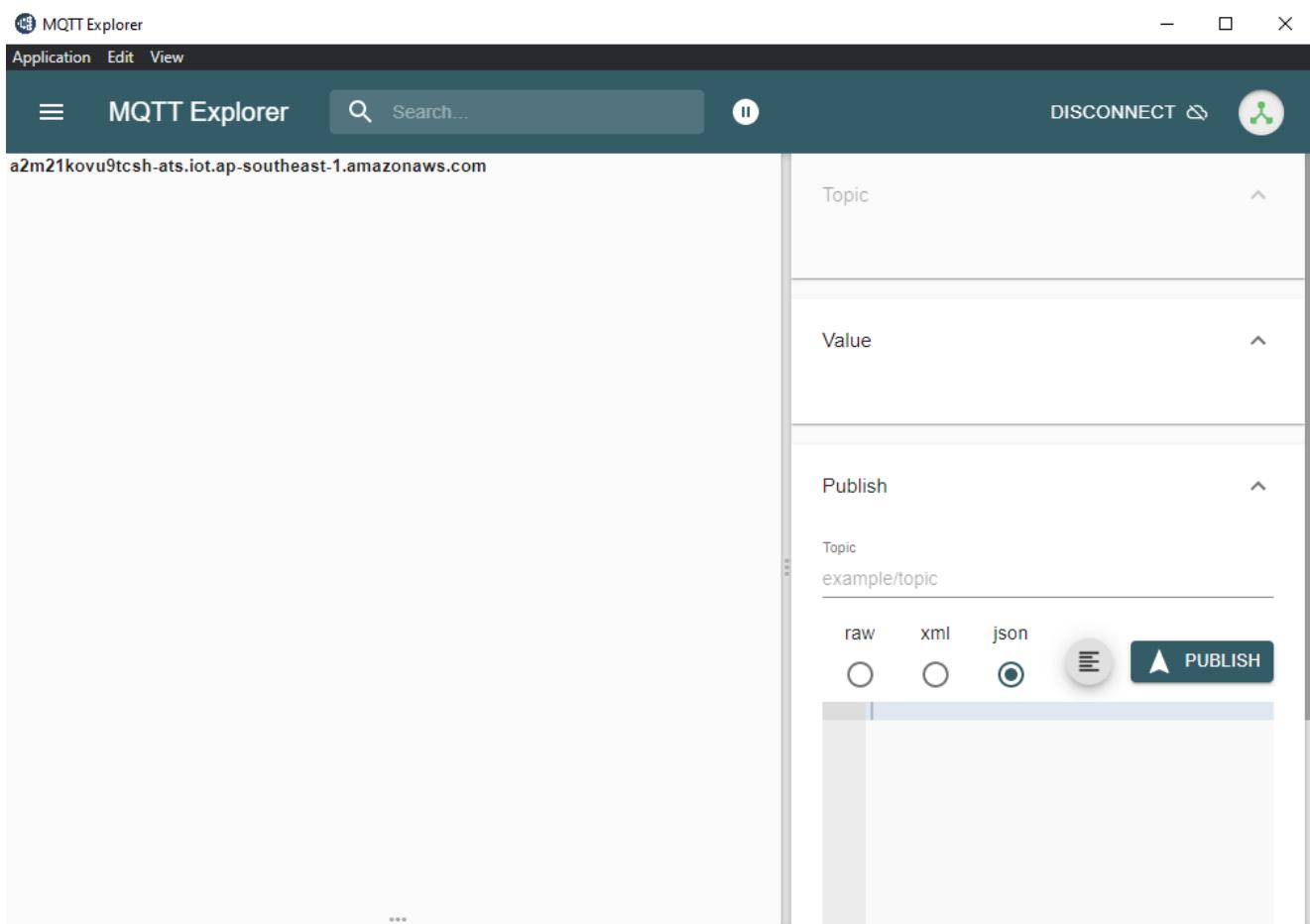
- Host: Your Host name (examples: a2m21kovu9tcsh-ats.iot.ap-southeast-1.amazonaws.com)
- Port: 8883
- Make sure you enable **Validate Certificate** and **Encryption**
- Click Advanced Settings



- Add application specific topics as shown below
 - For Lighting app, topic to be added (light/*)
 - For onoff plug app, topic to be added (light/*)
 - For Lock app, topic to be added (lock/*)
 - For thermostat app, below topics to be added
 - LocalTemperature/*
 - OccupiedCoolingSetpoint/*
 - OccupiedHeatingSetpoint/*
 - thermostat/*
 - For Windows app, below topics to be added
 - lift/*
 - tilt/*
- MQTT Client ID depends on the certificate set that you will use.
- Add the Certificate, following step 7 in [AWS installation](#).

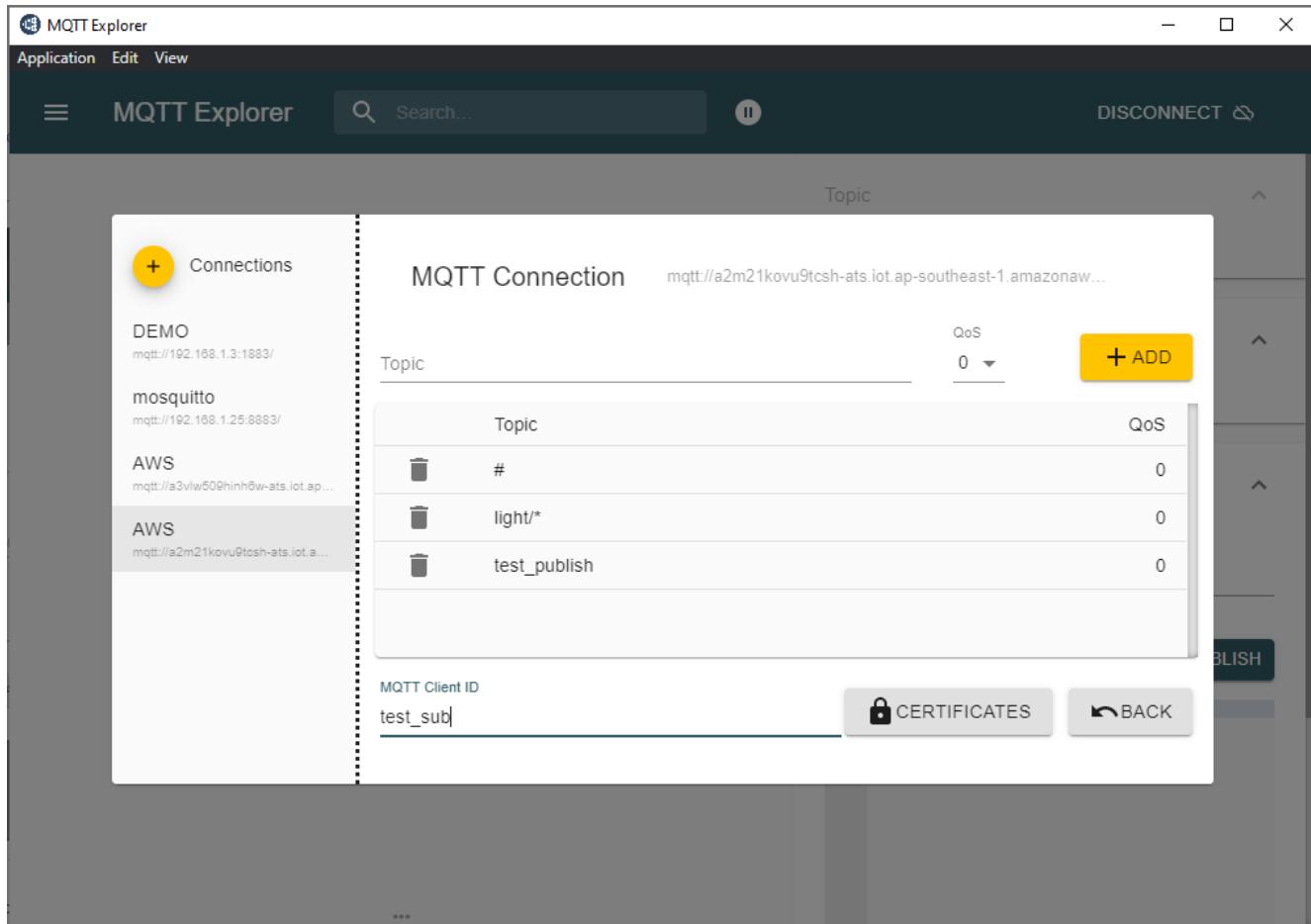


- Once the above steps are done, try connecting to AWS.

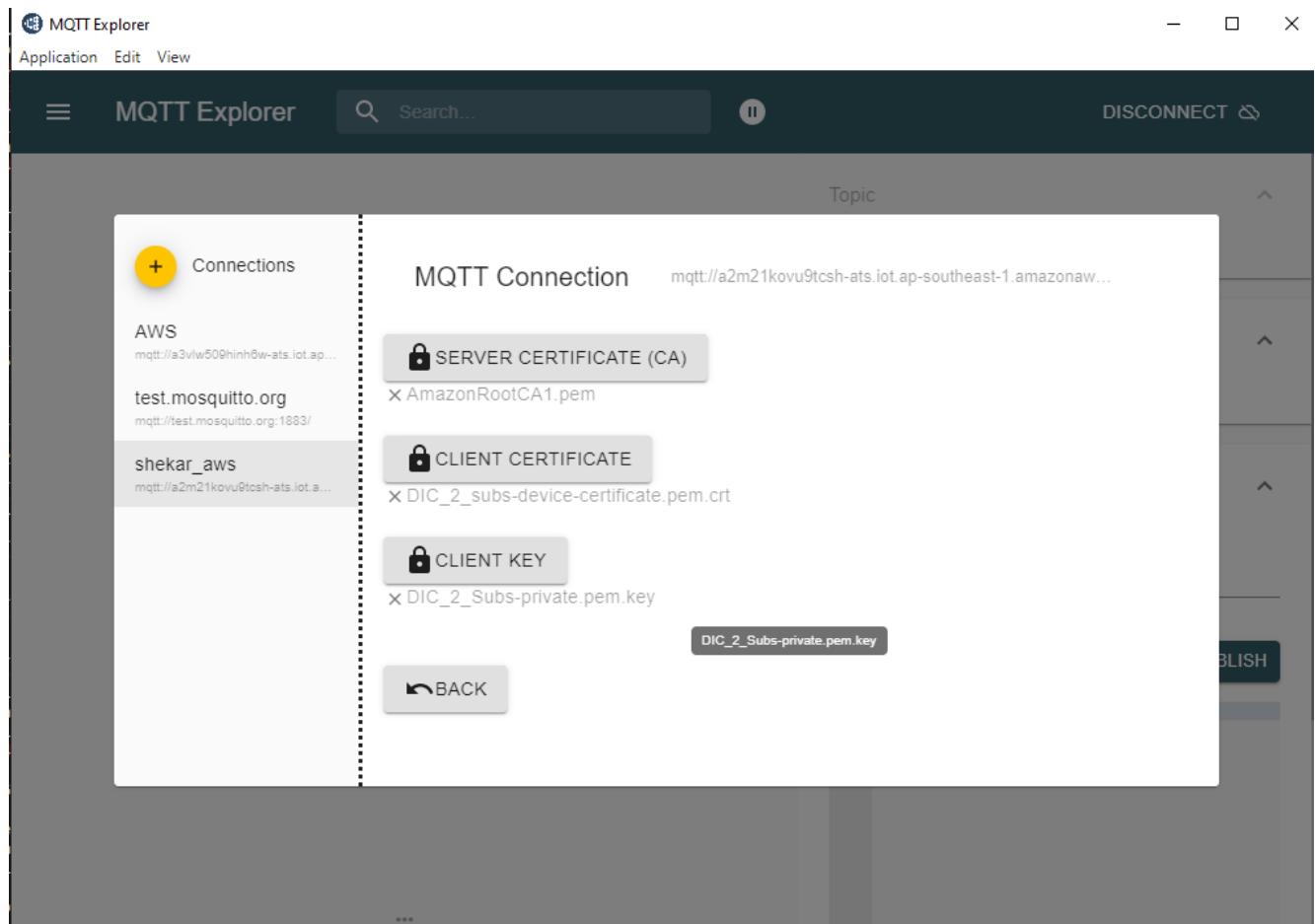


Connecting to Mosquitto Connection

- Host : Your Mosquitto ip address
- Port : 8883
- Make sure you enable Validate Certificate and Encryption
- Click Advanced Settings



- Add application specific topics as shown below
 - For Lighting app, topic to be added (light/*)
 - For onoff plug app, topic to be added (light/*)
 - For Lock app, topic to be added (lock/*)
 - For thermostat app, below topics to be added
 - LocalTemperature/*
 - OccupiedCoolingSetpoint/*
 - OccupiedHeatingSetpoint/*
 - thermostat/*
 - For Windows app, below topics to be added
 - lift/*
 - tilt/*
- MQTT Client ID depends on the certificate set that you will use.
- Add the Certificate, following step 5 in [openssl certificate create](#)



Build DIC Application

Build Procedure For Wi-Fi Direct Internet Connectivity (DIC)

The following components are common for all apps and should be modified in the corresponding app specific .slcp file.

How to Add the DIC Component

To add DIC Component, modify corresponding app specific .slcp file.

```
- id: matter_dic  
from: matter
```

How to Add the DIC AWS OTA Component

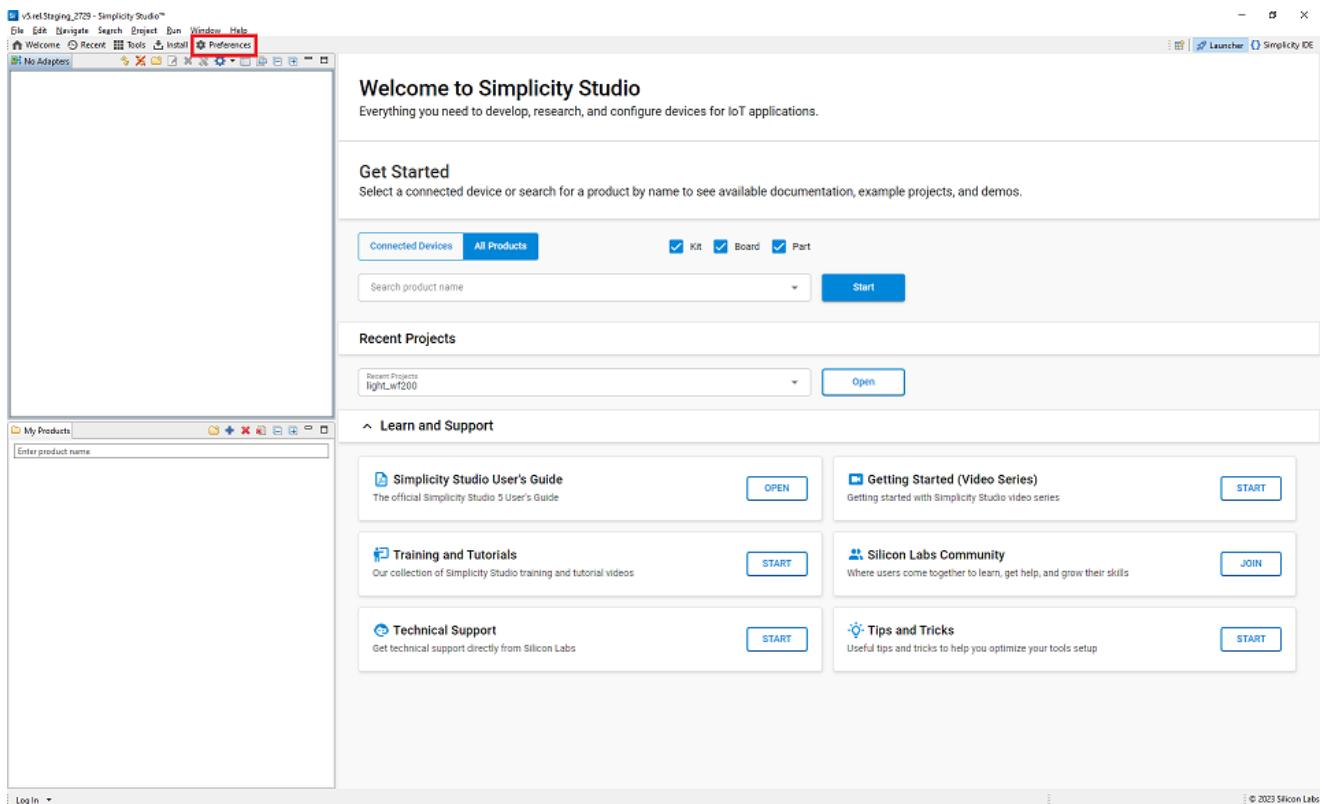
To add DIC AWS OTA Component, modify corresponding app specific .slcp file.

```
- id: aws_ota_wifi_dic  
from: matter
```

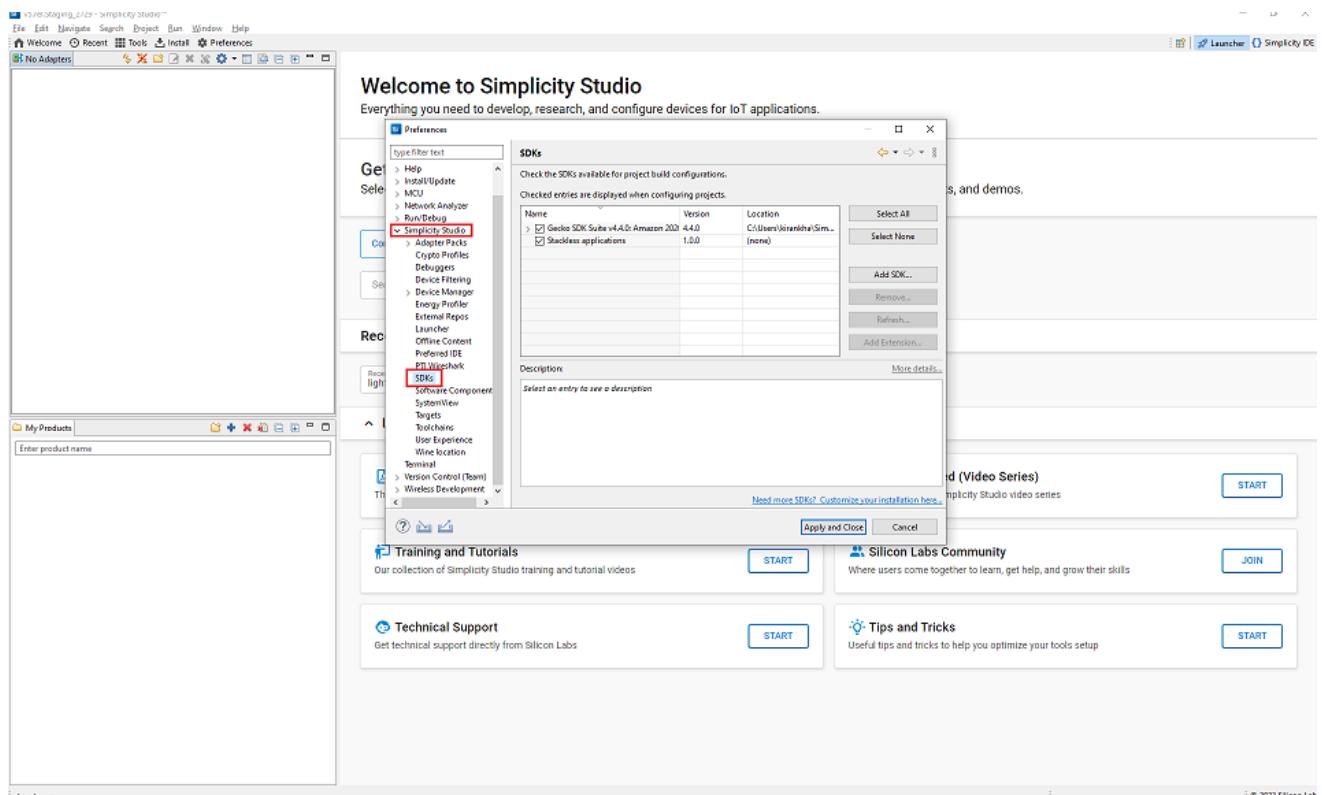
Note:- Building with aws_ota_wifi_dic component enables matter_dic component by default.

Building DIC Application

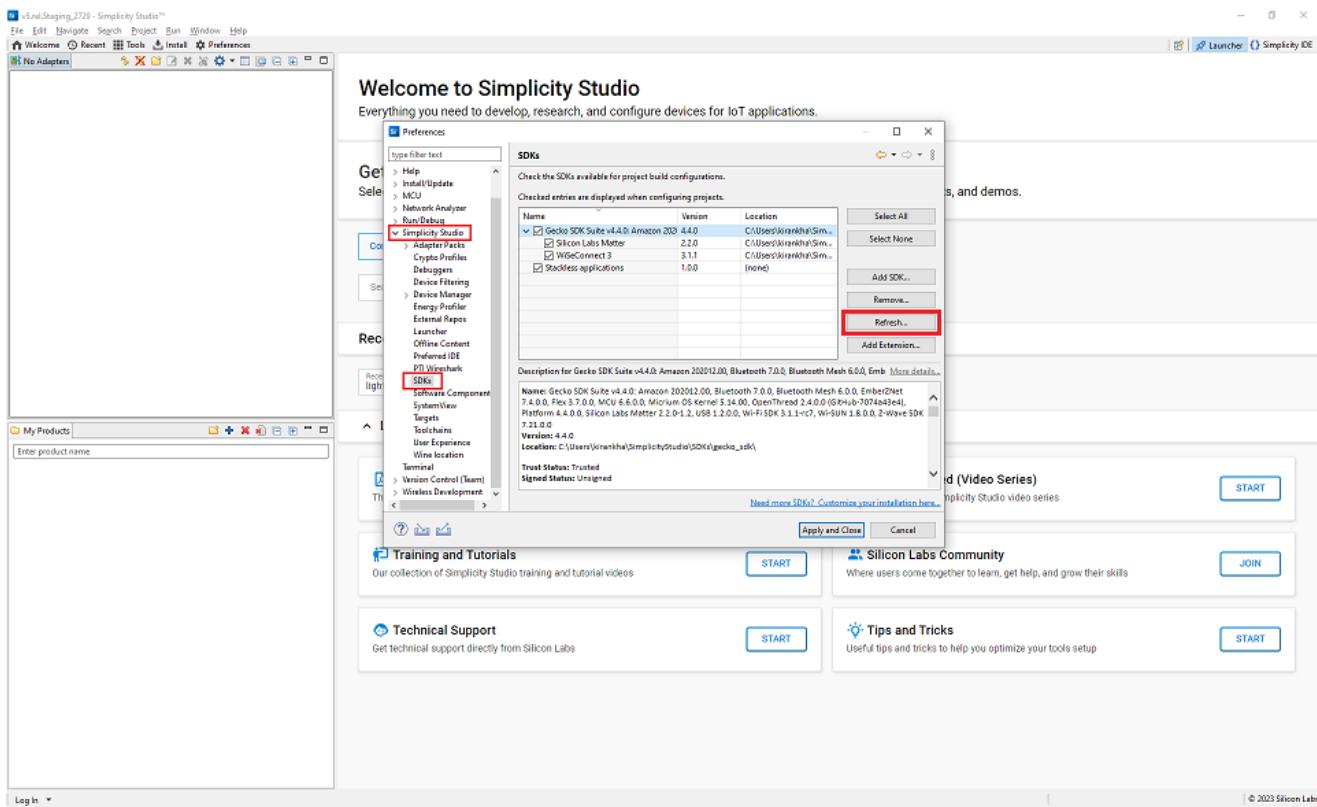
- After Modification in .slcp Project file as above step, refresh the **matter-extension** in Simplicity Studio.
- Select **Preferences** in **Launcher** tab.



- Expand Simplicity Studio section and click on **SDKs** Tab.



- Expand Gecko SDK and click the Refresh button from side menu.



- Build the DIC application using Simplicity Studio
 - [Build EFx32 Application Using Studio](#)
 - [Build SOC Application Using Studio](#)

Compile Using New/Different Certificates

- Two devices should not use the same `DIC_CLIENT_ID`. To use a different Client ID for your second connection, do the following:
- If using AWS, change the following file `matter_extension/examples/platform/silabs/DIC/matter_abs_interface/src/dic_nvm_cert.cpp` under `#if USE_AWS`.
 - Use `DIC_SERVER_HOST` name with your Server host name.
 - For Example: a2m21kovu9tcsh-ats.iot.ap-southeast-1.amazonaws.com
 - Use `device_certificate` and `device_key` with your device cert and device key.
 - Use `DIC_CLIENT_ID` macro value with your Client ID in `matter_extension/examples/platform/silabs/DIC/matter_abs_interface/inc/dic_config.h`
 - The preferred certificate type to use in the application is ECDSA.
 - If using mosquitto, change the following file `matter_extension/examples/platform/silabs/DIC/matter_abs_interface/src/dic_nvm_cert.cpp` enable `USE_MOSQUITTO` and disable `USE_AWS`.
- Under `#if USE_MOSQUITTO`
 - Use `ca_certificate`, `device_certificate` and `device_key` with your `ca_certificate`, device cert and device key.
 - Use `DIC_CLIENT_ID` macro value with your Client ID.
- The preferred certificate type to use in the application is ECDSA.

Matter Ecosystems

Overview of Matter Ecosystem

- The Silicon Labs Matter platform supports the most recent versions of the Matter protocol and works with several commercial Matter ecosystems.
- Matter allows multiple ecosystems to work together and control a single, shared Device.
- Silicon Labs Matter devices can be controlled by various Matter enabled Ecosystems.

Prerequisites

Ecosystems

"Off the shelf" devices which are compatible with the official implementation of Matter in at least one commercial ecosystem:

- [Google Matter Hubs](#)
- [Apple Matter](#)
- [Samsung SmartThings Matter](#)
- [Amazon Alexa Matter](#)

Smartphone to Control the Ecosystem

- Android smart phone installed with respective Ecosystem mobile apps [Amazon Alexa](#), [Google Home](#), [Samsung Smart Things](#).
- Apple iPhone installed with mobile app [Apple Home](#).

Silicon Labs Development Boards

For detailed information about Silicon Labs Development Boards, refer to the [Hardware Requirements Page](#).

Setting up Matter with an Ecosystem and Running a Matter Application

A Matter device can be controlled by single Ecosystem controller and is also interoperable with multiple controllers.

- Follow [Single Controller Setup and Execution](#) to operate with single controller
- Follow [Multiple Controller Setup and Execution](#) to interoperate with multiple controllers

Single Controller Configuration

How to Setup an Ecosystem with Matter

- In order to setup Matter over Wi-Fi in a compatible Ecosystem follow the links below:
 - [Google Ecosystem Setup - Section 3](#)
 - [Apple Ecosystem Setup - Section 3](#)
 - [Amazon Ecosystem Setup - Section 3](#)
 - [Samsung Ecosystem Setup - Section 3](#)

Commissioning a Matter Application within an Ecosystem

- Commissioning means controlling the Device and Application through Matter Compatible Ecosystems.
- In order to perform commissioning with an Ecosystem follow links below:
 - [Google Ecosystem Demo Execution - Section 5](#)
 - [Apple Ecosystem Demo Execution - Section 4](#)
 - [Amazon Ecosystem Demo Execution - Section 4](#)
 - [Samsung Ecosystem Demo Execution - Section 4](#)

Google Ecosystem Setup

Google Ecosystem Setup and Demo Execution

Hardware Requirements

For the hardware required for the Google Nest Hub Ecosystem, refer to the [Ecosystem overview Prerequisites section](#).

Software Requirements

- Google Account
- Google Home App with Beta Version

Set Up Google Home and Android Smartphone

Google Matter Early Access Program (EAP)

The Google Matter Early Access Program is a partnership between Google and silicon providers who support Matter development. This partnership allows for faster onboarding of new devices into Matter and Thread by lowering the bar for starting with development of a new Matter-based product.

The Google Matter Early Access Program is run through the [Google Developer Center](#).

The Matter-focused portion of the Google Developer Center is located [here](#).

The Google Matter Early Access Program is located [here](#).

Note: Until the public preview, access to this page is reserved to those allowed in by the Google Partner engineering team.

Prerequisites for Google Setup

To run the Google Ecosystem demo, you will need both Google and a Matter device. You will also need Google Nest Hub 2nd Generation and an Android phone (at least a Pixel 5 is recommended) that can run at least Android 8 (8.1, API Level 27) or newer and has Bluetooth LE capability.

Instructions for Setting Up EAP

Once you have access, you will need to set up the Nest Hub 2nd Gen and Android phone with the Google Home app using the same Google Account that is used to access the EAP website.

If you have set up the Nest Hub 2nd Gen with the correct Google account, you will receive the OTA update to the Beta version within 24 hours. You can verify this by going to Device information > Technical information > Update Channel and the channel should read "matter-dev-current-beta-channel".

Set Up the Android Phone

Follow these instructions to set up the Android phone with the necessary applications:

- [Set up the Google Home app](#)
- [Set up Google Play Services](#)

Create a Matter Integration in the Google Developer Console

Follow [these instructions](#) to create a Matter integration in the Google Developer Console.

After completing these steps, you should be ready to build your Matter accessory device.

Matter Integration Setup in the Developer Console

- Once you have created a home in your Android smartphone, add your Nest Hub to that home.
- After this, on a browser on your PC go to this to create a project: <https://developers.home.google.com/matter>
- Click **Console** at the top of the page.
- On the next page, click **Create a Project**.
- Give your project a name and click **Create a new project**.
- On the next page click **Add Matter integration**.
- On the next page click **Next: Develop**.
- Click **Next: Setup**.
- Set up the fields on this page as shown:
 - Product name: Light
 - Device type: light
 - Vendor ID (VID): Test VID
 - Test VID: 0xFFFF1
 - Product ID (PID): 0x8005
- Product ID options for Matter devices are as follows:
 - Light-Switch: 0x8004
 - Light: 0x8005
 - Lock: 0x8006
 - Thermostat: 0x800E
 - Window Covering: 0x8010
- Click **Save & Continue**.
- On the next page click **Save**.
- You will now see a Matter integration for device type light in your console.

You have now completed setting up the following:

- Your home in the Google home app in your Android smartphone
- A project in your Google developer console
- A matter integration for the light device type

Having finished the above, the only step left to have your setup ready is to open a QR code webpage for the light device type in your PC. A QR Code link will be present in Device configuration section of logs. Copy the link and paste it in google chrome so you will be able to QR Code.

```

00> [00:00:01.122][info ]|[DL] Device Configuration:
00> [00:00:01.123][info ]|[DL]   Serial Number: (not set)
00> [00:00:01.123][info ]|[DL]   Vendor Id: 65521 (0xFFFF1)
00> [00:00:01.123][info ]|[DL]   Product Id: 32773 (0x8005)
00> [00:00:01.123][info ]|[DL]   Product Name: SL_Sample
00> [00:00:01.123][info ]|[DL]   Hardware Version: 0
00> [00:00:01.123][info ]|[DL]   Setup Pin Code (0 for UNKNOWN/ERROR): 0
00> [00:00:01.124][info ]|[DL]   Setup Discriminator (0xFFFF for UNKNOWN/ERROR): 3840 (0xF00)
00> [00:00:01.124][info ]|[DL]   Manufacturing Date: (not set)
00> [00:00:01.124][info ]|[DL]   Device Type: 65535 (0xFFFF)
00> [00:00:01.124][info ]|[SVR] SetupQRCode: [MT:6FCJ142C00KA0648G00]
00> [00:00:01.124][info ]|[SVR] Copy/paste the below URL in a browser to see the QR Code:
00> [00:00:01.124][info ]|[SVR] https://project-chip.github.io/connectedhomeip/qrcode.html?data=MT%3A6FCJ142C00KA0648G00
00> [00:00:01.125][info ]|[ZCL] On/Off epi value: 0

```

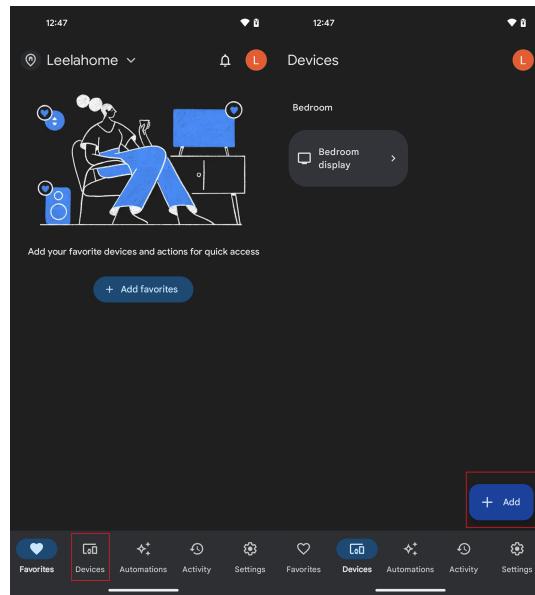
Matter Demo Execution using Google Home

Commission Matter Device through Google Home App

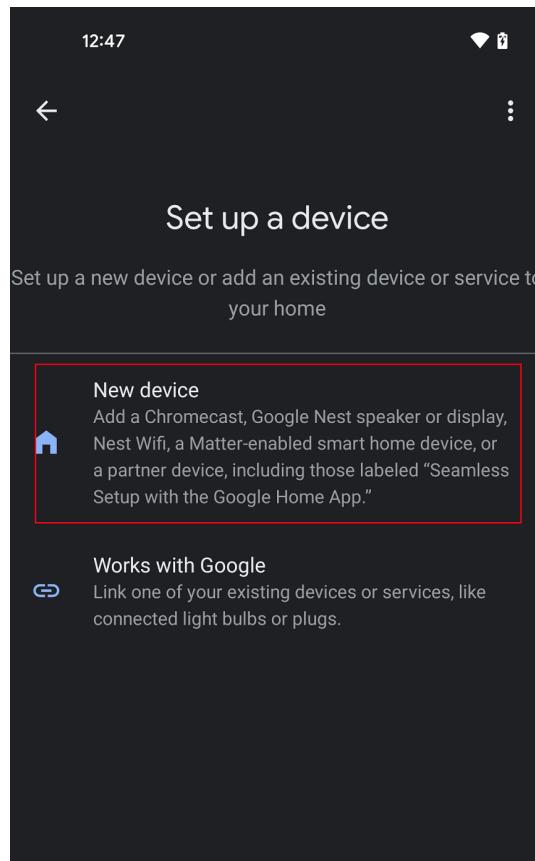
- Refer [Getting Started Overview Guide](#) for setting up a Silicon Labs Matter Accessory Device.
- Connect Board to a Computer
 - For Wi-Fi NCP Mode Boards see [Connect EFR32 Board to computer](#)
 - For Wi-Fi SOC Mode Boards see [Connect SiWx917 SOC to Computer](#)
- Flash the bootloader binary for your device along with the application (for example, lighting, lock, thermostat, window covering, or light-switch) using [Simplicity Commander](#).

4. Open the Google Home app on your phone.

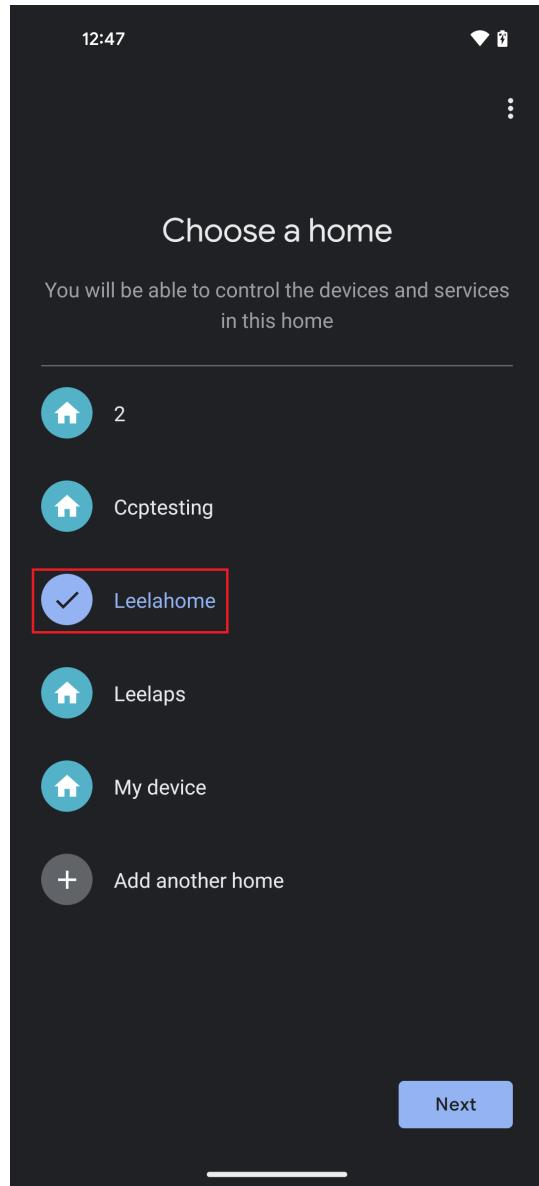
5. Click **Devices Section** and click **Add**.



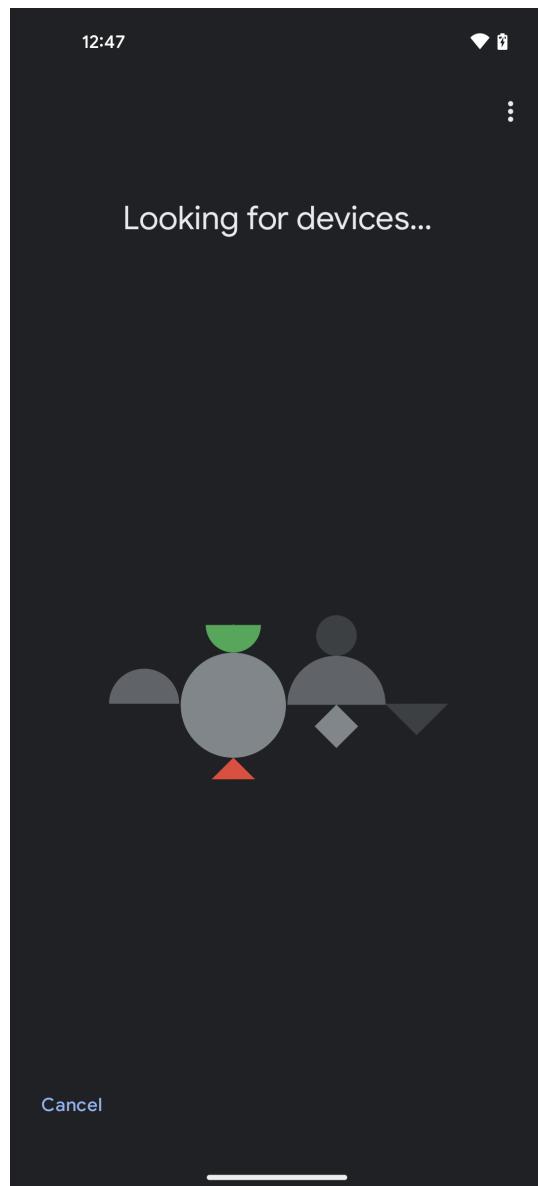
6. In the **Set Up Device** window, tap to select the **New Device** option.



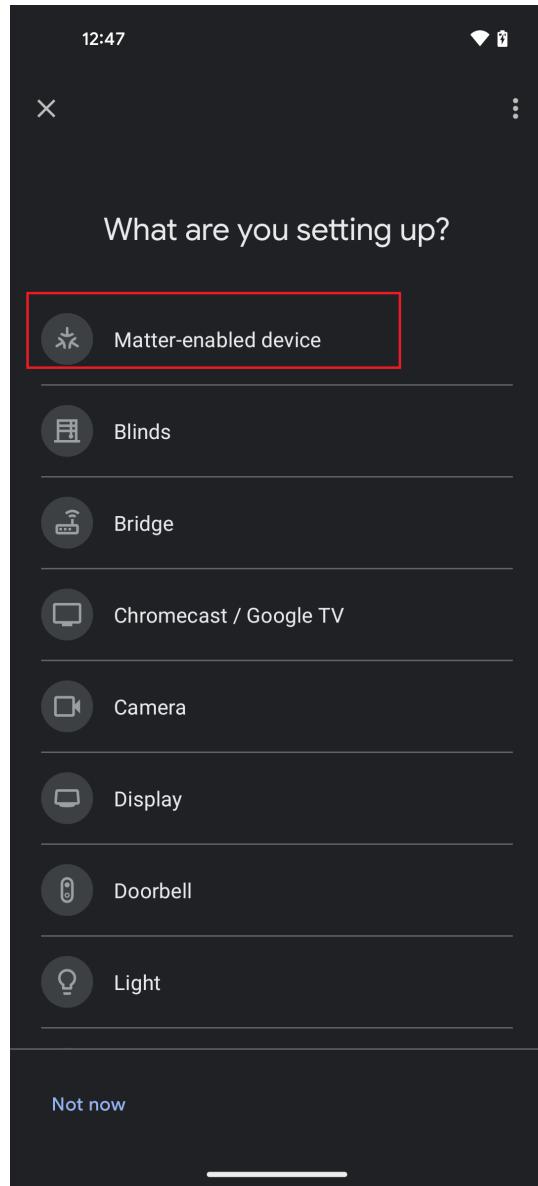
7. In the **Choose Home Section** select your home and click next.



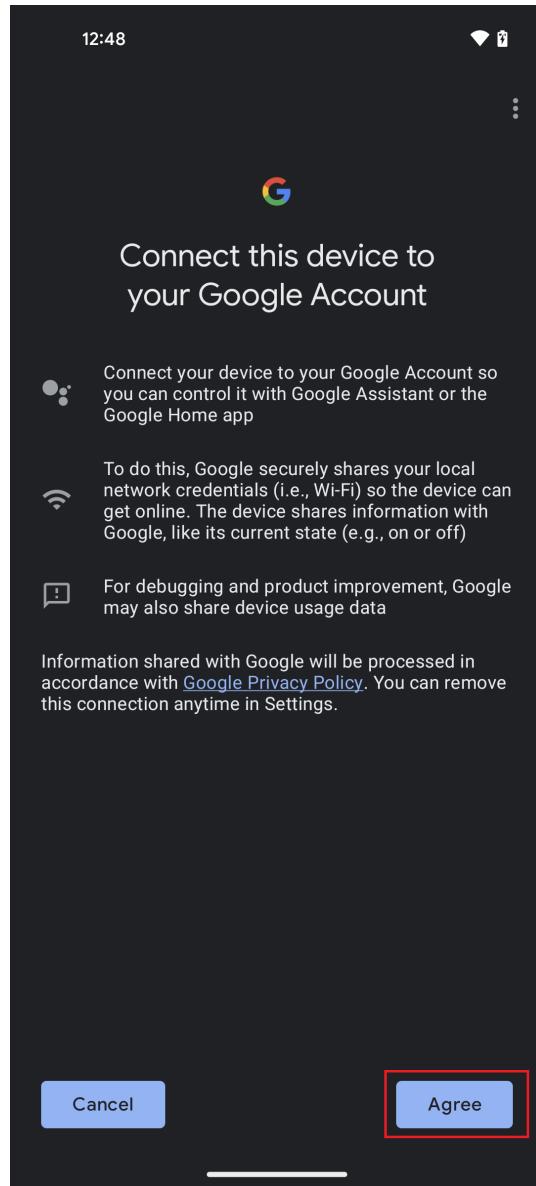
8. The Google Home App searches for a nearby Matter device.



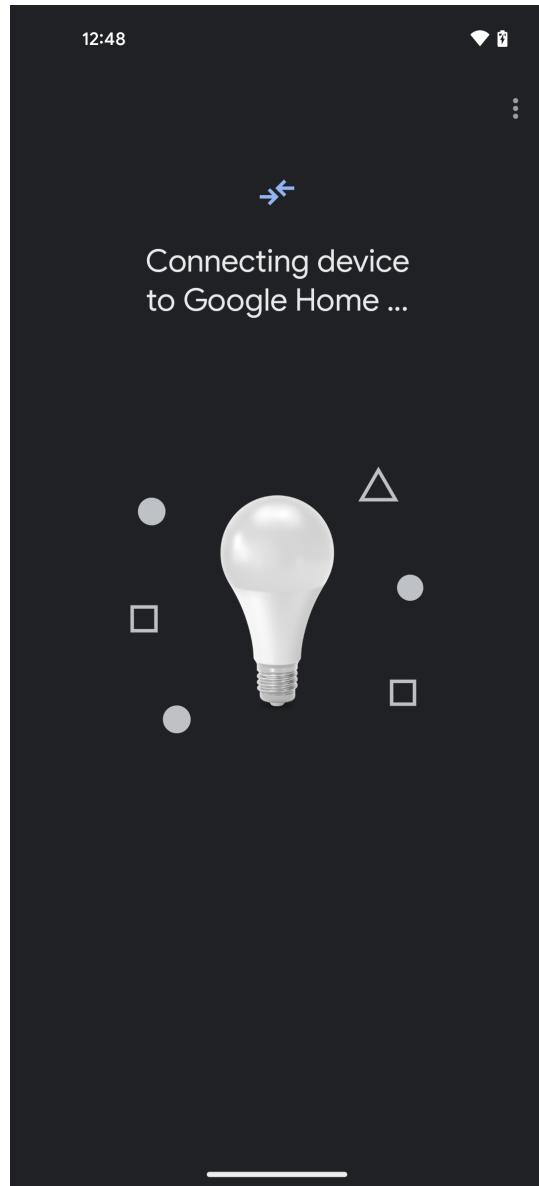
9. If the device is found, click the Application which you flashed on the device, such as Light or Lock.
10. If the device is not found, tap **Matter Enabled Device**.



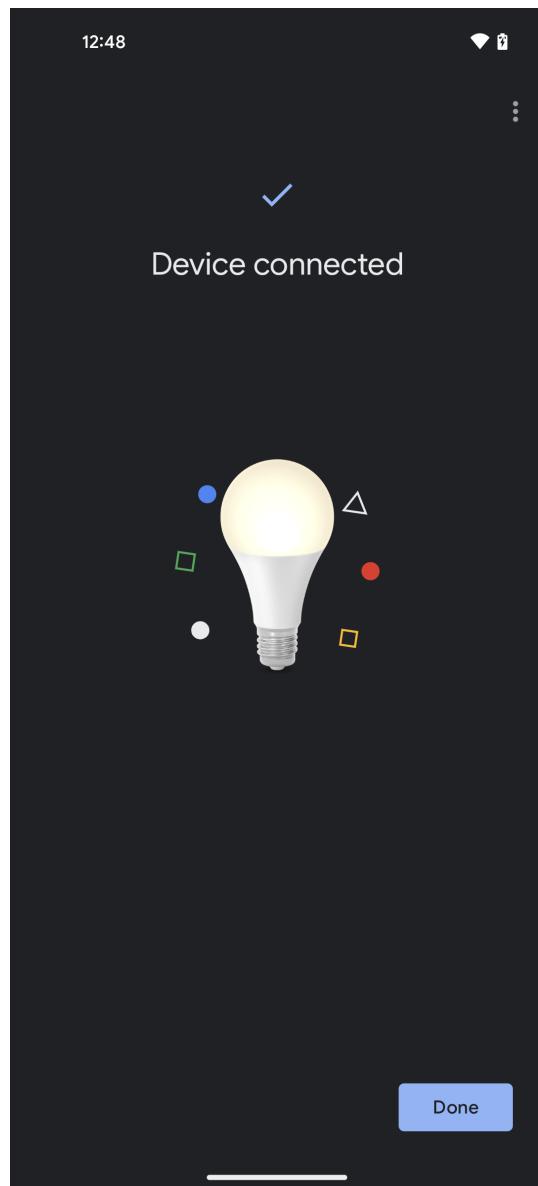
11. Once the Google Home app has found the device, it will ask you to scan its QR code.
12. Google Home app asks if you want to connect this device to your Google account. Tap **I agree**.



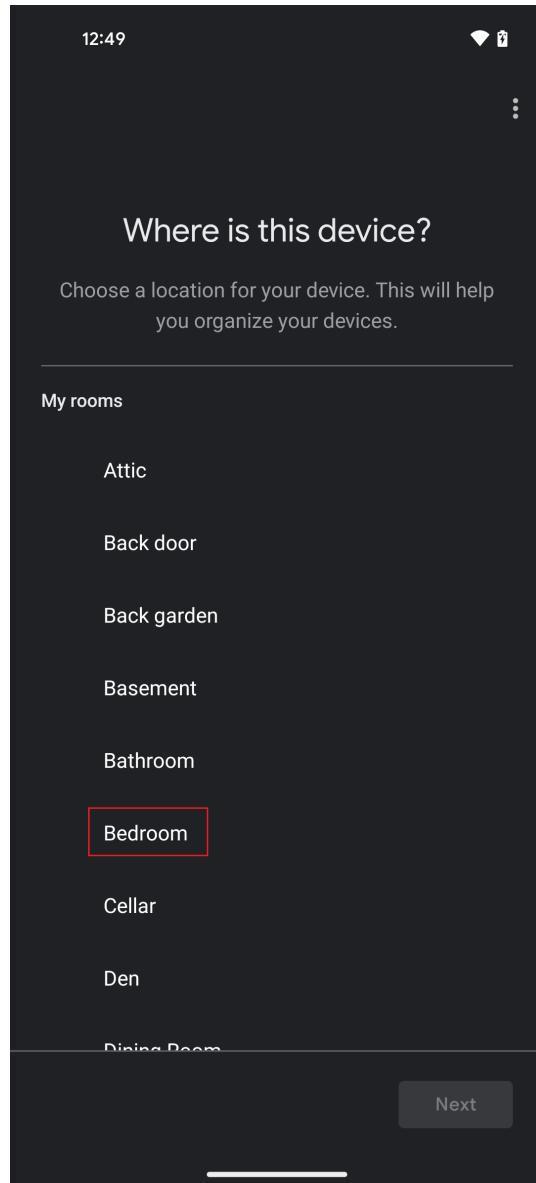
13. The Google Home app now starts commission the device with Bluetooth LE.



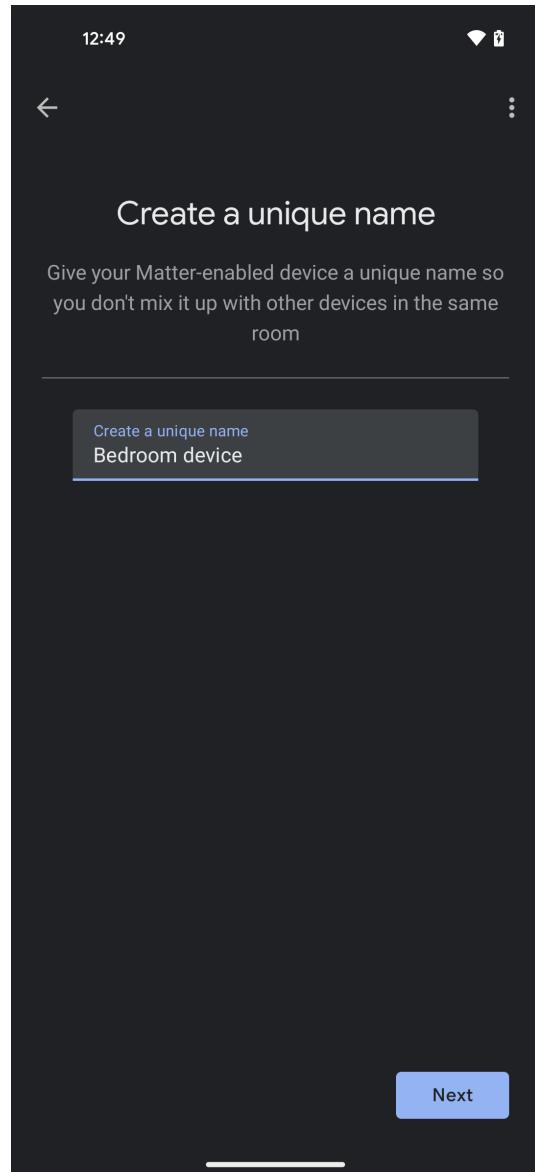
14. Once you see the 'Device connected' message, tap **Done**.



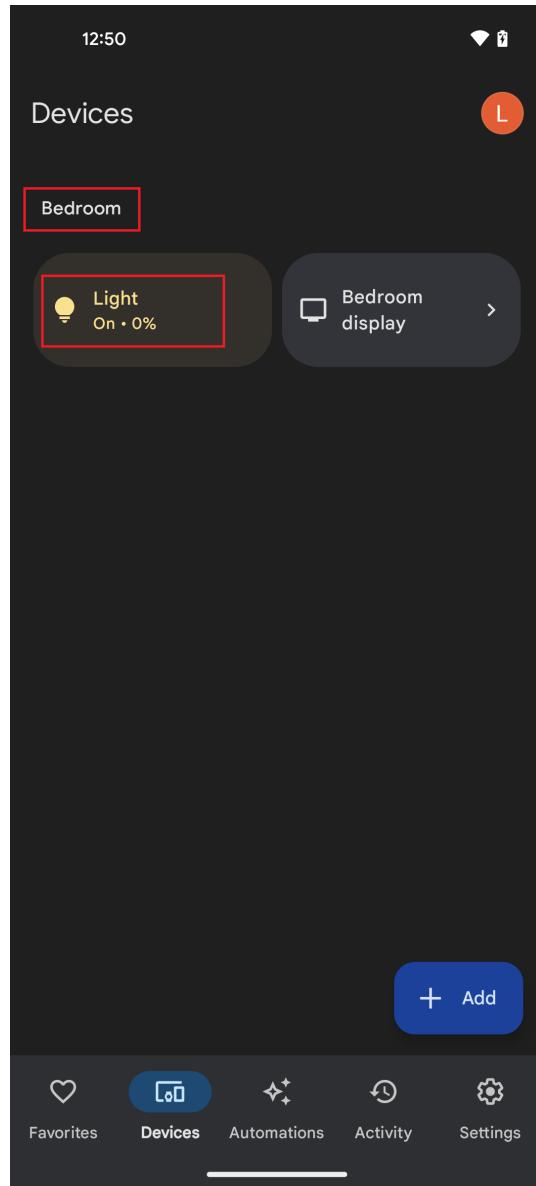
15. The Google Home App now asks you to select a Room where you want to keep the Application. You can select any room from the list and click **Next**.



16. At the prompt to create a unique name for the application (For Example: Light, Lock), create any name to identify the application and click **Next**.



1. You will now see your device (for example, Light) shown as being connected to your Google account and added to your selected Room at Step 15.



Control the Light via Google Home App

1. In the Google Home app, you will now be able to tap your light to turn it ON and OFF.
2. You can control the light by giving a voice command (for example, 'Ok Google! Turn ON Light') and through the app user interface.
3. You will see the LED1 on your WSTK board turned on or off depending on the command you enter.

Deleting Matter Application from Google Home

1. Press and hold Matter Application for detailed view.
2. Click **Setting** on the top right corner.
3. Select the **Remove device** option.
4. At the 'Unlink all Matter apps & services from device', select **Unlink**.

Apple Ecosystem Setup

Apple Ecosystem Setup and Demo Execution

This page describes how to set up the Apple Ecosystem for Matter and test a Matter application using Apple Home Pod Mini.

Hardware Requirements

For the Hardware required for an Apple EcoSystem, refer to the [Ecosystem overview Prerequisites section](#).

Software Requirements

- Apple Account
- Apple Home App on Smartphone

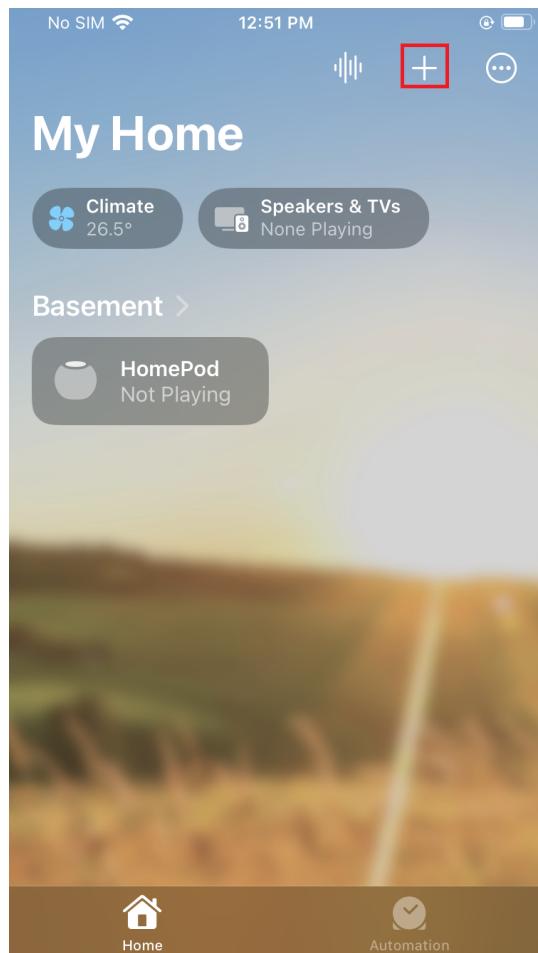
Note: Apple only has Matter support with IOS version-16.1 or higher.

Set Up Apple HomePod and Apple Phone

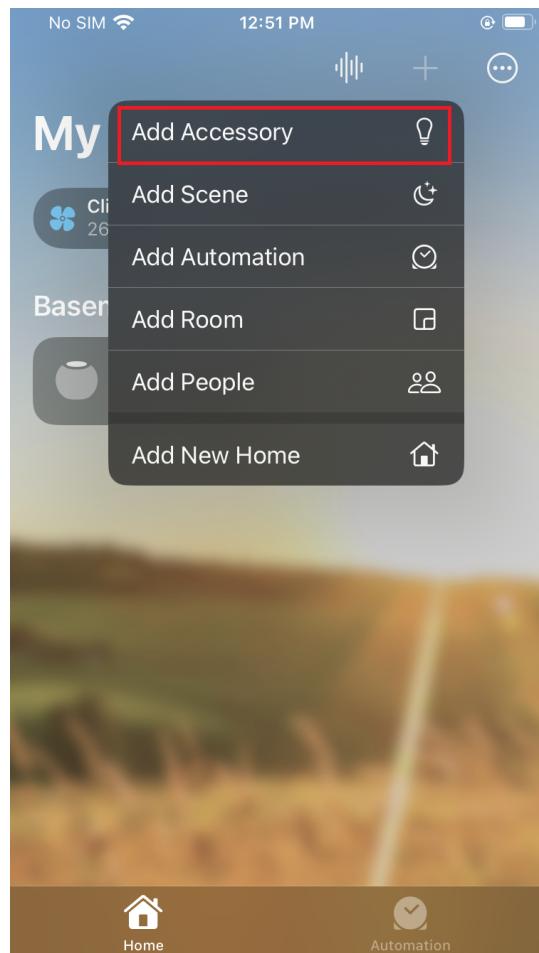
Refer to Apple's [Set up HomePod or HomePod mini](#)

Matter Demo Execution using Apple HomePod

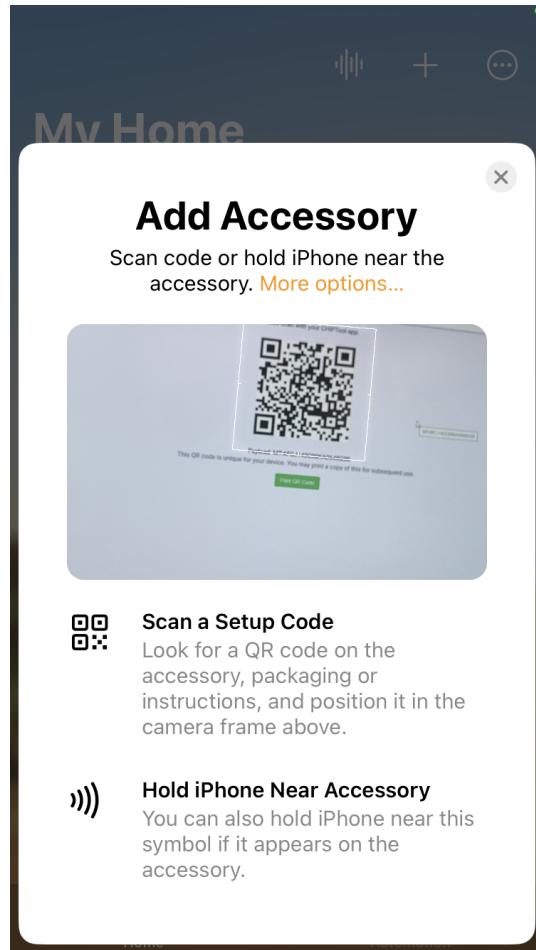
1. Refer to [Getting Started with Matter over Wi-Fi](#) for instructions on setting up Silicon Labs Matter Accessory Device.
2. Connect a board to a computer.
 - For Wi-Fi NCP Mode Boards, see [Connect EFR32 Board to Computer](#).
 - For Wi-Fi SoC Mode Boards, see [Connect SiWx917 SoC to Computer](#).
3. Flash the bootloader binary for your device along with the application (for example, lighting, lock, thermostat, window covering, or light-switch) using [Simplicity Commander](#).
4. In the Apple Home App, click the "+" button.



5. Select Add Accessory.



6. It will prompt you to scan the QR Code using a smartphone camera.



7. Connect the device to a computer and scan the QR code within the Home app.
8. Proceed to add the device to your home. You should see LED0 fast blinking when commissioning happens.
9. Once commissioning is complete, the Apple Home app prompts you to **select One Room** for your Matter application. Select any room as per your choice and enter the Application name. (For Example: Light, Lock)

Control the Light via Apple Home App

- In the Apple Home app, you can now tap your light to turn it ON and OFF.
- You can control the light by giving a voice command (for example, 'Hey Siri! Turn ON Light') and through the app user interface.
- You will see the LED1 on your WSTK board turned on or off depending on the command you enter.

Delete the Matter Application From Apple Home

1. Click the Matter Application for the detailed view.
2. Scroll down to the end.
3. Select **Remove Accessory**.
4. Confirm to remove from **My Home** and your Matter application is removed from Apple Home.

Note: Removing the Matter application from Apple Home App removes it from Apple Home Pod as well.

Amazon Ecosystem Setup

Amazon Ecosystem Setup and Demo Execution

Hardware Requirements

For the hardware required for an Amazon EcoSystem, refer to the [Ecosystem Prerequisites section](#).

Software Requirements

- Amazon account
- Amazon Alexa App on a smartphone

Amazon Alexa and Android Smartphone Setup

Amazon Alexa MSS (Matter Simple Setup)

As part of partnership with Amazon, the following link contains information required for Matter device certification with Amazon.

<https://developer.amazon.com/docs/frustration-free-setup/matter-simple-setup-for-wifi-overview.html>

In the context of MSS for Wi-Fi, the provisionee, or commissionee, is the device that is to be automatically set up. If you want to make your device eligible to be an MSS commissionee, you must satisfy the following:

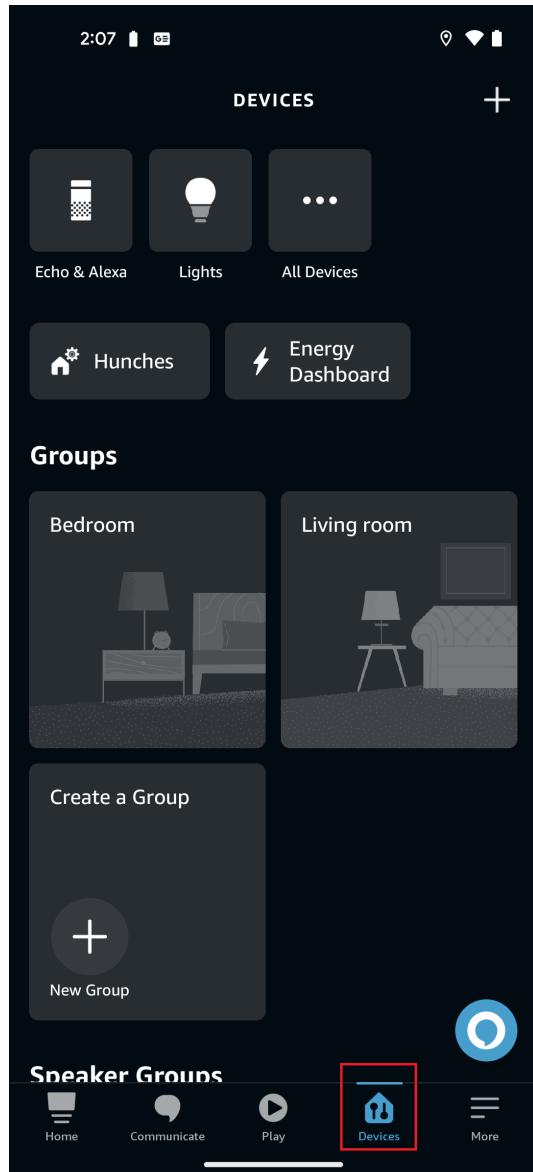
1. Configure the device to beacon over Bluetooth LE (BLE) with specific fields needed for MSS for Wi-Fi (detailed below).
2. Onboard your device via the FFS developer portal by creating a Matter new device type. On the developer portal, you will manage your FFS onboarding lifecycle tasks, like managing your test devices and manufacturing data and submitting for certification.
3. Integrate a unique barcode on your device packaging. You can also use an existing unique barcode on your packaging, such as a serial number, or MAC address.
4. Share your device control log data with Amazon services. Control Logs are a mechanism that allows manufacturers to provide Amazon with unique device identifiers and authentication material, such as the Matter passcode, that are critical to ensure a frictionless customer setup. The unique package barcode is associated with your device identifier through the control logs. See the Matter Control Logs section for more details.
5. Complete Frustration-Free Setup certification and Amazon ASIN onboarding. Review the certification section below for more information.

Amazon Alexa Setup

Refer to [Set up Alexa in a Few Easy Steps](#).

Matter Demo Execution using Amazon Alexa

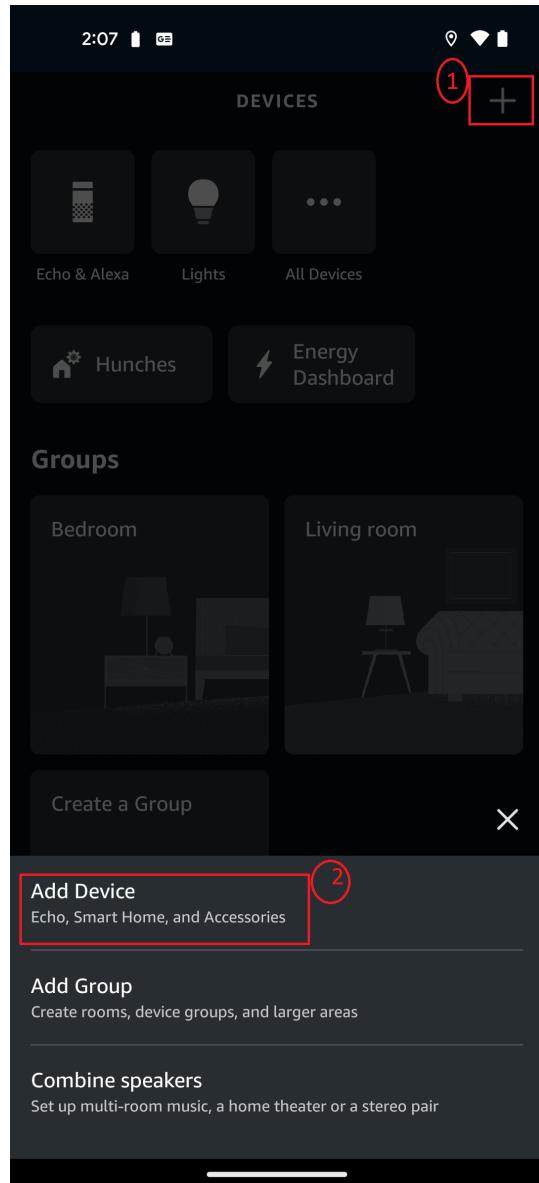
1. Refer to the [Getting Started Overview Guide](#) for setting up a Silicon Labs Matter Accessory Device.
2. Connect a board to a computer.
 - For Wi-Fi NCP Mode Boards, see [Connect EFR32 Board to Computer](#).
 - For Wi-Fi SoC Mode Boards, see [Connect SiWx917 SoC to Computer](#).
3. Flash the bootloader binary for your device along with the application (for example, lighting, lock, thermostat, window covering, light-switch) using [Simplicity Commander](#).
4. Once the Amazon Alexa setup is done, make sure echo dot is ready.
5. Make sure the Matter Application is flashed into the Matter Device (for example, EFR32MG24, SiWx917 SoC, SiWx917 NCP).
6. In the Alexa App, tap **Devices** section.



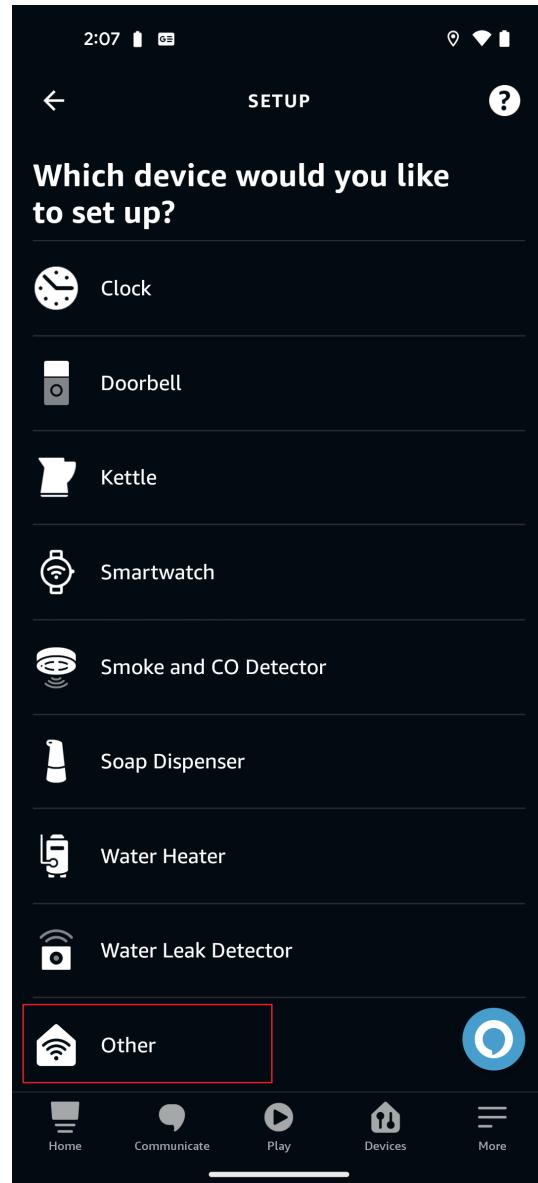
1. Tap "+" at the top right corner. Three options are displayed:

- Add device
- Add group
- Combine speakers

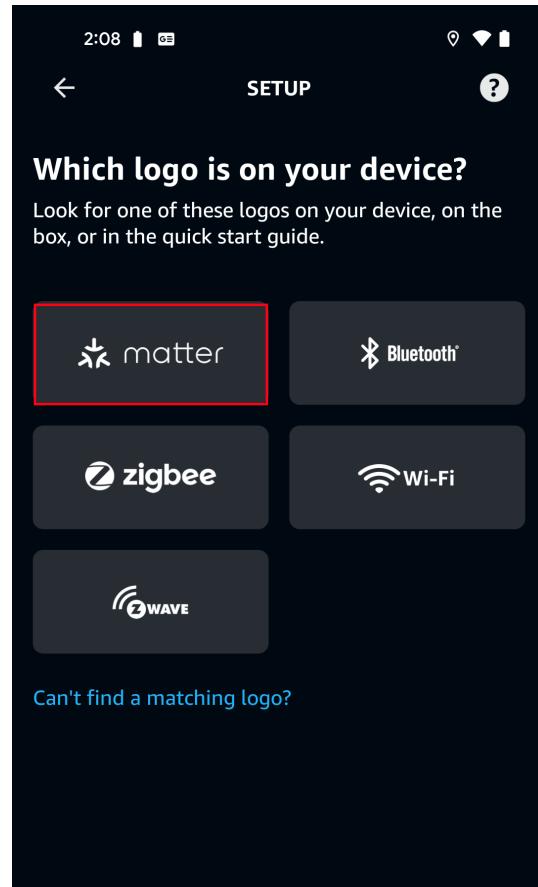
2. Tap **Add device**. Several options are displayed.



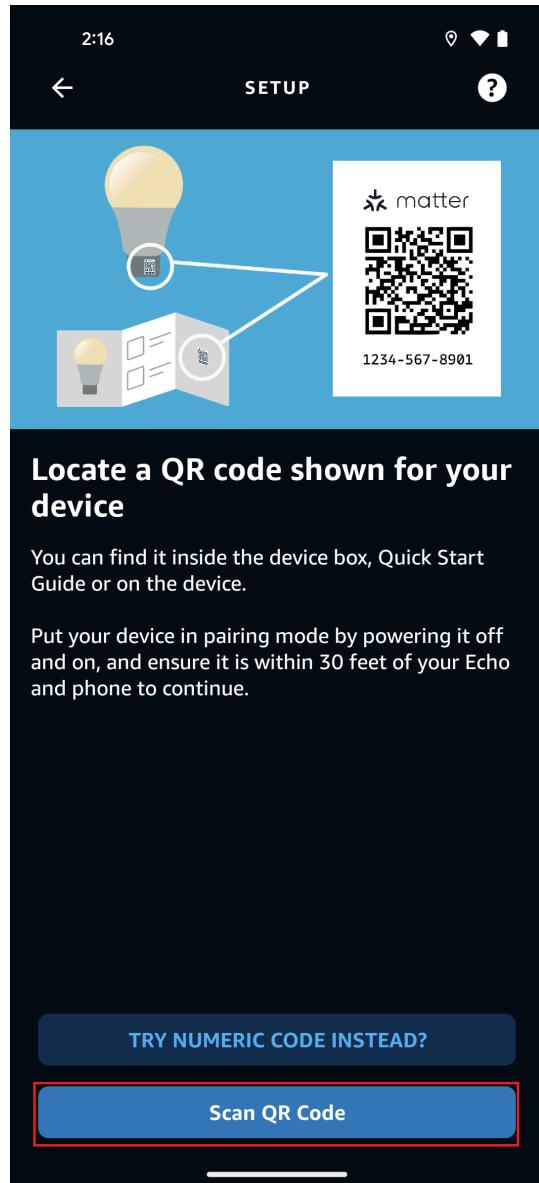
3. Scroll down and select **other** option.



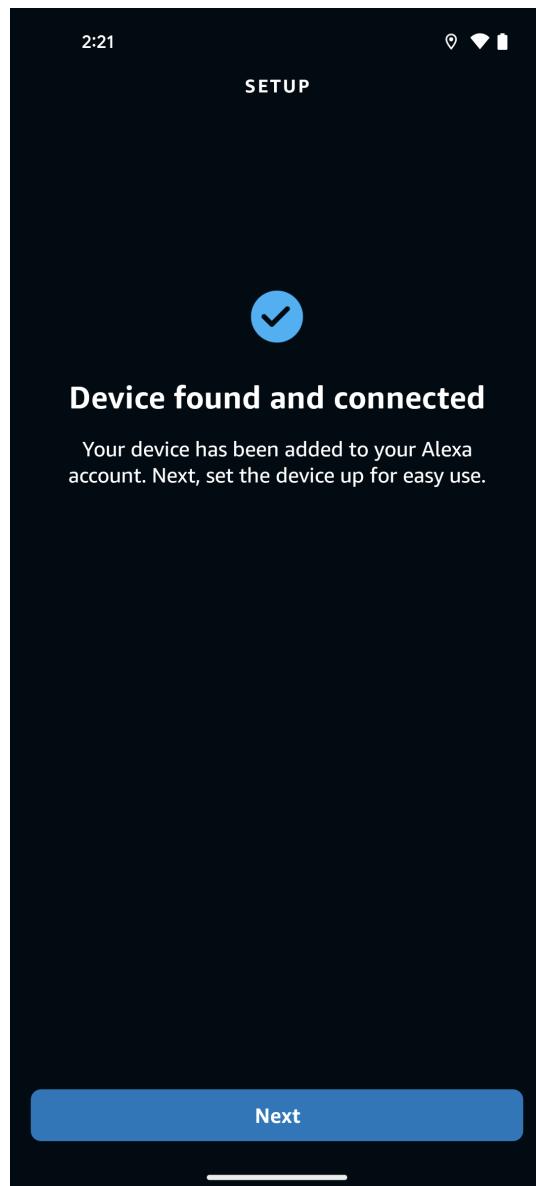
4. Logos such as "Matter", "Bluetooth", "Zigbee", "Wi-fi", "Z-wave" are displayed. Tap the "Matter" logo.



5. Alexa App will ask "Does your device have a matter logo?" Select "Yes".
6. Alexa will prompt "Locate a QR code shown for your device." Select Scan QR Code.



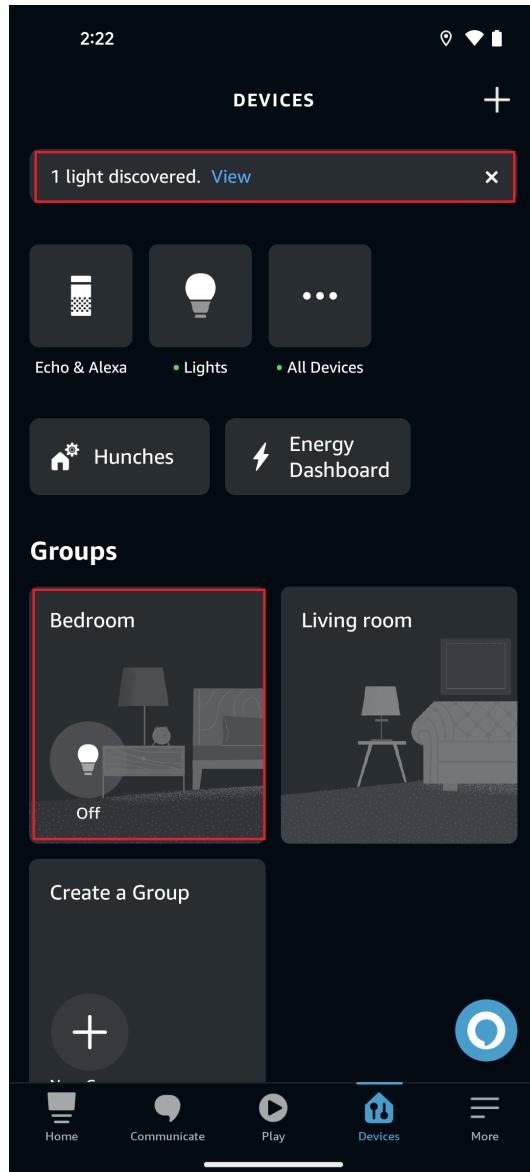
7. After scanning the QR code through a smartphone camera, verify Commissioning is started by checking the Device logs.
8. Once commissioning is triggered the Alexa app will prompt for Access Point Credentials. Add them.
9. After Access Point Credentials are provided the device will join to the network and commissioning is completed.



10. Next, select a group for your device (for example, Bedroom), and click **Add to Group**.



11. Now the application is ready to use. You can see the Matter application in Amazon Alexa app inside the **Groups Panel** at the **Bedroom** tab.



In the Amazon Alexa app, you will now be able to tap your light to turn it ON and OFF. You can also control the light by giving a voice command (for example, 'Hey Alexa! Turn ON Light') and through the app user interface.

The LED1 on your WSTK board will turn on or off depending on the command you enter.

Deleting Matter Application from Amazon Alexa

1. Click Matter Application for detailed view.
2. Click **Setting** button on top right corner.
3. Select **Dustbin Button** on top right corner, it will prompt **Remove "First light"**. Click **DELETE**.

Samsung Ecosystem Setup

Samsung Ecosystem Setup and Demo Execution

Hardware Requirements

For the hardware required for a Samsung Smart Thing EcoSystem, refer to the [Ecosystem Prerequisites](#) section.

Software Requirements

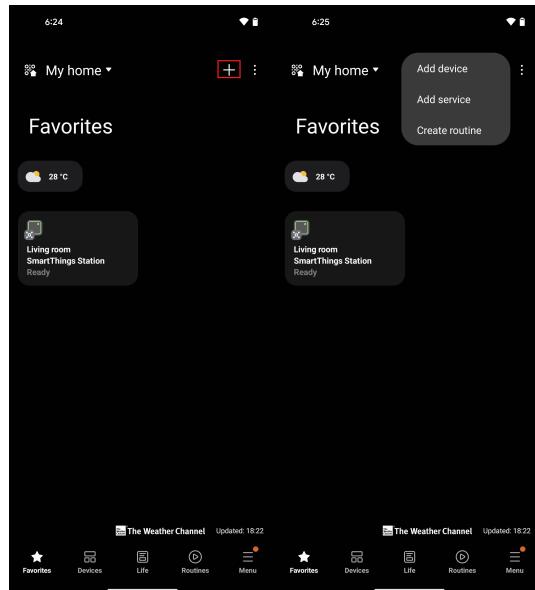
- Samsung Account
- Samsung Smart things App on smartphone, as described in the next section

Setup of Samsung Smart Home Hub

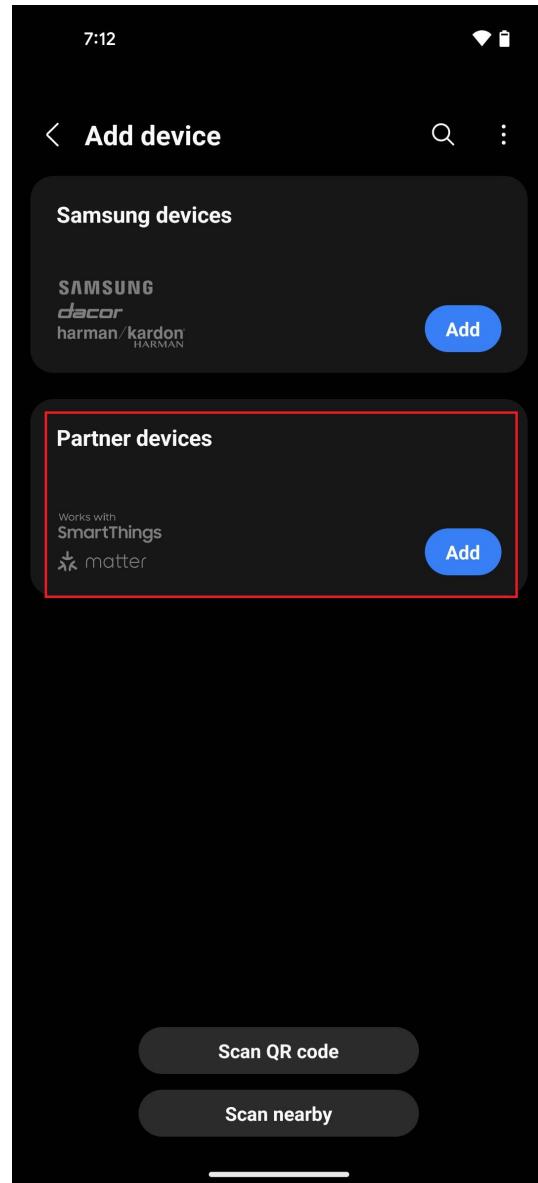
See the Aeotec instructions on [How to Set Up a Smart Home Hub](#).

Matter Demo Execution using Samsung Smart Aeotec

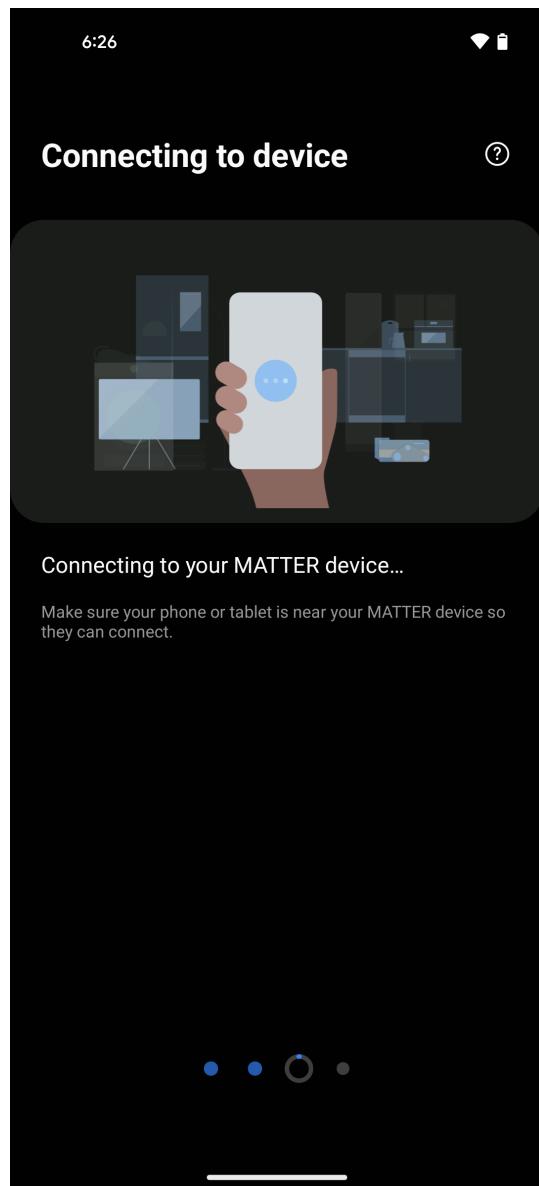
1. Refer to the [Getting Started Overview Guide](#) for setting up a Silicon Labs Matter Accessory Device.
2. Connect a board to a computer.
 - For Wi-Fi NCP Mode Boards, see [Connect EFR32 Board to Computer](#).
 - For Wi-Fi SoC Mode Boards, see [Connect SiWx917 SoC to Computer](#).
3. Flash the bootloader binary for your device along with the application (for example, lighting, lock, thermostat, window covering, or light-switch) using [Simplicity Commander](#).
4. Open the Smart Things app, tap '+', and select **Add device**.



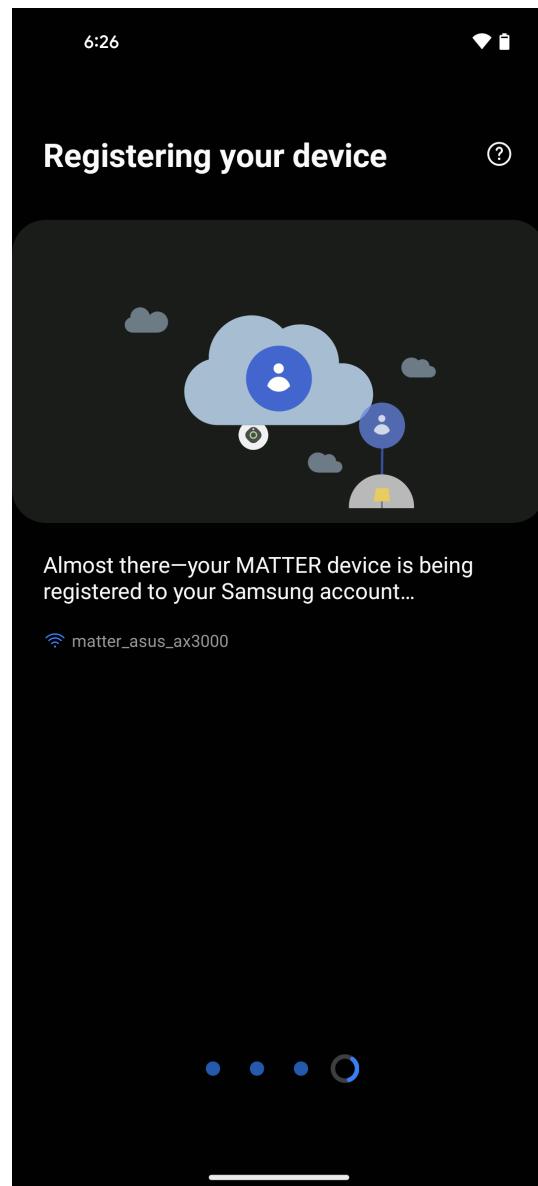
5. Select Partner Devices.



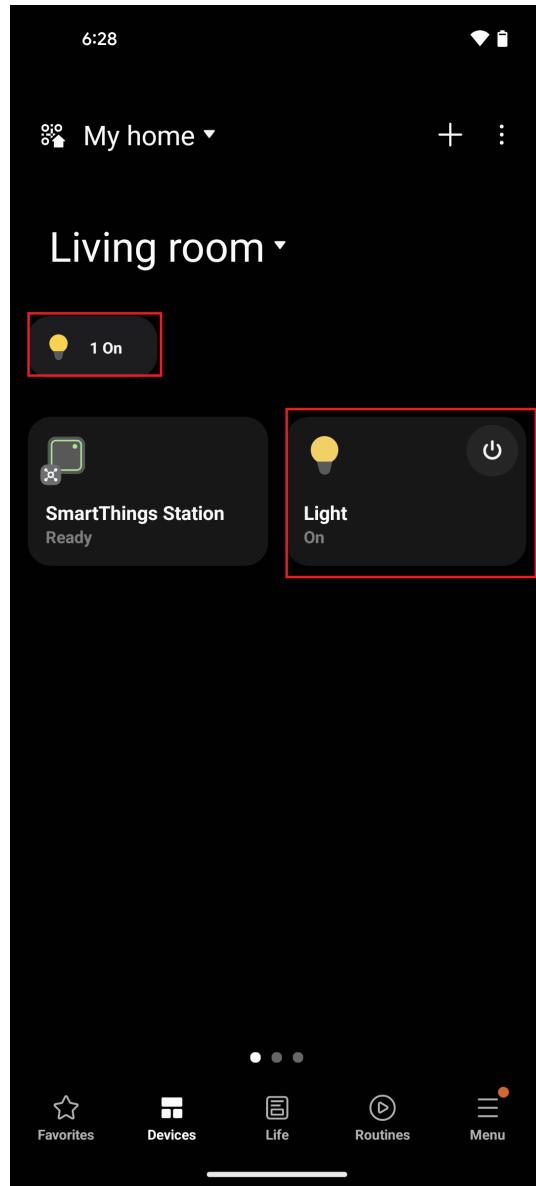
6. Through the Smart Things app scan the Application QR code to trigger commissioning.
7. After scanning the QR code, verify commissioning is triggered by checking on the DUT logs.



8. The last step of the commissioning process is to register your device with Access Point.



9. Once commissioning has succeeded, verify the Matter application is added in one room of the Smart Things app and is in online mode.



10. In the Samsung Smart Thing app, you can now tap your light to turn it ON and OFF. You will see the LED1 on your WSTK board turned on or off depending on the command you enter. **Note:** Samsung Smart Things does not support voice control commands.

Deleting the Matter Application From Samsung Smart Things App

- In order to remove the Matter application from Samsung Smart Things app go to Devices Section.
- Long Press and Hold Matter application and click Remove.
- If you want to add the Matter application again follow the procedure above.

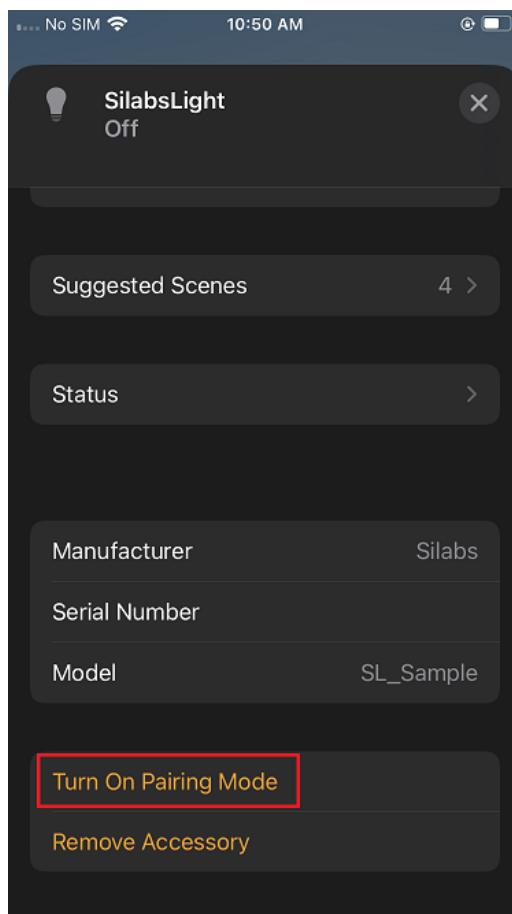
Multi-Controller Configuration

Sharing Multi-Control over Wi-Fi Light from Apple Home Pod to Google Home Ecosystem

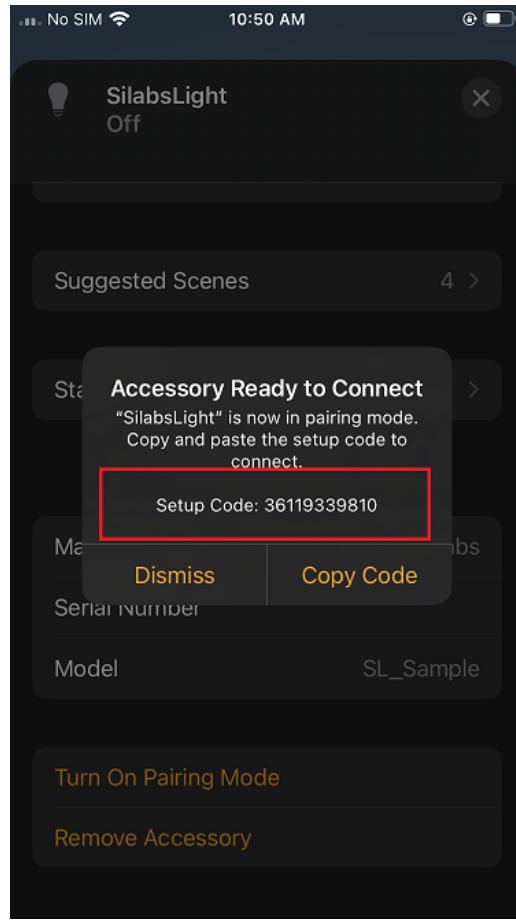
This phase is very similar to the [Single Controller](#). The difference is that the Matter Over Wi-Fi device will be controlled using the Apple Home Ecosystem. The process will be almost identical, with some minor difference related to Apple Home App UI.

See [How to Set Up an Apple Home Ecosystem](#). Then use the following procedure to share multi-control from Apple Home to Google Home.

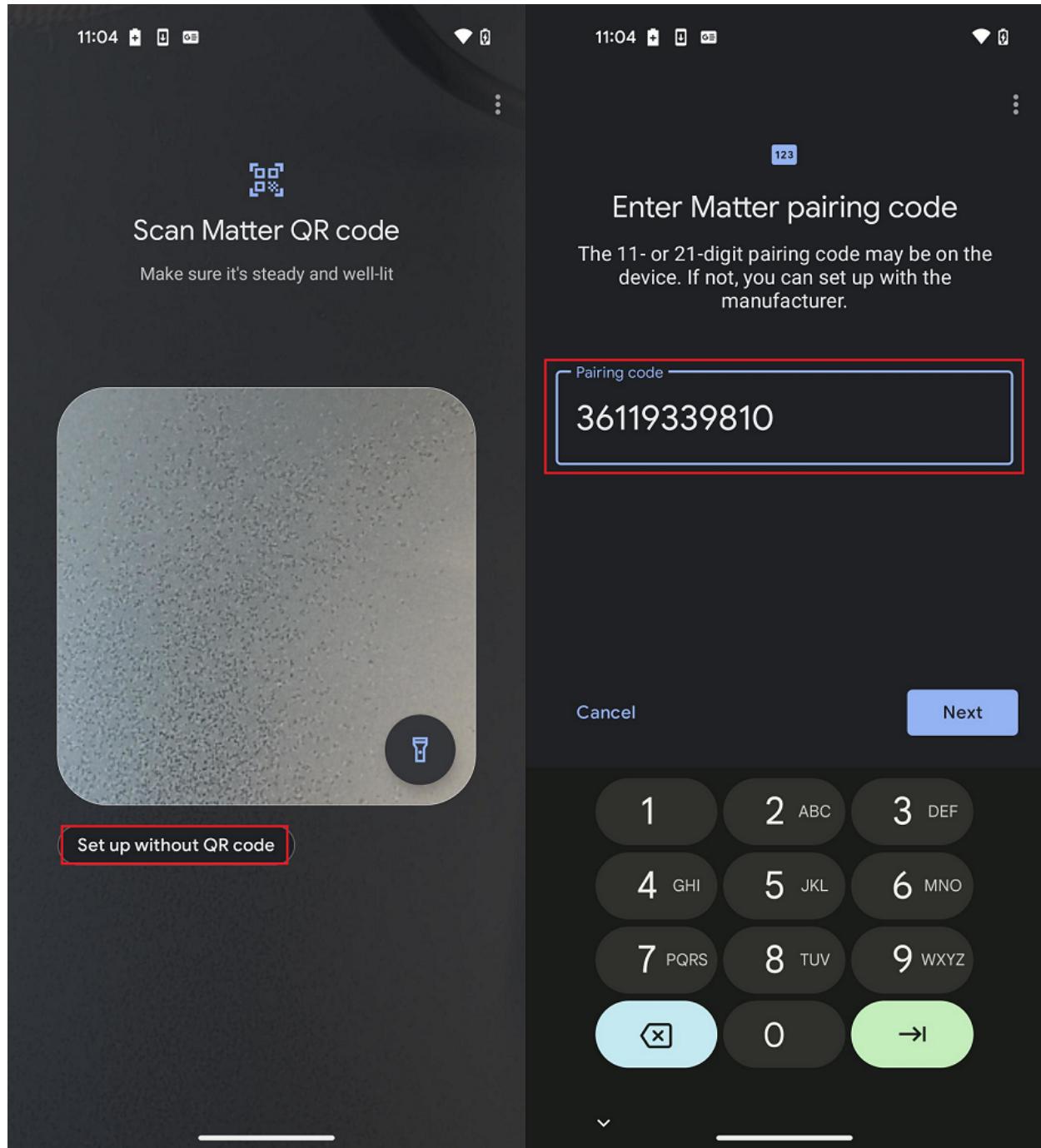
1. Open Apple Home App.
2. Click the Light Application to open the detailed view.
3. Scroll down at the bottom of the application detailed view.



4. Open the commissioning window for the Google Home Pod by clicking **Turn on Pairing Mode**.
5. The Setup Code appears, with a message printed as shown below in the Apple Home App.



6. Open Google Home App, click the **Devices Section**, and click **Add**.
7. Click **Set up without QR Code** and insert the setup code displayed in Step 5.



8. Commissioning will start and proceed in the same way as explained in [Google Ecosystem Setup and Demo Execution](#).
9. Once the commissioning is done, the new Matter Wi-Fi device should be visible in the Room view of the Google Home App.

Matter Bridge

The Unify Matter Bridge

The Unify Matter Bridge is an application that makes legacy devices, such as Z-Wave and Zigbee devices, accessible on a Matter fabric. It does so by acting as an *IoT Service* in a Unify Framework.

Support for the Unify Matter Bridge is NOT provided inside Simplicity Studio. The Unify Matter Bridge must be built out of the GitHub repo located here: [Silicon Labs Matter GitHub Repo](#). Documentation on using the Unify Matter Bridge is provided here, however the latest documentation on the Unify Matter Bridge is provided directly in the Matter GitHub Repo documentation located here: [Silicon Labs Unify Matter GitHub Documentation](#)

This section provides:

- An [overview](#) of the Unify Matter Bridge
- Instructions for [building the bridge](#)
- Instructions for [running the bridge](#)

Matter Bridge Overview

Unify Matter Bridge Overview

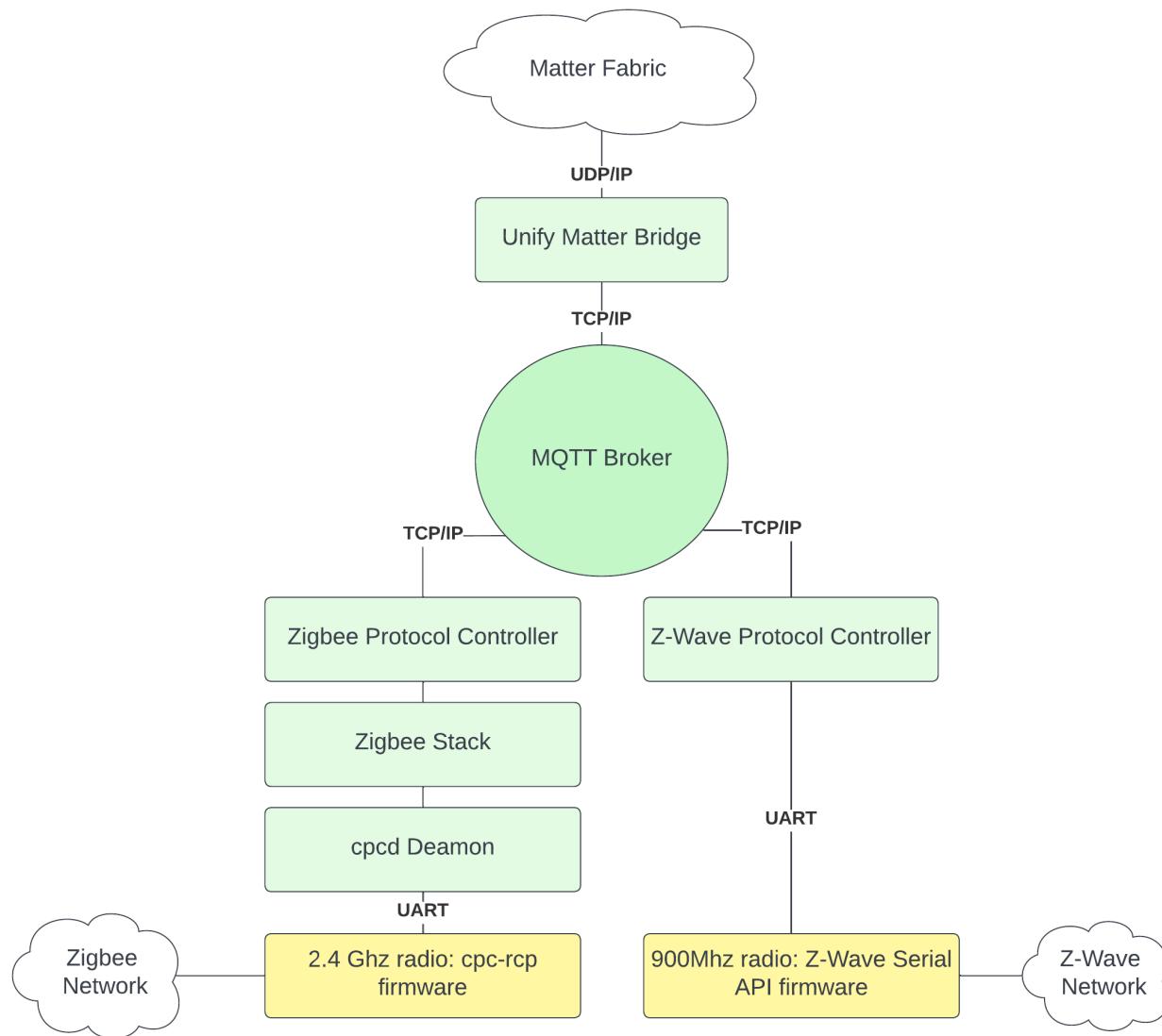
The Unify Matter Bridge is an application that makes legacy devices, such as Z-Wave and Zigbee devices, accessible on a Matter fabric. It does so by acting as an *IoT Service* in a Unify Framework.

In the Unify Framework, *protocol controllers* translate raw wireless application protocols such as Z-Wave and Zigbee into a common API called the Unify Controller Language (UCL). This enables IoT services to operate and monitor Z-Wave and Zigbee networks without being aware of the underlying wireless protocol.

In Unify, the transport between IoT services and Protocol Controllers is MQTT using JSON payloads for data representation.

On the Matter fabric, the Unify Matter Bridge is a Matter device that has dynamic endpoints, each representing an endpoint on one of the nodes in the Unify network. See the "Matter Specification" section "9.12. Bridge for non-Matter devices" for details.

The figure below illustrates the system architecture of the Unify Matter Bridge and two Unify protocol controllers.



More Information about the Unify Framework can be found [here](#)

Trying Out the Unify Matter Bridge

To test the Unify Matter Bridge, a Raspberry Pi 4 is recommended. Install the latest release of the Unify SDK following the [Unify Host SDK Getting Started Guide](#). Once the base Unify system is up and running, the Unify Matter Bridge may be installed on the Raspberry Pi 4.

The [Silicon Labs Matter GitHub release](#) contains ready-to-use binaries of the Unify Matter Bridge and the chip-tool.

Note that the Unify Host SDK uses Raspberry Pi OS as the base system as opposed to the standard Ubuntu system used for the Matter OpenThread Border Router image.

Unify Matter Bridge as an IoT Service

The Unify Matter Bridge is a Unify IoT Service that allows for control of Unify devices from a Matter fabric. It translates Matter cluster commands and attributes accesses into the corresponding Unify MQTT publish messages. Unify node attributes are readable from the Matter Fabric, for example by a Google Home App, as the Unify Matter Bridge also caches the state of those attributes.

The Unify data model is largely based on the same data model as Matter, making the job of the Unify Matter Bridge relatively simple. There is almost a 1-1 relationship between them.

Note: Currently no control goes out to the Matter Fabric from the Unify Matter Bridge. The Unify Matter Bridge cannot 'see' what else is on the Matter Fabric, let alone control end devices in the Matter Fabric.

See the [GitHub release notes](#) for details on feature additions, bug fixes, and known issues.

Supported Clusters/Devices

The Unify Matter Bridge currently supports mapping the following clusters/device types.

Cluster
Bridge Device Information
Level
OnOff
Identify
Group
Color Control
Occupancy Sensing
Temperature Measurement
Illuminance Measurement
Pressure Measurement
Flow Measurement
RelativeHumidity Measurement

Next Steps

- [Building the Matter Bridge](#)
- [Running the Matter Bridge](#)
- [Controlling a Z-Wave OnOff device](#)
- [Toggling a group of OnOff devices](#)

For more information about the Unify SDK see [Unify Host SDK Documentation](#)

Building The Matter Bridge

Building the Unify Matter Bridge

This build guide cross-compiles for arm64 architecture to be run on Unify's reference platform - a Raspberry Pi 4 (RPi4) with the 64-bit version of Debian Bullseye.

Note: In the following subsections the commands should either be run on your local development machine or inside a running Docker container, as distinguished by the structure of the example.

- *some-command* should be executed on your local machine.
 - *dev-machine:~\$ some-command*
- *some-other-command* should be executed inside the Docker container.
 - *root@docker:/<dir>\$ some-other-command*

Check Out Submodules

Assuming you have cloned the matter repo in ~/matter

Check out the necessary submodules with the following command.

```
dev-machine:~/matter$ ./scripts/checkout_submodules.py --platform linux
```

Clone and Stage the Unify SDK Repository

Assuming you have cloned the matter repo in ~/matter

```
dev-machine:~/matter$ git clone --depth 1 https://github.com/SiliconLabs/UnifySDK.git --recursive ../uic-matter
```

Build the Docker Container (arm64 compilation)

```
dev-machine:~/matter$ docker build -t unify-matter silabs_examples/unify-matter-bridge/docker/
```

Run the docker container (arm64 compilation)

Make sure the directory structure is like follows where Unify repo `uic-matter/` and matter repo `matter/` are at same directory level

```
.
├── matter
└── uic-matter
```

Start the docker from `matter/` directory where you cloned the matter repo: Here we assume `matter/` is in `~`

```
dev-machine:~$ cd matter/
dev-machine:~/matter$ docker run -it -v $PWD:/matter -v $PWD/..uic-matter:/uic unify-matter
```

Build libunify

The Unify Matter Bridge depends on the libunify library from the Unify project.

This library must first be compiled for the target system, by changing directory to the `/uic` directory and running the following:

```
root@docker:/uic$ cmake -DCMAKE_INSTALL_PREFIX=$PWD/stage -GNinja -DCMAKE_TOOLCHAIN_FILE=../cmake/arm64_debian.cmake -B build_unify_arm64/ -S components -DBUILD_TESTING=OFF
root@docker:/uic$ cmake --build build_unify_arm64
root@docker:/uic$ cmake --install build_unify_arm64 --prefix $PWD/stage
```

After cross-compiling the Unify library, set two paths in the `PKG_CONFIG_PATH`. The first path is for the staged Unify library and the second is for cross compiling to arm64.

```
root@docker:/uic$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$PWD/stage/share/pkgconfig
root@docker:/uic$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/lib/aarch64-linux-gnu/pkgconfig
```

If you want to be able to use Zap to generate code from Unify XML files you need to export `UCL_XML_PATH` as well.

```
root@docker:/uic$ export UCL_XML_PATH=$PWD/stage/share/uic/ucl
```

Run activate in matter

Once you have all the necessary submodules, source the Matter environment with the following command. This loads a number of build tools and makes sure the correct toolchains and compilers are used for compiling the Unify Matter Bridge.

Make sure you are in `matter/` directory

```
root@docker:/matter$ source ./scripts/activate.sh
```

Compile the Unify Bridge

```
root@docker:/matter$ cd silabs_examples/unify-matter-bridge/linux/
root@docker:/matter/silabs_examples/unify-matter-bridge/linux$ gn gen out/arm64 --args='target_cpu="arm64"'
root@docker:/matter/silabs_examples/unify-matter-bridge/linux$ ninja -C out/arm64
```

After building, the `unify-matter-bridge` binary is located at `/matter/silabs_examples/unify-matter-bridge/linux/out/arm64/obj/bin/unify-matter-bridge`.

Compile the chip-tool

The `chip-tool` is a CLI tool that can be used to commission the bridge and to control end devices.

```
root@docker:/matter$ cd examples/chip-tool
root@docker:/matter/examples/chip-tool$ gn gen out/arm64 --args='target_cpu="arm64"'
root@docker:/matter/examples/chip-tool$ ninja -C out/arm64
```

After building, the `chip-tool` binary is located at `/matter/examples/chip-tool/out/arm64/obj/bin/chip-tool`.

Unit Testing

Unit testing is always a good idea for quality software. Documentation on writing unit tests for the Matter Unify Bridge is in the [README.md](#) in the `linux/src/tests` folder.

Troubleshooting

1. If you do not source the `matter/scripts/activate.sh` as described above in [Set Up the Matter Build Environment](#), `gn` and other common build tools will not be found.
2. If you do not export the `pkgconfig` for the `aarch64-linux-gnu` toolchain as described above in [Build libunify](#) you will get errors such as `G_STATIC_ASSERT(sizeof (unsigned long long) == sizeof (quint64));`
3. If you are compiling unit tests, do not try to compile the Unify Matter Bridge at the same time. This will not work as when compiling unit tests you are also compiling unit tests for all other sub-components.
4. If you encounter errors linking to `libunify`, try redoing the `libunify` [compile steps](#).
5. Encountering problems with the submodules can be due to trying to check out the submodules inside the docker container.
6. If the Unify Matter Bridge gets stuck while booting. Try to pass `--args="chip_config_network_layer_ble=false"` to `gn gen` command while building

Running The Matter Bridge

Unify Matter Bridge User's Guide

The Unify Matter Bridge is a Unify IoT Service that enables interaction with Unify devices from a Matter fabric. For a more thorough description see the [Unify Matter Bridge Overview](#).

As a prerequisite for the Matter Bridge to work, at least one Unify protocol controller should be set up and running. This guide assumes that you have set up the Z-Wave Protocol Controller (uic-zpc) to run on a Raspberry Pi 4 and connected it to an MQTT broker in your network. Read the [Unify Host SDK's Getting Started Guide](#) for information on how to set this up.

Once a protocol controller is running, the Matter Bridge can be started.

The following documentation assumes that you have built the Unify Matter Bridge application by following the [Build Guide](#) and have transferred the `unify-matter-bridge` to your Raspberry Pi 4 (RPi4) running the 64-bit version of Raspberry Pi OS Bullseye.

- [Unify Matter Bridge User's Guide](#)
 - [Running the Matter Bridge](#)
 - [Important Configuration Settings](#)
 - [Starting the Matter Bridge](#)
 - [Commissioning the Bridge to a Network](#)
 - [Using the chip-tool to Commission](#)
 - [Using Google Nest Hub](#)
 - [Toggle an OnOff device](#)
 - [Toggle a Group of OnOff Devices](#)
 - [Running the matter bridge in strict device mapping mode](#)
 - [Command Line Arguments](#)
 - [Running chip-tool tests on Unify Matter Bridge endpoints](#)
 - [Troubleshooting](#)

Running the Matter Bridge

At start-up, the Matter Bridge needs to connect to the Matter Fabric as well as the MQTT Broker. It is therefore critical that you have access to port 1883, the default MQTT Broker's port, as well as a network setup that allows mDNS through.

A few important runtime configurations must be considered, along with some other configuration options. A full list of command-line parameters is provided in the [Command line arguments](#) section.

Important Configuration Settings

- Network Interface

Specify the network interface on which the Matter Fabric runs. In a regular RPi4 setup it would be `wlan0` for WiFi and `eth0` for ethernet. Specify this with the '`--interface`' argument, as such:

```
./unify-matter-bridge --interface eth0
```

- Key-Value store (KVS)

The Matter Bridge uses a Key-Value store for persisting various run-time configurations. Make sure to have read/write access to the default path '`/var/chip_unify_bridge.kvs`' or provide the path to where writing this data is allowed. If this file is deleted before start-up, everything is reset and the bridge will not belong to any Matter Fabric until it has again been commissioned.

```
./unify-matter-bridge --kvs ./matter-bridge.kvs
```

- MQTT Host

If you have followed the [Unify Host SDK's Getting Started Guide](#), your MQTT Broker should now be running on 'localhost'. If you have decided to run the MQTT broker on a different host, you can tell the Unify Matter Bridge to connect to a different host.

```
./unify-matter-bridge --mqtt.host 10.0.0.42
```

- Vendor and Product ID

If you have access to the EAP and you want to use the Google Home App, you need to set a specific VID and PID for the Matter Bridge.

```
./unify-matter-bridge --vendor fff1 --product 8001
```

Starting the Matter Bridge

Once the configuration parameters are set it is time to start the bridge application.

```
./unify-matter-bridge --interface eth0 --kvs ./matter-bridge.kvs --mqtt.host localhost --mqtt.port 1337
```

Commissioning the Bridge to a Network

To include the bridge in the Matter network, it must first be commissioned. The first time the bridge starts it will automatically go into commissioning mode. After 10 minutes the bridge will exit commissioning mode. If the bridge has not been commissioned within this window, the application must be restarted to open the commissioning window again or the window can be opened by writing `commission` in the CLI when running the bridge. The commission command may also be used for multi-fabric commissioning.

The Unify Matter Bridge uses the "On Network" commissioning method. For now, there is no Bluetooth commissioning support.

The commissioning procedure requires use of a pairing code. This pairing code is written to the console when running the Matter Bridge. Look for something similar to '`MT:-24J029Q00KA0648G00`', used as the pairing code in the following example. This code can be used when commissioning with the CLI commissioning tool `chip-tool`.

```
[1659615301.367669][1967:1967] CHIP:SVR: SetupQRCode: [MT:-24J029Q00KA0648G00]
```

Additionally the pairing code will be published on the MQTT Broker on the topic `ucl/SmartStart/CommissionableDevice/MT:-24J029Q00KA0648G00`. The Unify Developer GUI has a page which displays the QR Codes of all commissionable bridges which are connected to the broker, ready to be scanned with a Google Home App or similar.

Another way to get the QR code is to look for an URL in the console log similar to and copy the link into a browser. Note that two codes are printed at startup one for *Standard Commissioning flow* and one for custom commissioning flow. Be sure to use the standard flow with Eco system devices.

```
[1659615301.367723][1967:1967] CHIP:SVR: https://dhrishi.github.io/connectedhomeip/qrcode.html?data=MT%3A-24J029Q00KA0648G00
```

It should be noted that the commissioner **must** be on the same network as the Raspberry Pi. Note that by default the bridge binds to the `eth0` interface. If another interface is to be used, see the description of the command line arguments for setting [Network Interface](#).

Using the chip-tool to Commission

In the following procedure make sure to use the pairing code taken from the console output, as described above. To commission the Matter Bridge with the `chip-tool` and assign the bridge the Node ID 1:

```
chip-tool pairing code 1 MT:-24J0AFN00KA0648G00
```

Using Google Nest Hub

It is possible to use the Google Nest Hub (2nd. Gen) for controlling the Matter devices on the Unify Matter Bridge. Go through the following steps to configure this:

Prerequisites:

- Android Phone, Android 12 or newer
- Google Nest Hub, 2nd. Generation

Setup:

- Create a Google Account or using existing
- Go to [Google Developer Console](#)
 - Click "Create a new project"
 - Next page click "Create project"
 - Input a unique project name
 - Click "+ Add Matter integration"
 - Next: Develop
 - Next: Setup
 - Input following fields:
 - Product name of your choice
 - Device Type: Control Bridge
 - Test VID: default
 - Product ID (PID): 0x8001
 - Save & Continue
 - Save
- On your Android Phone
 - Configure the phone to use the same Google Account
 - Install Google Home application
 - To add the Nest Hub
 - Click "+" in the Google Home app - Add new device and let the phone search for your hub over BT - make sure hub is in reach
 - To add the Matter Bridge to your Google Home (and Hub)
 - In the console of the Matter Bridge application running on the Raspberry Pi
 - Hit `Return`, this should present `Unify>`
 - Type `commission`, this will show SetupQRCode, either click the link or in the Developer UI go to the "Commissionable Devices" page (note that a new QR code will be created whenever bridge is restarted, make sure to use the latest as identified in the output on the console)
 - In the Google Home app click "+"
 - Google Home should report "Matter-enabled device found"
 - If the bridge is not automatically found, a list of device types will be shown, click the "Matter-device" on the list
 - Google Home will now ask for scanning the QR code - scan the QR code as described above
 - If Google Home is stuck during commissioning, type `commission` again in the Matter Bridge console while Google Home is waiting
 - All supported Unify devices should now be available for control in both Google Home application as well as the Google Nest Hub
 - On the Nest Hub, swipe down from the top of the display or select "Home Control" to access the devices

Toggle an OnOff device

To send an OnOff cluster Toggle command to a bridged endpoint with id 2, via Matter Fabric Node ID 1:

```
chip-tool onoff toggle 1 2
```

For more information on how to use the `chip-tool` see the [chip-tool manual](#) on the Matter website.

Toggle a Group of OnOff Devices

The Matter Bridge has support for forwarding group messages from the Matter Fabric to Unify Nodes. The protocol controllers will send the group messages as actual group cast messages on the destination network (Z-Wave/Zigbee).

To send a group command, first set up the group keys in the bridge. This example assumes the bridge to be Node ID 1, and GroupKeySetID 42 is added to Group ID 1:

```
chip-tool accesscontrol write acl '[{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [112233], "targets": null }, {"fabricIndex": 1, "privilege": 4, "authMode": 3, "subjects": [1], "targets": null }]' 1 0
chip-tool groupkeymanagement key-set-write '{"groupKeySetID": 42, "groupKeySecurityPolicy": 0, "epochKey0": "d0d1d2d3d4d5d6d7d8d9dadbdccccdf", "epochStartTime0": 2220000, "epochKey1": "d1d1d2d3d4d5d6d7d8d9dadbdccccdf", "epochStartTime1": 2220001, "epochKey2": "d2d1d2d3d4d5d6d7d8d9dadbdccccdf", "epochStartTime2": 2220002}' 1 0
chip-tool groupkeymanagement write group-key-map '[{"groupId": 1, "groupKeySetID": 42, "fabricIndex": 1}]' 1 0
```

Next, add bridge endpoint 2 to Group ID 0x0001

```
chip-tool groups add-group 0x0001 grp1 1 2
```

Next, program the chip-tool:

```
chip-tool groupsettings add-group grp1 0x0002
chip-tool groupsettings add-keysets 0x0042 0 0x000000000021dfe0 hex:d0d1d2d3d4d5d6d7d8d9dadbdccccdf
chip-tool groupsettings bind-keyset 0x0001 0x0042
```

Finally, a multicast command may be sent using the chip-tool.

```
// Send actual multicast command
./chip-tool onoff toggle 0xfffffffffffff0001 1
```

Running the matter bridge in strict device mapping mode

By default Unify Matter Bridge tries and map devices that does not necessarily conform to the Matter specificaiton.

To enable a mode where Unify Matter Bridge strictly only maps devices from the Unify Controller Language protocol to the Matter protocol that complies with the Matter specification. You can run the bridge with the command line argument

```
./unify-matter-bridge --strict_device_mapping true .
```

Command Line Arguments

The Unify Matter Bridge provides the following command line arguments:

Using `--help` displays the following text.

```
Usage: ./unify_matter_bridge [Options]

Options:
--conf arg (=etc/uic/uic.cfg)      Config file in YAML format. UIC_CONF
                                         env variable can be set to override the
                                         default config file path
--help                             Print this help message and quit
--dump-config                      Dump the current configuration in a
                                         YAML config file format that can be
                                         passed to the --conf option
--version                          Print version information and quit
```

The following options can also be in a config file. Options and values passed on the command line take precedence over the options and values in the config file.

```
--log_level arg (=i)          Log Level (d,i,w,e,c)
--log_tag_level arg           Tag-based log level
                             Format: <tag>:<severity>,
                             <tag>:<severity>, ...
--interface arg (=en0)        Ethernet interface to use
--kvs arg (=~/var/chip_unify_bridge.kvs)
                             Matter key value store path
--vendor arg (=65521)         Vendor ID
--product arg (=32769)         Product ID
--mqtt.host arg (=localhost)  MQTT broker hostname or IP
--mqtt.port arg (=1883)        MQTT broker port
--mqtt.cafile arg             Path to file containing the PEM-encoded
                             CA certificate to connect to Mosquitto
                             MQTT broker for TLS encryption
--mqtt.certfile arg           Path to file containing the PEM-encoded
                             client certificate to connect to
                             Mosquitto MQTT broker for TLS
                             encryption
--mqtt.keyfile arg            Path to a file containing the PEM-
                             encoded unencrypted private key for
                             this client
--mqtt.client_id arg (=unify_matter_bridge_71460)
                             Set the MQTT client ID of the
                             application.
```

Running chip-tool tests on Unify Matter Bridge endpoints

For e.g. OnOff Cluster chip-tool test Test_TC_OO_2_3 can be ran on Unify Matter Bridge endpoint 2 for node 1 as follows.

```
./chip-tool tests Test_TC_OO_2_3 --nodeId 1 --endpoint 2 --delayInMs 1400
```

Mapping of Matter Endpoint to Unify Node IDs can be seen by giving "epmap" command to Unify matter bridge command prompt as follows

```
Unify>epmap
Unify Unid |Unify Endpoint |Matter Endpoint
zw-CE7F3772-0008|      | 2
```

Note: Endpoint 0(Root) and Endpoint 1(Aggregator) are Bridge itself. They are not shown in the epmap. But these endpoints support some clusters as well. For e.g. Identify. Where you might want to run chip-tool tests on those endpoints

For further information on chip-tool tests, refer to the [test suite's README](#)

Troubleshooting

- Time sensitive chip-tool tests might fail because of the latencies in Unify Matter Bridge. The '`--delayInMs <number of milliseconds>`' command line option to chip-tool can be helpful in such cases.
- The Unify Matter Bridge needs to be commissioned to the Matter fabric before running the tests. Or you will see following message on the chip-tool tests

```
***** Test Step 0 : 1: Wait for the commissioned device to be retrieved
```

- To run all the tests without exiting on a failed one, '`--continueOnFailure true`' can be used.
- Every cluster command sent by chip-tool can be seen on Unify Matter bridge as MQTT topic publish as follows

for .e.g if chip-tool sends OnOff On command on zw-CE7F3772-0008 Unify end node(Matter Endpoint 2). Then Unify Matter bridge publishes following MQTT payload and topic

```
2023-Jan-30 10:36:43.803360 <d> [command_translator_interface] --- send_unify_mqtt_cmd ucl/by-unid/zw-CE7F3772-0008/ep0/OnOff/Commands/On → {} ---
```

The above MQTT debug message can also be traced in Unify logs.

- Every attribute read sent by chip-tool will only get correct value, if Unify Matter Bridge publishes MQTT payload and topic like following, before chip-tool sends the attribute read command.

For e.g. if chip-tool tries to read OnOff attribute of OnOff cluster on zw-CE7F3772-0008 Unify end node(Matter Endpoint 2) Unify Matter Bridge must have received following kind of MQTT message before the correct attribute value will be reflected in Unify Matter Bridge.

```
2023-Jan-30 10:36:44.515748 <d> [mqtt_client] mqtt_client::on_message: ucl/by-unid/zw-CE7F3772-0008/ep0/OnOff/Attributes/OnOff/Reported, {"value":true}, 0
```

The above MQTT debug messages can also be traced in Unify logs.

- Alternatively to disable a particular command or disable reading particular attributes from the test refer to PICS Usage from [README](#)

Overview Guides

Overview Guides

The Overview pages offer a general review of Matter topics including:

- [Matter Provisioning](#)
- [Test Matter Certificates for Development](#)
- [Matter Commissioning](#)
- [Matter Intermittently Connected Devices \(ICD\)](#)
- [Matter Over Thread ICD End Devices](#)
- [Matter Serial Port Communication](#)
- [Matter SLC CLI Setup and Build Instructions](#)
- [Matter Solutions](#)

Matter Provisioning

Matter Provisioning

Tools in the Silicon Labs Matter GitHub `provision` folder are used to load mandatory authentication information into Matter devices. For more information on accessing the provision tools and cloning the Silicon Labs Matter GitHub repository, see the documentation located here: [Silicon Labs Matter GitHub repo](#).

Most of the required parameters are stored once during the manufacturing process, and shall not change during the lifetime of the device. During runtime, two interfaces are used to pull the authentication data from permanent storage:

- [CommissionableDataProvider](#), implemented as [EFR32DeviceDataProvider](#)
- [DeviceAttestationCredsProvider](#), implemented as [SilabsDeviceAttestationCreds](#)

The provisioning script in this folder now supersedes the following tool:

- [Factory Data Provider](#)

Provisioned Data

The Commissionable Data includes Serial Number, Vendor Id, Product Id, and the Setup Payload (typically displayed in the QR), while the Attestation Credentials includes the Certificate Declaration (CD), the Product Attestation Intermediate (PAI) certificate, and the DAC (Device Attestation Certificate).

During commissioning, Matter devices perform a Password Authenticated Key Exchange using the SPAKE2+ protocol. The SPAKE2+ verifier is pre-calculated [using an external tool](#).

The passcode is used to derive a QR code, typically printed on the label, or displayed by the device itself. The QR code contains the pre-computed setup payload, which allows the commissioner to establish a session with the device. The parameters required to generate and validate the session keys are static and stored in NVM3.

To protect the attestation private-key (used to generate the DAC), the asymmetric key-pair should be generated on-device, using PSA, and the most secure storage location available to the specific part. However, the private-key may be generated externally, and imported using the `--dac_key` parameter.

The DAC is generated and signed by a Certification Authority (CA), which may reside on a separate host. The `modules/signing_server.py` script simulates the role of the CA, and uses OpenSSL to generate and sign the DAC. In a real factory environment, this script is replaced by an actual CA.

Generator Firmware

The Generator Firmware (GFW) is a baremetal application that runs on the targeted device, and assists with the initial setup of the device. The GFW performs the following tasks:

- Generates the device key-pair on the most secure location available
- Returns a CSR (Certificate Signing Request) to the provisioning script. The CSR contains the device public-key, Vendor Id, Product Id, and Serial Number.
- Calculates the Setup Payload
- Stores the Commissionable Data into NVM3 (including the Setup Payload)
- Stores the Attestation Data on the main flash (CD, PAI, DAC)
- Stores the size and offsets used to store the Attestation Data, along with the KeyId used to generate the private-key

The main source code of the GFW is located under `provision/generator`, while the board support is located under `provision/support`. Pre-compiled images for the supported chips can be found in `provision/images`.

The directory structure is as follows:

- provision
 - generator
 - images
 - modules
 - support
 - efr32mg12
 - efr32mg24

Provisioner Script

The `provision.py` file is the main script used to load all the required data on the Matter device. This script requires:

- [Simplicity Commander](#)
- [SEGGER J-Link](#)
- [SPAKE2+ generator](#)
- [PyLink](#)

```
sudo pip3 install ecdsa
```

The Provisioner Script executes the following steps:

1. Parses and validates the command-line arguments
2. Obtains the Part Number from the connected device (using Simplicity Commander)
3. If no SPAKE2+ verifier is provided:
 - 3.1. Generates SPAKE2+ verifier (using the external `spake2p` tool)
4. Loads the Generator Firmware into the device (if no GFW path is provided, the Part Number is used to choose the corresponding file from the `provision/images`)
5. Configures the NVM3 based on the flash size of the connected device
6. If CSR mode is used (`--csr`):
 - 6.1. Requests a CSR from the device - The GFW generates the key-pair and CSR, then returns the CSR to the host script
 - 6.2. Sends the CSR to the Signing Server (`provision/modules/signing_server.py`), and retrieves the DAC
7. Sends CD, PAI, and DAC to the GFW - The GFW stores CD, PAI, and DAC on the last page of main flash, and updates the offsets and sizes in NVM3
8. Sends the Commissionable Data to the GFW - The GFW initializes the flash, generates the Setup Payload, and stores the data into NVM3
9. If a PFW is provided, writes the PFW into flash using Simplicity Commander

The provisioning script and the GFW communicates through J-Link RTT using the PyLink module.

Arguments

Arguments	Conformance	Type	Description
<code>-c, --config</code>	optional	string	Path to a JSON configuration file
<code>-j, --jlink</code>	optional ^1	dec/hex	JLink connection string.
<code>-l, --pylink_lib</code>	optional	string	Path to the PyLink library.
<code>-g, --generate</code>	optional	flag	Auto-generate test certificates
<code>-m, --stop</code>	optional	flag	Stop mode: When true, only generate the JSON configuration, and exit.
<code>-r, --csr</code>	optional	flag	CSR mode: When true, instructs the GFW to generate the private key, and issue a CSR.
<code>-gf, --gen_fw</code>	optional	dec/hex	Path to the Generator Firmware image.
<code>-pf, --prod_fw</code>	optional	dec/hex	Path to the Production Firmware image.
<code>-v, --vendor_id</code>	optional	dec/hex	Vendor ID. e.g: 65521 or 0xFFFF (Max 2 bytes).
<code>-V, --vendor_name</code>	optional	string	Vendor name (Max 32 char).

Arguments	Conformance	Type	Description
-p, --product_id	optional	dec/hex	Product ID. e.g: 32773 or 0x8005 (Max 2 bytes).
-P, --product_name	optional	string	Product name (Max 32 char).
-pl, --product_label	optional	string	Product label.
-pu, --product_url	optional	string	Product URL.
-pn, --part_number	optional	dec/hex	Device Part Number (Max 32 char).
-hv, --hw_version	optional	dec/hex	The hardware version value (Max 2 bytes).
-hs, --hw_version_str	optional	string	The hardware version string (Max 64 char).
-cf, --commissioning_flow	optional	dec/hex	Commissioning Flow 0=Standard, 1=User Action, 2=Custom.
-rf, --rendezvous_flags	optional	dec/hex	Rendez-vous flag: 1=SoftAP, 2=BLE 4=OnNetwork (Can be combined).
-md, --manufacturing_date	optional	string	Manufacturing date.
-d, --discriminator	optional ^2	dec/hex	BLE pairing discriminator. e.g: 3840 or 0xF00. (12-bit)
-ct, --cert_tool	optional	string	Path to the chip-cert tool. Defaults to ../out/tools/chip-cert
-ki, --key_id	required	dec/hex	Attestation Key ID.
-kp, --key_pass	optional ^3	string	Password for the key file.
-xc, --att_certs	optional ^3	string	Path to the PKCS#12 attestation certificates file.
-ic, --pai_cert	required	string	Path to the PAI certificate.
-dc, --dac_cert	optional ^3	string	Path to the PAI certificate.
-dk, --dac_key	optional ^3	dec/hex	Path to the PAI private-key.
-cd, --certification	required	string	Path to the Certification Declaration (CD) file.
-cn, --common_name	optional ^4	string	Common Name to use in the Device Certificate (DAC) .
-u, --unique_id	optional ^5	hex string	A 128 bits hex string unique id (without 0x).
-sv, --spake2p_verifier	optional	string ^6	Pre-generated SPAKE2+ verifier.
-sp, --spake2p_passcode	required	dec/hex	Session passcode used to generate the SPAKE2+ verifier.
-ss, --spake2p_salt	required	string ^6	Salt used to generate the SPAKE2+ verifier.
-si, --spake2p_iterations	required	dec/hex	Iteration count used to generate the SPAKE2+ verifier.

^{^1} Use xxxxxxxx for serial, or xxx.xxx.xxx.xxx[:yyyy] for TCP. ^{^2} If not provided (or zero), the `discriminator` is calculated as the last 12 bits of SHA256(serial_number) ^{^3} If the DAC is provided, its corresponding private-key also must be provided ^{^4} Required if the DAC is not provided ^{^5} If not provided, the `unique_id` is calculated as the first 128 bits of SHA256(serial_number) ^{^6} Salt and verifier must be provided as base64 string

For the hex type, provide the value with the `0x` prefix. For hex string type, do not add the `0x` prefix.

The `-c`--config argument allows to read all the required parameters from a JSON file. The same validation rules apply both for command line or configuration file, but JSON does not support hexadecimal numbers. Command line arguments override arguments read from a configuration file. For instance, with the configuration `example.json`:

```
{
  "version": "1.0",
  "matter": {
    "prod_fw": "/git/matter/out/lighting-app/BRD4187C/chip-efr32-lighting-example.s37",
    "vendor_id": 4169,
    "product_id": 32773,
    "discriminator": 3841,
    "attestation": {
      "key_id": "00000000000000000000000000000000"
    }
  }
}
```

```

    "dac_key": "temp/certs/dac_key.pem",
    "pai_cert": "temp/certs/pai_cert.pem",
    "certification": "temp/certs/cd.der",
  },
  "spake2p": {
    "passcode": 62034001,
    "salt": "95834coRGvFhCB69ldmJyr5qYlzFgSirw6Ja7g5ySYA",
    "iterations": 15000
  }
}
}

```

You may run:

```
python3 ./provision.py -c example.json -d 2748 -p 0x8006 -si 10000
```

Which will set the connected device with discriminator 2748 (instead of 3841), product ID 32774 (instead of 32773), and use 10000 SPAKE2+ iterations (instead of 15000).

To ease development and testing, the `provision.py` script provides defaults for most of the parameters. The only arguments that are truly mandatory are `vendor_id`, and `product_id`. Test certificates may be auto-generated using the `-g` flag, provided the `chip-cert` can be found, either in the default location, or through the `--cert-tool` argument. For instance, you may run:

```
python3 ./provision.py -v 0x1049 -p 0x8005 -g
```

Which will generate the test certificates using `chip-cert`, and set the device with the following parameters:

```

{
  "version": "1.0",
  "matter": {
    "generate": true,
    "vendor_id": 65522,
    "product_id": 32773,
    "discriminator": 3840,
    "attestation": {
      "cert_tool": "./out/tools/chip-cert",
      "key_id": 2,
    },
    "spake2p": {
      "verifier": null,
      "passcode": 62034001,
      "salt": "95834coRGvFhCB69ldmJyr5qYlzFgSirw6Ja7g5ySYA",
      "iterations": 15000
    }
  }
}

```

For each run, `provision.py` will generate the file `provision/config/latest.json`, containing the arguments used to set up the device. A default configuration with developer settings can be found at `provision/config/develop.json`:

```
python ./provision.py -c config/develop.json
```

Attestation Files

The `--generate` option instructs the `provider.py` script to generate test attestation files with the given Vendor ID, and Product ID. These files are generated using [the chip-cert tool](#), and stored under the `provision/temp` folder.

To generate the certificates manually:

```
./chip-cert gen-cd -f 1 -V 0xffff1 -p 0x8005 -d 0x0016 -c ZIG20142ZB330003-24 -l 0 -i 0 -n 257 -t 0 -o 0xffff1 -r 0x8005 -C ./credentials/test/certification-declaration/Chip-Test-CD-Signing-Cert.pem -K ./credentials/test/certification-declaration/Chip-Test-CD-Signing-Key.pem -O ./temp/cd.der

./chip-cert gen-att-cert -t a -l 3660 -c "Matter PAA" -V 0xffff1 -o ./temp/paa_cert.pem -O ./temp/paa_key.pem

./chip-cert gen-att-cert -t i -l 3660 -c "Matter PAI" -V 0xffff1 -P 0x8005 -C ./temp/paa_cert.pem -K ./temp/paa_key.pem -o ./temp/pai_cert.pem -O ./temp/pai_key.pem

./chip-cert gen-att-cert -t d -l 3660 -c "Matter DAC" -V 0xffff1 -P 0x8005 -C ./temp/pai_cert.pem -K ./temp/pai_key.pem -o ./temp/dac_cert.pem -O ./temp/dac_key.pem
```

NOTE: The commissioning fails if the commissioner do not recognize the root certificate (PAA). When using [chip-tool](#), you can use the `--paa-trust-store-path` to enabled the PAA certificates for testing purposes.

Example

In Simplicity Studio, add the Attestation Certificate provisioning configuration component to the project and build it. The resulting s37 file is used as an input to the `provision.py` script in the next step.

Set up the device with key generation:

```
python3 ./provision.py -v 0x1049 -p 0x8005 \
-r -ki 2 -cn "Silabs Device" -ic ./temp/pai_cert.pem -cd ./temp/cd.der \
-sp 62034001 -ss 95834coRGvFhCB69ldmJyr5qYlzFgSirw6Ja7g5ySYA= -si 15000 \
-d 0xf01 -j 440266330 -pf chip-epr32-lighting-example.s37
```

Or, set up the device with imported key:

```
python3 ./provision.py -v 0x1049 -p 0x8005 \
-ki 2 -cn "Silabs Device" -dc ./temp/dac_cert.pem -dk ./temp/dac_key.pem -ic ./temp/pai_cert.pem -cd ./temp/cd.der \
-sp 62034001 -ss 95834coRGvFhCB69ldmJyr5qYlzFgSirw6Ja7g5ySYA= -si 15000 \
-d 0xf01 -j 440266330 -pf chip-epr32-lighting-example.s37
```

Self-Provisioning

Silicon Labs' Matter examples include the same provisioning engine used by the GFW. This allows applications to be flashed once but provisioned multiple times. There are two ways to put the application in provisioning mode:

- Factory-reset by pressing both BTN0 and BTN1 for six seconds
- Write 1 to the NVM3 key 0x87228. This is useful in boards with less than two buttons, and can be accomplished using Simplicity Commander:

```
commander nvm3 read -o ./temp/nvm3.s37
commander nvm3 set ./temp/nvm3.s37 --object 0x87228:01 --outfile ./temp/nvm3+.s37
commander flash ./temp/nvm3+.s37
```

Once in provisioning mode, the example firmware can be used as GFW, for instance:

```
python3 provision.py -c config/develop.json -gf ../out/light/BRD4187C/matter-silabs-lighting-example.s37
```

If the device was factory-reset, it becomes ready for commissioning right after self-provisioning.

Validation

If the certificate injection is successful, the commissioning process should complete normally. In order to verify that the new certificates are actually being used, first check the last page of the flash using Commander. The content of the flash must then be compared with the credentials received by the commissioner, which can be done using a debugger.

Flash Dump

On EFR32MG12, the last page starts at address 0x000FF800. On EFR32MG24, the last page is located at 0x0817E000. These addresses can be found in the memory map of the board's datasheet. For instance, for a MG24 board:

```
commander readmem --range 0x0817E000:+1536 --serialno 440266330`
```

The output should look something like:

```
commander readmem --range 0x0817E000:+1536 --serialno 440266330
Reading 1536 bytes from 0x0817e000...
{address: 0 1 2 3 4 5 6 7 8 9 A B C D E F}
000ff800: 30 82 01 D8 30 82 01 7F A0 03 02 01 02 02 04 07
000ff810: 5B CD 15 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30
...
000ff9c0: 2B BA 15 32 2F 4C 69 F2 38 48 D2 BC 62 2A 47 FB
000ff9d0: 3F F7 28 8A 7C 90 75 72 58 84 96 E7 00 00 00 00
000ff9e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ff9f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ffa00: 30 82 01 C8 30 82 01 6E A0 03 02 01 02 02 08 79
000ffa10: 6E 32 5A FA 5B D1 F8 30 0A 06 08 2A 86 48 CE 3D
...
000ffbb0: FD 92 D1 EB 59 95 D8 38 DE 5D 80 E3 05 65 24 4A
000ffbc0: 62 FD 9F E9 D8 00 FA CD 0F 32 7C C9 00 00 00 00
000ffbd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ffbe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ffbfo: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ffc00: 30 81 EF 06 09 2A 86 48 86 F7 0D 01 07 02 A0 81
000ffc10: E1 30 81 DE 02 01 03 31 0D 30 0B 06 09 60 86 48
...
000ffce0: 28 41 FD B8 28 CD 19 F2 BB DB A0 0F 33 B2 21 D3
000ffcf0: 33 CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000ffd00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

On this example, the DAC is located at address 0817e000 (offset 0), and has 476 octets:

```
0817e000: 30 82 01 D9 30 82 01 7F A0 03 02 01 02 02 04 07
0817e010: 5B CD 15 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30
...
0817e1c0: 2E 4F 10 20 38 BA A6 B5 F6 A4 77 7A 19 91 23 79
0817e1d0: 2F A0 FF AF F5 5C A1 59 98 08 C7 BC 5F 00 00 00
```

This should match the contents of the DER-formatted DAC certificate, which is stored by the setup script as ./temp/dac.der :

```
$ xxd ./temp/dac.der
```

```
00000000: 3082 01d8 3082 017f a003 0201 0202 0407 0...0.....
00000010: 5bcd 1530 0a06 082a 8648 ce3d 0403 0230 [..0...*H.=...0
...
000001c0: 2bba 1532 2f4c 69f2 3848 d2bc 622a 47fb +..2/Li.8H..b*G.
000001d0: 3ff7 288a 7c90 7572 5884 96e7 ?.(.|.urX...
```

The PAI certificate is located at address 0x0817e200 (offset 512), and has 460 octets:

```
0817e200: 30 82 01 C8 30 82 01 6E A0 03 02 01 02 02 08 79
0817e210: 6E 32 5A FA 5B D1 F8 30 0A 06 08 2A 86 48 CE 3D
...
0817e3b0: FD 92 D1 EB 59 95 D8 38 DE 5D 80 E3 05 65 24 4A
0817e3c0: 62 FD 9F E9 D8 00 FA CD 0F 32 7C C9 00 00 00 00
```

This should match the contents of the DER-formatted PAI certificate, which is stored by the setup script as ./temp/pai_cert.der :

```
$ xxd ./temp/pai_cert.der
```

```
00000000: 3082 01c8 3082 016e a003 0201 0202 0879 0...0..n.....y
00000010: 6e32 5afa 5bd1 f830 0a06 082a 8648 ce3d n2Z.[..0...*H.=
...
000001b0: fd92 d1eb 5995 d838 de5d 80e3 0565 244a ....Y..8.].e$J
000001c0: 62fd 9fe9 d800 facd 0f32 7cc9 b.....2|.
```

Finally, on this example the CD is located at address 0817e400 (offset 1024), and contains 541 octets:

```
0817e400: 30 81 EF 06 09 2A 86 48 86 F7 0D 01 07 02 A0 81
0817e410: E1 30 81 DE 02 01 03 31 0D 30 0B 06 09 60 86 48
...
0817e4d0: 02 20 38 B9 9C 73 B2 30 92 D7 A2 92 47 30 14 F7
0817e4e0: 28 41 FD B8 28 CD 19 F2 BB DB A0 0F 33 B2 21 D3
0817e4f0: 33 CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The CD is a binary file, and is neither modified, nor validated by the setup script. It is simply stored in flash after the DAC:

```
$ xxd cd.der
```

```
00000000: 3081 ef06 092a 8648 86f7 0d01 0702 a081 0....*H.....
00000010: e130 81de 0201 0331 0d30 0b06 0960 8648 .0....1.0...`H
...
000000e0: 2841 fdb8 28cd 19f2 bbdb a00f 33b2 21d3 (A..(.....3!.|
000000f0: 33ce 3.
```

The 0xff octets between the files and at the end of the flash are unmodified sections of the flash storage.

Device Terminal

Logs have been added to the SilabsDeviceAttestationCreds, to help verifying if the Attestation files are loaded correctly. The size and first eight bytes of CD, PAI, and DAC are printed, and must match the contents of `cd.der`, `pai_cert.der`, and `dac.der`, respectively:

```
...
[00:00:05.109][info ][ZCL] OpCreds: Certificate Chain request received for PAI
[00:00:05.109][info ][DL] GetProductAttestationIntermediateCert, addr:0xffa00, size:460
[00:00:05.110][detail][ZCL] 0x30, 0x82, 0x01, 0xc8, 0x30, 0x82, 0x01, 0x6e,
...
[00:00:05.401][info ][ZCL] OpCreds: Certificate Chain request received for DAC
[00:00:05.402][info ][DL] GetDeviceAttestationCert, addr:0xff800, size:477
[00:00:05.402][detail][ZCL] 0x30, 0x82, 0x01, 0xd8, 0x30, 0x82, 0x01, 0x7f,
...
[00:00:05.694][info ][ZCL] OpCreds: Received an AttestationRequest command
[00:00:05.695][info ][DL] GetCertificationDeclaration, addr:0xffc00, size:242
[00:00:05.695][detail][ZCL] 0x30, 0x81, 0xef, 0x06, 0x09, 0x2a, 0x86, 0x48,
...
```

Board Support

Pre-compiled images of the Generator Firmware can be found under provision/images. The source code of these images is found under provision/support. A single image is provided for all EFR32MG12 parts, and another one for the EFR32MG24 family. To copy with the different flash sizes, the `provision.py` script reads the device information using `commander`, and send it to the GFW, which configures the NVM3 during the initialization step.

Test Matter Certificates for Development

Test Matter Certificates for Development

Hardware and Software Requirements

- Commissioner
 - Raspberry Pi flashed with the Matter Hub Image
 - Setting Up your Matter Hub: [Developing with Silicon Labs Matter](#)
 - WSTK with supported boards for RCP
 - How to set up your RCP: [Developing with Silicon Labs Matter](#)
- Commissionee
 - WSTK with MG24A or MG24B (Initial boards supported for Matter CPMS Alpha program). The provisioning script also supports MG12. In this tutorial, you will build an application with a BRD4187C.

Introduction to Provisioning

Matter devices require a minimum amount of data that should be installed. Besides the Vendor ID, Product ID, Discriminator, and other device information, session establishment requires a CD (Certification Declaration), PAI (Product Attestation Intermediate certificate), and DAC (Device Attestation Certificate). There is no requirement about how or where this data is stored, but the Matter stack uses three distinct interfaces to retrieve it:

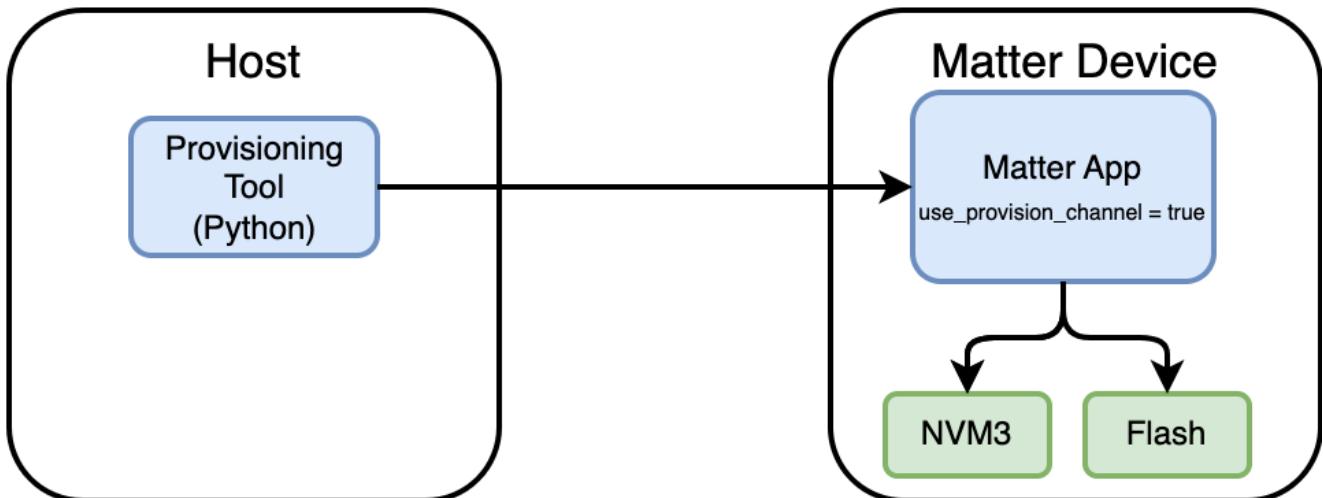
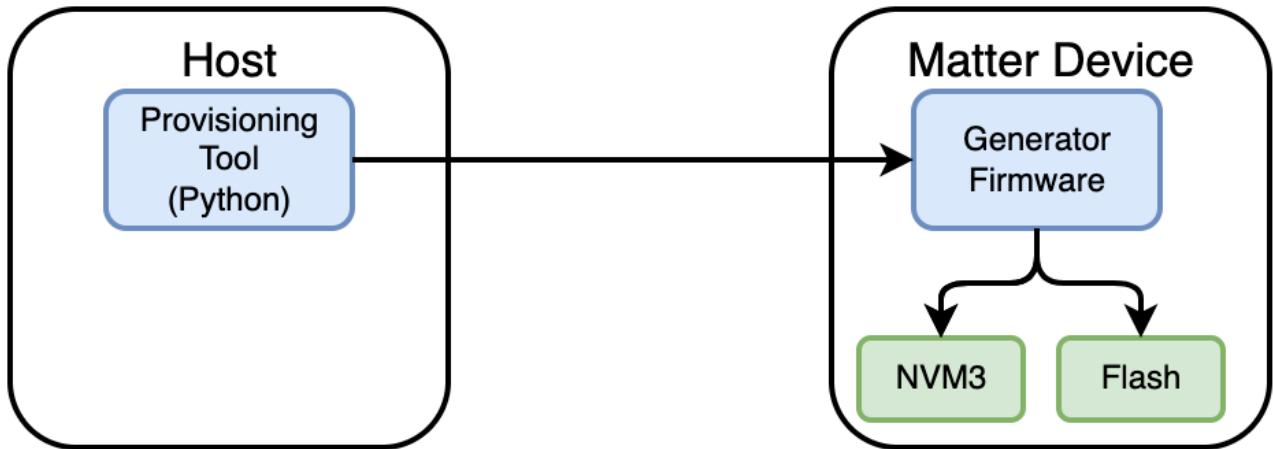
- [DeviceInstanceInfoProvider](#)
- [CommissionableDataProvider](#)
- [DeviceAttestationCredsProvider](#)

There is a long list of parameters that may be loaded on the device in-factory, and there are also complex dependencies between different arguments and files. For instance, Vendor ID must be returned by the DeviceInstanceInfoProvider interface, but Vendor ID must also be present in the CD and DAC and must match in all instances. Furthermore, each DAC must be loaded alongside its private key, which is used to sign outgoing messages. The secrecy of the private key is critical for the security of the inter-device communication; therefore, measures must be taken to limit access to this key, and this key should be stored in the most secure part of your device.

In a production environment, different devices should have unique identifiers, passcodes, discriminators, private-keys and DACs, which implies customization during the manufacturing process. To ease the development and manufacturing of customized devices, Silicon Labs provides the [Custom Part Manufacturing Service \(CPMS\)](#).

In a development environment, Silicon Labs provides a Provisioning Tool that is used to generate a Matter Certificate Chain based on Test or Development Certificates provided by the CSA. These certificates can have data modified such as VID, PID, Discriminator, Passcode, etc. For more information and arguments to this tool, refer to [Using CPMS](#).

To provide flexibility, Silicon Labs provides two ways to write Commissionable Data and Device Attestation Credentials to Matter Devices. The following figure depicts these two flows:



These two provisioning flows use the same Python Provisioning Tool script to initiate writing the Commissionable Data and the Device Attestation Data. The Generator Firmware and the Matter Sample application have the ability to store all the required data for a successful commissioning.

The use of a Generator Firmware is intended for Provisioning at Manufacturing and to reduce time writing Matter Credentials as there is no need for a bootloader or Matter Application.

When using a Matter Sample Application for provisioning, the user must ensure that the Matter Device has the flag `use_provision_channel = true` upon booting the device. The provision flag may be set to 1 at runtime. This may be accomplished in three ways:

- On example application with two buttons, factory reset by pressing BTN0 and BNT1 at the same time. Besides the regular factory reset, the provision flag is set to 1, and the device will wait for provisioning upon restart.
- The application may set the provision flag to 1 given certain condition, for instance, upon receiving a wireless command.
- When connected through USB or TCP/IP, the provision flag may be overwritten using Simplicity Commander:

```
commander nvm3 read -o ./temp/nvm3.s37
commander nvm3 set ./temp/nvm3.s37 --object 0x87228:01 --outfile ./temp/nvm3+.s37
commander flash ./temp/nvm3+.s37
```

A hands-on example of these provisioning flows will be provided in the following Sections.

Initial Setup

Using your PC for development, including:

- Studio with Matter Extension or SMG - *preferably the latest GSDK version.*
- Clone the matter repo

```
git clone https://github.com/SiliconLabs/matter.git
cd matter
git submodule update --init
```

In the matter repo directory:

```
source ./scripts/bootstrap.sh
source ./scripts/activate.sh
gn gen out/tools
ninja -C out/tools
cd provision/
```

Generating Matter Certificates (CD, PAA,PAI,DAC) - Provisioning Script

Reference and detailed explanation of the different processes that take place in the provisioning script are detailed in https://github.com/SiliconLabs/matter_extension/tree/v2.2.0/provision. The following is an example on how to generate certificates using the chip-cert tool. Start with generating the Certification Declaration as follows:

```
./out/tools/chip-cert gen-cd -K credentials/test/certification-declaration/Chip-Test-CD-Signing-Key.pem -C credentials/test/certification-declaration/Chip-Test-CD-Signing-Cert.pem -O credentials/test/certification-declaration/Chip-Test-CD-1049-8005.der -f 1 -V 0x1049 -p 0x8005 -c ZIG20142ZB330001-24 -l 0 -i 0 -n 257 -t 0 -o 0x1049 -r 0x8005
```

This chip-cert command uses the Chip-Test-CD-Signing-Key.pem and Chip-Test-CD-Signing-Cert.pem to sign the output CD which is Chip-Test-CD-1049-8005.der with **Vendor ID:** 0x1049 and **Product ID:** 0x8005.

The next step is to generate the Product Attestation Intermediate (PAI) and Device Attestation Certificate (DAC) using a test Product Attestation Authority (PAA) provided by the CSA and can be found in [~/matter/credentials/test/attestation/Chip-Test-PAA-NoVID-Cert.pem](#). This PAA will be the root certificate to sign the PAI which will then sign the DAC and we will obtain our Public Key Infrastructure Matter Certificate Chain:

```
./out/tools/chip-cert gen-att-cert -t i -l 3660 -c "Matter PAI" -V 0x1049 -P 0x8005 -C ./credentials/test/attestation/Chip-Test-PAA-NoVID-Cert.pem
-K ./credentials/test/attestation/Chip-Test-PAA-NoVID-Key.pem -o ./credentials/test/attestation/pai_cert.pem -O
./credentials/test/attestation/pai_key.pem
```

If you plan on using different PID for different Matter devices you can remove the -P 0x8005 from the PAI and this will provide more flexibility to generate DACs with different PIDs under the same PAI. You can use the following command to generate a DAC using the PAI Cert and the PAI key:

```
./out/tools/chip-cert gen-att-cert -t d -l 3660 -c "Matter DAC" -V 0x1049 -P 0x8005 -C ./credentials/test/attestation/pai_cert.pem -K
./credentials/test/attestation/pai_key.pem -o ./credentials/test/attestation/dac_cert.pem -O ./credentials/test/attestation/dac_key.pem
```

To verify the Certificate Chain you can also use chip-cert:

```
./out/tools/chip-cert validate-att-cert --dac credentials/test/attestation/dac_cert.pem --pai credentials/test/attestation/pai_cert.pem --paa
credentials/test/attestation/Chip-Test-PAA-NoVID-Cert.pem
```

If the chain is correctly verified, no errors should be output from this command.

Once you have finished generating you Certificates, you can proceed with installing the Provisioning Tool in order to flash the Matter Commissionable Data and the Device Attestation Data onto your device.

Provisioning Tool

Important: Please review the required installations in the [Provisioner Script](#) Section.

Required Installation

- Simplicity Commander:
 - Please install Simplicity Commander and add it to your environment variables.
 - example for Mac:

```
export
PATH=$PATH:/Applications/SimplicityStudio.app/Contents/Eclipse/developer/adapter_packs/commander/Commander.app/Contents/MacOS/
```
- SEGGER:
 - You can use the dlibjlinkarm.dylib in Simplicity Studio Eclipse/developer/adapter_packs/jlink/ or,
 - You can download SEGGER depending on your OS [here](#).
 - Once you have the dlibjlinkarm.dylib please try one of these options so the provisioning script can use jlink:

```
# Option A: Copy the library to your libraries directory.
$ cp libjlinkarm.dylib /usr/local/lib/
```



```
# Option B: Add SEGGER's J-Link directory to your dynamic libraries path.
$ export DYLD_LIBRARY_PATH=/Applications/SEGGER/JLink:$DYLD_LIBRARY_PATH
```

- SPAKE2 + generator: This tool is already part of the files in the cloned repo.
- Pylink
 - Please follow the [PyLink Installation Instructions](#).

Running the Provisioning Tool

Once you have generated the PAA, PAI and DAC and have installed the provisioning tool you can use it to write the Commissionable Data and the Device Attestation Data. As previously mentioned, there are two provisioning flows possible, following are the necessary steps to correctly provision your device.

Go to the ~/matter/provision/directory:

Generator Firmware

To choose different provisioning flows, the provisioning script has the argument option -gf to direct the script:

```
python3.7 ./provision.py -c config/silabs.json -ic ../credentials/test/attestation/pai_cert.pem -dc ../credentials/test/attestation/dac_cert.pem -dk
../credentials/test/attestation/dac_key.pem -cd ../credentials/test/certification-declaration/Chip-Test-CD-1049-8005.der -d 0xab2 -gf
images/efr32mg24.s37 -j 10.4.215.22
```

Provisioning Tool Output

```

> commander device info --ip 10.4.215.22
> mkdir -p ./temp

◆ Device Info:
· part: 'efr32mg24b220f1536im48'
· family: 'efr32mg24'
· version: '4'
· revision: 'A1'
· flash_addr: 0x08000000
· flash_size: 0x00180000

◆ Writing firmware
> commander flash images/efr32mg24.s37 --ip 10.4.215.22
> commander device reset --ip 10.4.215.22

◆ Preparing credentials

◆ SPAKE2+ Verifier
· pass: 62034001
· salt: U1BBS0UyUCBLZXkgU2FsdA==
· iter: 1500
▪ EBoBESpGG+XUOhcw8DXk4+4C7jQ8KQI5ZtrA7BEIzZ4EDK2l6QnVVlnxr1VfGla2ht3VIIj1gs/ZPUZ57GZuVcf2xOKN0gjM0GBj0mzvzoYAV0Dxo8RcL3Dxo9

> cp ..//credentials/test/certification-declaration/Chip-Test-CD-1049-8005.der ./temp/cd.der
> cp ..//credentials/test/attestation/pai_cert.pem ./temp/pai_cert.pem
> openssl x509 -outform der -in ./temp/pai_cert.pem -out ./temp/pai_cert.der
> cp ..//credentials/test/attestation/dac_cert.pem ./temp/dac_cert.pem
> openssl x509 -outform der -in ./temp/dac_cert.pem -out ./temp/dac_cert.der
> cp ..//credentials/test/attestation/dac_key.pem ./temp/dac_key.pem
> openssl ec -inform pem -in ./temp/dac_key.pem -outform der -out ./temp/dac_key.der

◆ Connecting to device
▪ Open TCP connection 10.4.215.22:19020 to efr32mg24b220f1536im48
△ Init send(12)
▲ Init response(17)
Init:
· addr:0817e000, page:8192

◆ Credentials: Import

△ KEY send(135)
▲ KEY response(22)
Import(KEY):
· key:0, off:0x0, size:121

◆ Credentials: Write

△ DAC send(491)
▲ DAC response(22)
Import(DAC):
· key:0, off:0x0, size:477
△ PAI send(456)
▲ PAI response(22)
Import(PAI):
· key:0, off:0x0, size:442
△ CD send(257)
▲ CD response(22)
Import(CD):
· key:0, off:0x0, size:243

◆ Credentials: silabs_creds.h (legacy)

◆ Write Factory Data

```

Setup:

- passcode: 0x3b29051
- discriminator: 0xab2
- uid: 4974be050220f9d10f899828fba42562
- payload: 48822800444056a3206507

Note: If you have connected more than one device, provide the -j <j-link no>.

Note: Invalid chip-cert path (--cert_tool): './out/tools/chip-cert' (The provisioning script uses the cert-tool. If the activate.sh script has not been successfully run, this error will occur.)

Building a Sample Application - SMG

To build a lighting app using SMG, you can run these commands for an MG24 or an MG12. This example is using a BRD4162A. Go to the matter directory and execute the following command (in the matter directory):

```
cd ..  
./scripts/examples/gn_silabs_example.sh ./examples/lighting-app/silabs/ ./out/lighting-app/ BRD4187C  
chip_build_platform_attestation_credentials_provider = true
```

`chip_build_platform_attestation_credentials_provider = true` instructs the software to use the credentials that have been provided by the provisioning tool in the last page of flash.

Your application will be in /out/lighting-app/BRD4187C.

Building a Sample Application - Studio - Matter Extension

Create a New Lighting Over Thread Project

New Project Wizard

Example Project Selection
Select the project template to open in Simplicity IDE.

Target, SDK Examples Configuration

Filter on keywords

Example Projects

Solution Examples

Wireless Technology

- Bluetooth (4)
- Bluetooth Mesh (0)
- Connect (0)
- Matter (15)
- RAIL (0)
- Sidewalk (0)
- Thread (16)
- Zigbee (6)

Device Type

- Host (1)
- NCP (0)
- RCP (0)
- SoC (14)

Ecosystem

15 resources found

Matter - SoC OnOff Plug over Thread
This project builds a Matter OnOffPlug app that can be developed inside Simplicity Studio

Matter - SoC Lock over Thread with Bootloader Solution
This project builds a Matter Lock and Matter Bootloader that can be developed inside Simplicity Studio

Matter - SoC Lock over Thread
This project builds a Matter Lock that can be developed inside Simplicity Studio

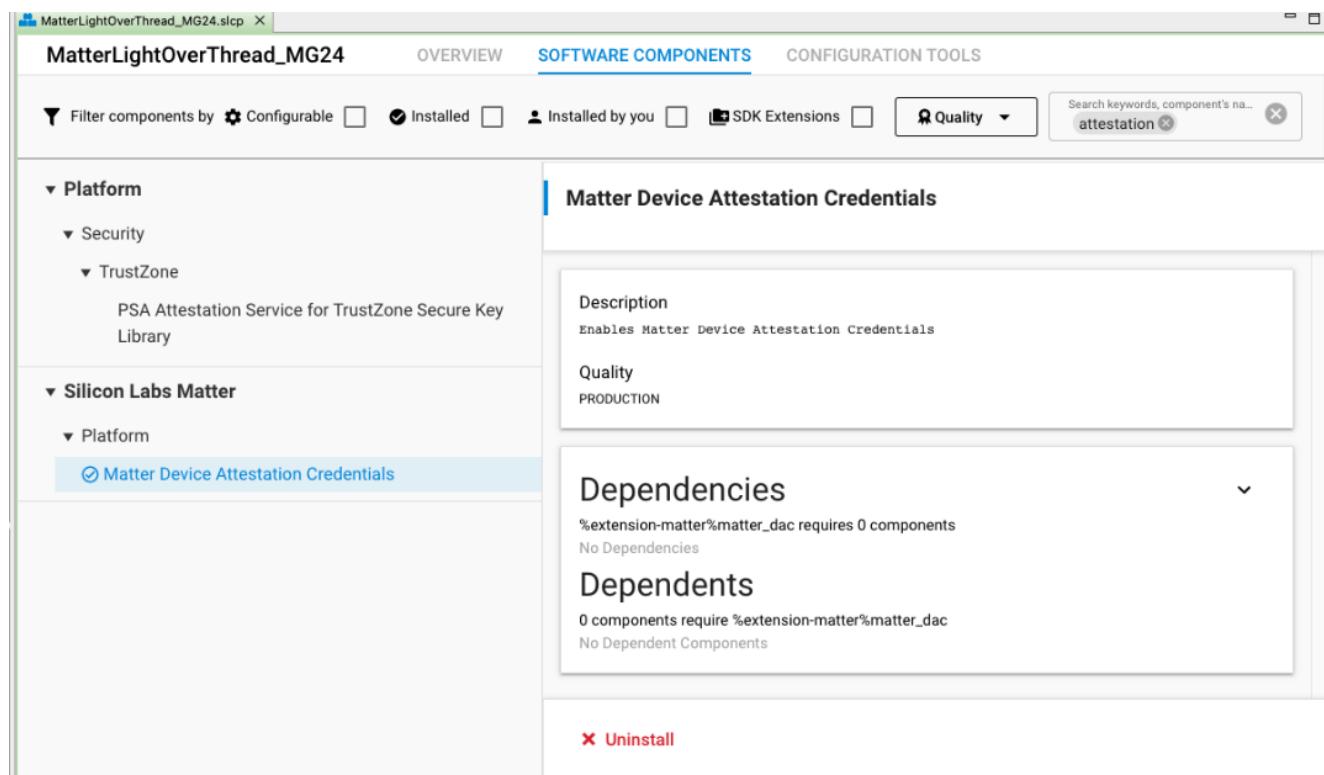
Matter - SoC Lighting over Thread with Bootloader Solution
This project builds a Matter Light and Matter Bootloader that can be developed inside Simplicity Studio

Matter - SoC Lighting over Thread
This project builds a Matter Lighting app that can be developed inside Simplicity Studio

Matter - SoC Light Switch over Thread with Bootloader Solut...
This project builds a Matter Light Switch and Matter Bootloader that can be developed inside Simplicity Studio

CANCEL **BACK** **NEXT** **FINISH**

Install Matter Device Attestation Credentials Component



Matter Device Attestation Credentials

Description
Enables Matter Device Attestation Credentials

Quality
PRODUCTION

Dependencies
%extension-matter%matter_dac requires 0 components
No Dependencies

Dependents
0 components require %extension-matter%matter_dac
No Dependent Components

X Uninstall

This component is meant for the firmware to refer to the credentials injected by the provisioning tool.

Once this is completed, you can build your image and flash the <image>.s37 using Simplicity Studio.

Store Commissionable Data (NVM3), Attestation Data CD,PAI, DAC (Main Flash)

```
python3.7 ./provision.py -c config/silabs.json -r -cn "Silabs Device" -ic ./temp/pai_cert.pem -cd ./temp/cd.der -pf \<prod-fw-path\> -j \<jlink-no or ip-address\>
```

Flashing Certificates Provisioning Tool Output

```

▸ commander device info --serialno 440292679
▸ mkdir -p ./temp

◆ Device Info:
· part: 'efr32mg12p332f1024gl125'
· family: 'efr32mg12'
· version: '25'
· revision: 'A2'
· flash_addr: 0x00000000
· flash_size: 0x00100000

◆ Prepare
▸ openssl x509 -outform der -in ./temp/pai_cert.pem -out ./temp/pai_cert.der

◆ Loading Generator Firmware
▸ commander flash ./images/efr32mg12.s37 --serialno 440292679

▪ Open SERIAL connection 440292679 to efr32mg12p332f1024gl125
Init:
· addr:000ff800, page:2048

◆ Credentials: CSR
CSR, key:1, len:391
-----BEGIN CERTIFICATE REQUEST-----
MIHqMIGOAgEAMCwkJAoBgNVBAMMIVNpbGFicyBEZXZpY2UgTXZpZDoxMDQ5IE1w
aWQ6ODAwNTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IBBiLc2VN15CDtv1EVGQ
iv5jSchjW2f8cx56rc4tq6/LY9COnZoAmj2fNrHonD60GO78bXJ+ka42/Uaxyecq
/wWgADAMBggqhkjOPQQDAgUAA0kAMEYCIQCrfnIrRWirEy0dP9ybK78Ty0G3mCf
UmYFbvFb4LpcTgIhALd+70nQO/zKdipBmHscHZ9PN7UyjcflMza5L7WzzC4I
-----END CERTIFICATE REQUEST-----

◆ Credentials: DAC
◆ Sign
· serial number: 41449
▸ openssl x509 -sha256 -req -days 18250 -extensions v3_ica -extfile ./csa_openssl.cnf -set_serial 41449 -CA ./temp/pai_cert.der -CAkey
./temp/pai_key.der -CAform DER -CAkeyform DER -in ./temp/csr.pem -outform der -out ./temp/dac_cert.der

◆ Credentials: Write
Import(DAC):
· key:1, off:0x0, size:453
Import(PAI):
· key:1, off:0x200, size:460
Import(CD):
· key:1, off:0x400, size:243

◆ Credentials: silabs_creds.h (legacy)

◆ Write Factory Data

Setup:
· passcode: 0x5abcdef
· discriminator: 0xe9a
· uid: b3d79923b02237dd75903fdf553c887d
· payload: 488228004440d3df9b570b

◆ Write app
▸ commander flash ../out/lighting-app/BRD4162A/matter-silabs-lighting-example.hex --serialno 440292679

```

You can see in the provisioning tool output the Payload Setup

Setup:

- passcode: 0x5abcdef
- discriminator: 0xe9a
- uid: b3d79923b02237dd75903fdf553c887d
- payload: 488228004440d3df9b570b

Matter Commissioning - Using MatterHub (RaspberryPi)

Start a Thread Network

```
mattertool startThread
```

This will provide you with the Thread Network Dataset.

```
mattertool pairing ble-thread 1111  
hex:0e080000000000000000000000000000300000b35060004001fffe00208dead00beef00cafe0708fddead00beef000005106dab1ff61b8a77e5795876fd  
0x5ABCDEF 0xd02 --paa-trust-store-path /home/ubuntu/temp
```

Where:

1111 is the Node Id that will be assigned to the MAD being commissioned

hex:0e080000000000000000000000000000300000b35060004001fffe00208dead00beef00cafe0708fddead00beef000005106dab1ff61b8a77e5795876fd
is the dataset of the thread network created

0x5ABCDEF is the passcode

0xd02 is the discriminator

/home/ubuntu/temp/ is the folder where the corresponding PAA is located in the Commissioner.

Matter Commissioning

Commissioning

Overview

The commissioning process supports two potential starting points:

1. The device is already on the network
2. The device needs network credentials for Wi-Fi or Thread (requires Bluetooth LE (BLE) support)

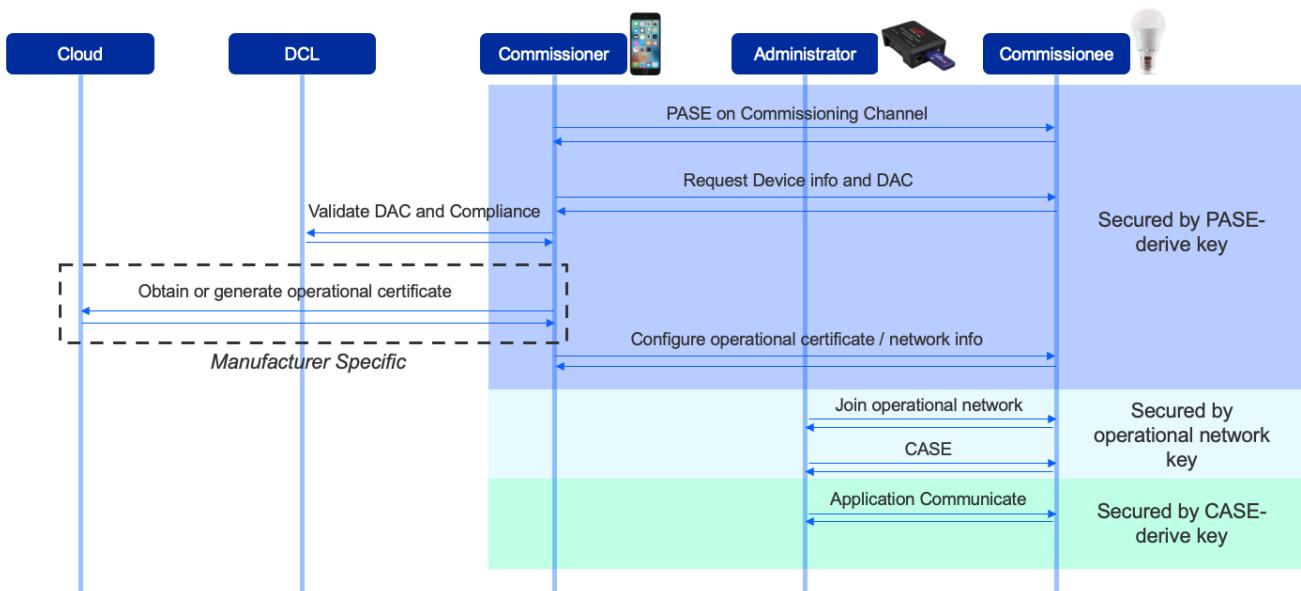
The current Matter revision supports Ethernet, Wi-Fi, and Thread devices.

- Ethernet devices get into the operational network when their Ethernet cable is connected. Therefore the devices are normally already on the network before commissioning.
- Wi-Fi and Thread devices must have credentials configured before the devices can be joined into the operational network. This is normally done over BLE.

This page focuses on Wi-Fi and Thread. The first step for these devices is to enter commissioning mode, following one of two scenarios:

Scenario Name	Description
Standard	Device automatically goes into the commissioning mode on power-up. Beneficial for limited UI devices (such as light bulbs)
User-Directed	Device only enters commissioning mode when initiated by the user. Helpful for devices that have user interfaces or for which commissioning should not be initiated without a user present.

The following figure provides an overview of the commissioning process and the actions each role performs.



Example Commissioning Flow

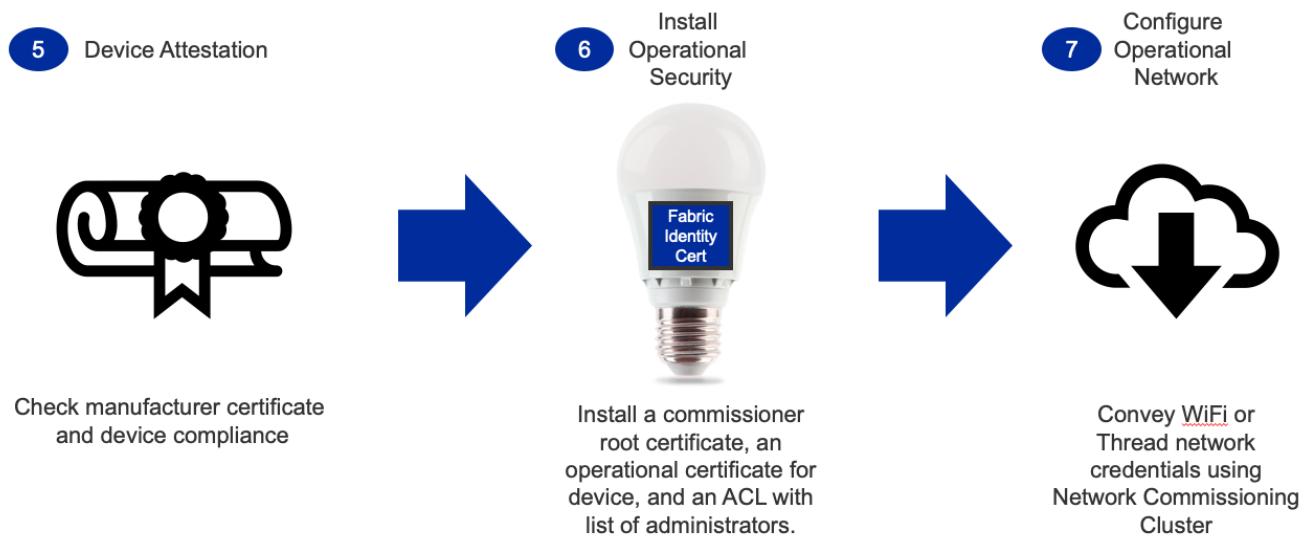


In step 1, the Matter device must enter commissioning mode in one of the two scenarios described above.

Usually, a mobile phone serves as the administrator. Step 2 is to use the mobile phone to scan the QR code of the Matter device. The QR code is used as a passcode to set up a secured BLE connection.

Step 3 is to set up the BLE beaconing and connection between the mobile phone and the Matter device, so that the commissioning information can be exchanged through the BLE connection channel.

As the connection should be secure, step 4 is to secure the connection in a process known as password-authenticated session establishment (**PASE**). The passcode derived from the QR code is used as an input for this process. The output is the security key used by the connection.

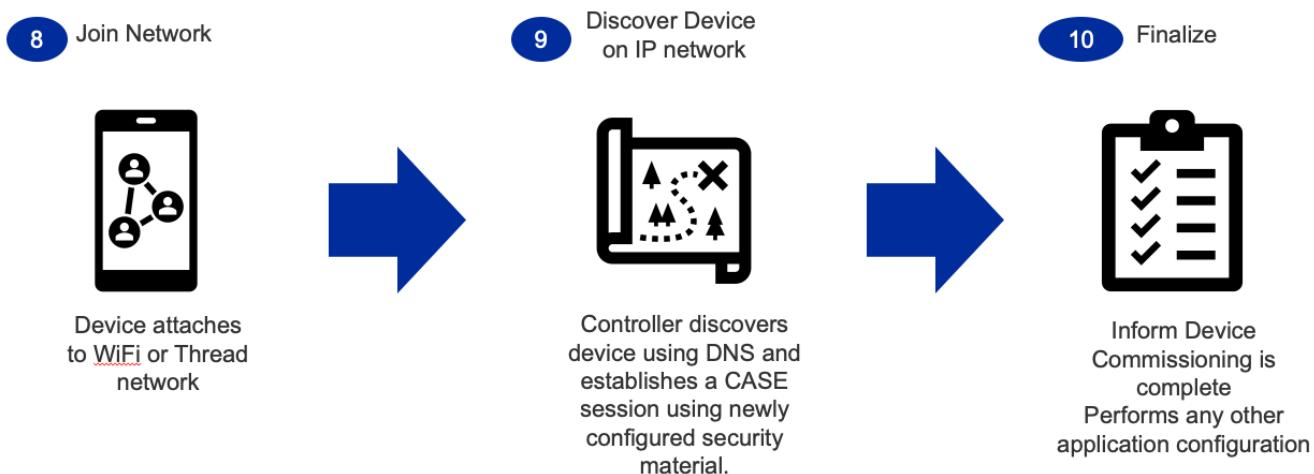


After the secured connection is established, step 5 is to verify the Matter device's manufacturer certificate and compliance status. Each Matter device must have a device certificate programmed before it is shipped. The mobile phone, acting as administrator, reads the device certificate through the commissioning channel, then communicates with a remote database to validate the certificate and the compliance status of the device. The remote database is called the Distributed Compliance Ledger (**DCL**).

Step 6 is to install the operational certificate for the device. The administrator either obtains the certificate from the remote server or generates the certificate locally and then transfers the certificate to the device. The administrator also

configures the Access Control List (**ACL**) with the list of administrators.

After operational security is configured, step 7 is to configure the operational network for the device. For Wi-Fi devices, the SSID and the password are configured. For Thread devices, the PAN ID, network key, and other parameters are configured.



In step 8, the device starts to join the operational network with the configured parameters.

Once the device is attached to the network (step 9), it can be discovered through Service Registration Protocol (**SRP**). To control that device, you must establish a secured connection through the Certification Authorized Session Establishment (**CASE**) process.

After the CASE session is established, the Matter device is commissioned successfully and can communicate with other devices in the Matter network (step 10).

Matter Intermittently Connected Devices (ICD)

Matter Intermittently Connected Devices (ICD)

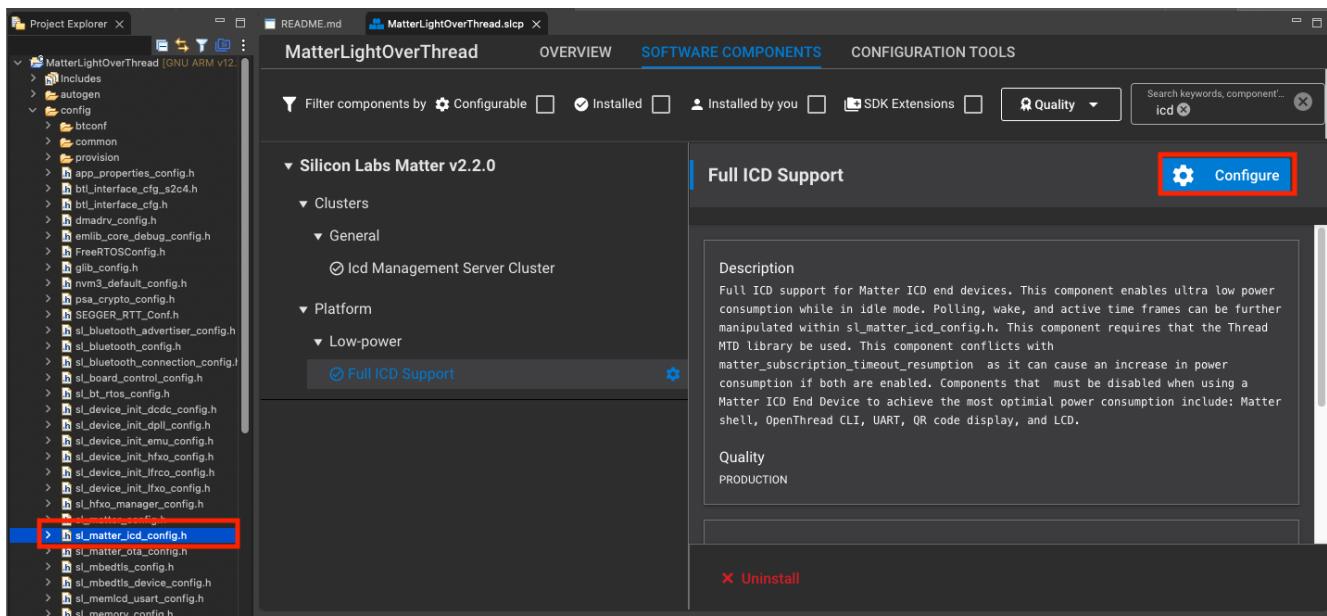
Matter introduces the concept of Intermittently Connected Devices (ICD) in the SDK and in the specification. An Intermittently Connected Device is the Matter representation of a device that is not always reachable. This covers battery-powered devices that disable their underlying hardware when in a low-power mode or devices that can be disconnected from the network, like a phone app.

This page focuses on features designed to improve the performance and reliability of battery-powered devices. Matter ICD functionality can be enabled with the `matter_icd` component.

Configuration

To change default values corresponding to Matter ICD examples, modify them in either:

1. `config/sl_matter_icd_config.h`
2. ICD component configurator



ICD Device Types

Matter introduces two types of ICDs.

- Short Idle Time ICDs
- Long Idle Time ICDs

Short Idle Time ICDs

Short Idle Time ICDs are battery powered devices that can always be reached by clients. This means that their polling intervals are small enough to guarantee that a message sent from a client will be able to reach the ICD without any

synchronization. A door lock, for example, is typically a short idle time ICD because it needs to be able to receive commands from clients at any given time. These devices are usually not the initiators in the communication flow.

Long Idle ICDs

Long Idle Time ICDs are battery powered devices that require synchronization between the client and the ICD for communication to succeed. A sensor device is an example of a device that are typically long idle time ICDs.

Long Idle Time ICDs are provisioned with the Matter 1.2 release.

ICD Management Cluster

The ICD Management Cluster enables configuration of the ICD's behavior. It is required for an ICD to have this cluster enabled on endpoint 0.

The ICDM Cluster exposes three configuration attributes that enable to configure an ICD.

Attribute	Type	Constraints	Description
IdleModelInterval	uint32	1 to 64800	Maximum interval in seconds or milliseconds the server can stay in idle mode
ActiveModelInterval	uint32	min 300	minimum interval in milliseconds the server will stay in active mode
ActiveModeThreshold	uint32	min 300	minimum amount of time in milliseconds the server typically will stay active after network activity when in active mode

These configurations can be changed by modifying the values within `sl_matter_icd_config.h` or within the settings of the `matter_icd` component.

```
#define SL_IDLE_MODE_INTERVAL = 600 // 10min Idle Mode Interval
#define SL_ACTIVE_MODE_INTERVAL = 1000 // 1s Active Mode Interval
#define SL_ACTIVE_MODE_THRESHOLD = 500 // 500ms Active Mode Threshold
```

Subscription Maximum Interval

The subscription mechanism is used by ecosystems and controllers to receive attribute change updates and liveness checks. The maximum interval of a subscription request is what defines the frequency at which a device will send a liveness check if there are no attribute changes.

Within the subscription request / response model, a device has the opportunity to decide the maximum interval at which it will send its liveness check (Empty Report Update). The device can set a maximum interval within this range if and only if it is an ICD:

```
MinIntervalRequested ≤ MaxInterval ≤ MAX(IdleModelInterval, MaxIntervalRequested)
```

The following table shows the subscribe response fields.

Action Field	Type	Description
SubscriptionId	uint32	identifies the subscription
MaxInterval	uint16	the final maximum interval for the subscription in seconds

Maximum Interval Negotiation

The Matter SDK provides a default implementation that allows an ICD to negotiate its MaxInterval. The goal of the algorithm is to set the MaxInterval to the IdleModelInterval.

```
#if CHIP_CONFIG_ENABLE_ICD_SERVER

// Default behavior for ICDs where the wanted MaxInterval for a subscription is the IdleModelInterval
// defined in the ICD Management Cluster.
// Behavior can be changed with the OnSubscriptionRequested function defined in the application callbacks
```

```
// Default Behavior Steps :// If MinInterval > IdleModeInterval, try to set the MaxInterval to the first interval of IdleModeIntervals above the//  
MinInterval:// If the next interval is greater than the MaxIntervalCeiling, use the MaxIntervalCeiling:// Otherwise, use IdleModeInterval as  
MaxInterval// GetPublisherSelectedIntervalLimit() returns the IdleModeInterval if the device is an ICD  
uint32_t decidedMaxInterval = GetPublisherSelectedIntervalLimit();// Check if the PublisherSelectedIntervalLimit is 0. If so, set  
decidedMaxInterval to MaxIntervalCeilingif(decidedMaxInterval == 0){  
    decidedMaxInterval = mMaxInterval;}// If requestedMinInterval is greater than the IdleTimeInterval, select next active up time as max  
intervalif(mMinIntervalFloorSeconds > decidedMaxInterval){  
    uint16_t ratio = mMinIntervalFloorSeconds / static_cast<uint16_t>(decidedMaxInterval);if(mMinIntervalFloorSeconds % decidedMaxInterval){  
        ratio++;}  
  
    decidedMaxInterval *= ratio;// Verify that decidedMaxInterval is an acceptable value (overflow)if(decidedMaxInterval >  
System::Clock::Seconds16::max().count()){  
    decidedMaxInterval = System::Clock::Seconds16::max().count();}// Verify that the decidedMaxInterval respects  
MAX(GetPublisherSelectedIntervalLimit(), MaxIntervalCeiling)  
uint16_t maximumMaxInterval = std::max(GetPublisherSelectedIntervalLimit(), mMaxInterval);if(decidedMaxInterval > maximumMaxInterval){  
    decidedMaxInterval = maximumMaxInterval;}// Set max interval of the subscription  
mMaxInterval = static_cast<uint16_t>(decidedMaxInterval);  
  
#endif // CHIP_CONFIG_ENABLE_ICD_SERVER
```

If the default implementation does fit within the use-case, an implementation can override the default implementation. The first step is to implement the `ApplicationCallback` class from the `ReadHandler.h` header.

```
/*  
 * A callback used to interact with the application.  
 */  
class ApplicationCallback  
{  
public:  
    virtual ~ApplicationCallback() = default;  
    /*  
     * Called right after a SubscribeRequest has been parsed and processed. This notifies an interested application  
     * of a subscription that is about to be established. It also provides an avenue for altering the parameters of the  
     * subscription (specifically, the min/max negotiated intervals) or even outright rejecting the subscription for  
     * application-specific reasons.  
     *  
     * TODO: Need a new IM status code to convey application-rejected subscribes. Currently, a Failure IM status code is sent  
     * back to the subscriber, which isn't sufficient.  
     *  
     * To reject the subscription, a CHIP_ERROR code that is not equivalent to CHIP_NO_ERROR should be returned.  
     *  
     * More information about the set of paths associated with this subscription can be retrieved by calling the appropriate  
     * Get* methods below.  
     *  
     * aReadHandler: Reference to the ReadHandler associated with the subscription.  
     * aSecureSession: A reference to the underlying secure session associated with the subscription.  
     */  
    virtual CHIP_ERROR OnSubscriptionRequested(ReadHandler & aReadHandler, Transport::SecureSession & aSecureSession)  
    {  
        return CHIP_NO_ERROR;  
    }  
    /*  
     * Called after a subscription has been fully established.  
     */  
    virtual void OnSubscriptionEstablished(ReadHandler & aReadHandler){};  
    /*  
     * Called right before a subscription is about to get terminated. This is only called on subscriptions that were terminated  
     * after they had been fully established (and therefore had called OnSubscriptionEstablished).  
     * OnSubscriptionTermination().  
     */  
    virtual void OnSubscriptionTerminated(ReadHandler & aReadHandler){};  
};
```

The second step is registering the callback object to the Interaction Model Engine.

```
// Register ICD subscription callback to match subscription max intervals to its idle time interval  
chip::app::InteractionModelEngine::GetInstance()->RegisterReadHandlerAppCallback(&mICDSubscriptionHandler);
```

Persistent Subscriptions

Persistent subscriptions were added to Matter as a means to ensure that an ICD can re-establish its subscription and by extension its secure session to a subscriber in the event of a power cycle. When a device accepts a subscription request, it will persist the subscription. When the device reboots, it will try to re-establish its subscription with the subscriber. If the subscription is torn down during normal operations or if the re-establishment fails, the subscription will be deleted.

Persistent subscriptions are enabled by default on all Silicon Labs sample applications.

Subscription Timeout Resumption

Matter also provides a retry mechanism for devices to try to re-establish a lost subscription with a client. This functionality is provided by the component `matter_subscription_timeout_resumption`. This feature should not be used on an ICD since it can significantly reduce battery life.

This feature is enabled by default on all examples with the exception of the door-lock and light-switch example.

Subscription Synchronization

To avoid forcing an ICD to become active multiple times, the Matter SDK allows an ICD to synchronize its subscription reporting and send all the reports at the same time. The mechanism synchronizes the maximum interval of all subscriptions to only require the ICD to become active once. This functionality is provided by component `matter_subscription_synchronization`.

This feature is enabled by default on the door-lock sample app and the light-switch sample application.

Matter OpenThread ICD Device

Matter Intermittently Connected Devices over OpenThread

This page explains how Matter OpenThread Intermittently Connected Devices (ICDs) work and how to configure an ICD example.

Overview

Matter provides an Intermittently Connected Device (ICD) operating mode to extend the battery life of power-limited devices. The Matter ICD manager leverages subscription report synchronization and OpenThread functionalities to allow devices to sleep for set periods without disrupting their Matter sessions.

Currently, in Matter v1.2, only ICD with Short Idle Time (SIT) is supported. ICD SIT are devices that SHOULD be configured with a Slow Polling Interval shorter than or equal to 15 seconds. For example, in a typical scenario for door locks and window coverings, commands need to be sent to the ICD with a use-case imposed latency requirement. Typically, devices that are Short Idle Time ICDs are not initiators in the communication flow.

Operating Modes

ICDs have two operating modes: Idle and Active. An ICD alternates normally between the Idle mode and Active mode based on the `IdleModelInterval` and `ActiveModelInterval` respectively.

When the device is in *Active Mode*, the ICD is set into a fast-polling interval for maximum responsiveness. The `CHIP_DEVICE_CONFIG_ICD_FAST_POLL_INTERVAL` parameter communicates the maximum sleep interval of a node in active mode.

Any of the following device states will start or keep the ICD in *Active Mode*:

- A commissioning window is open
- An exchange context is awaiting a response or ack
- The fail-safe is armed

Any of the following events can trigger the start of the *Active Mode* interval or extend it by one `ActiveModeThreshold`:

- A message needs to be sent
- A message was received
- An implemented user action occurred

Once the active mode is triggered, the ICD stays in this mode for a minimum duration of `ActiveModeInterval`. When the active interval has elapsed and none of the aforementioned states are active, the device will switch its operating mode to the *Idle Mode*.

In *Idle mode*, the ICD will poll its associated router at its slow-polling interval to see if another device has tried to communicate with it while it was sleeping. If no event occurs, the ICD stays in its idle mode for the entirety of the `IdleModelInterval`. The `CHIP_DEVICE_CONFIG_ICD_SLOW_POLL_INTERVAL` parameter communicates the slow-polling interval and therefore the maximum sleep interval of the node in idle mode. This parameter affects both the minimum power consumption and maximum latency.

Thread Communication

In order to receive messages that were sent while the ICD was sleeping, the ICD relies on its associated Thread router which buffers any incoming messages. The Thread router will send all buffered messages to the ICD when it polls the router at the end of its slow-polling interval.

Configuration

Matter exposes some defines to configure the polling intervals of the OpenThread stack in both Idle and Active modes.

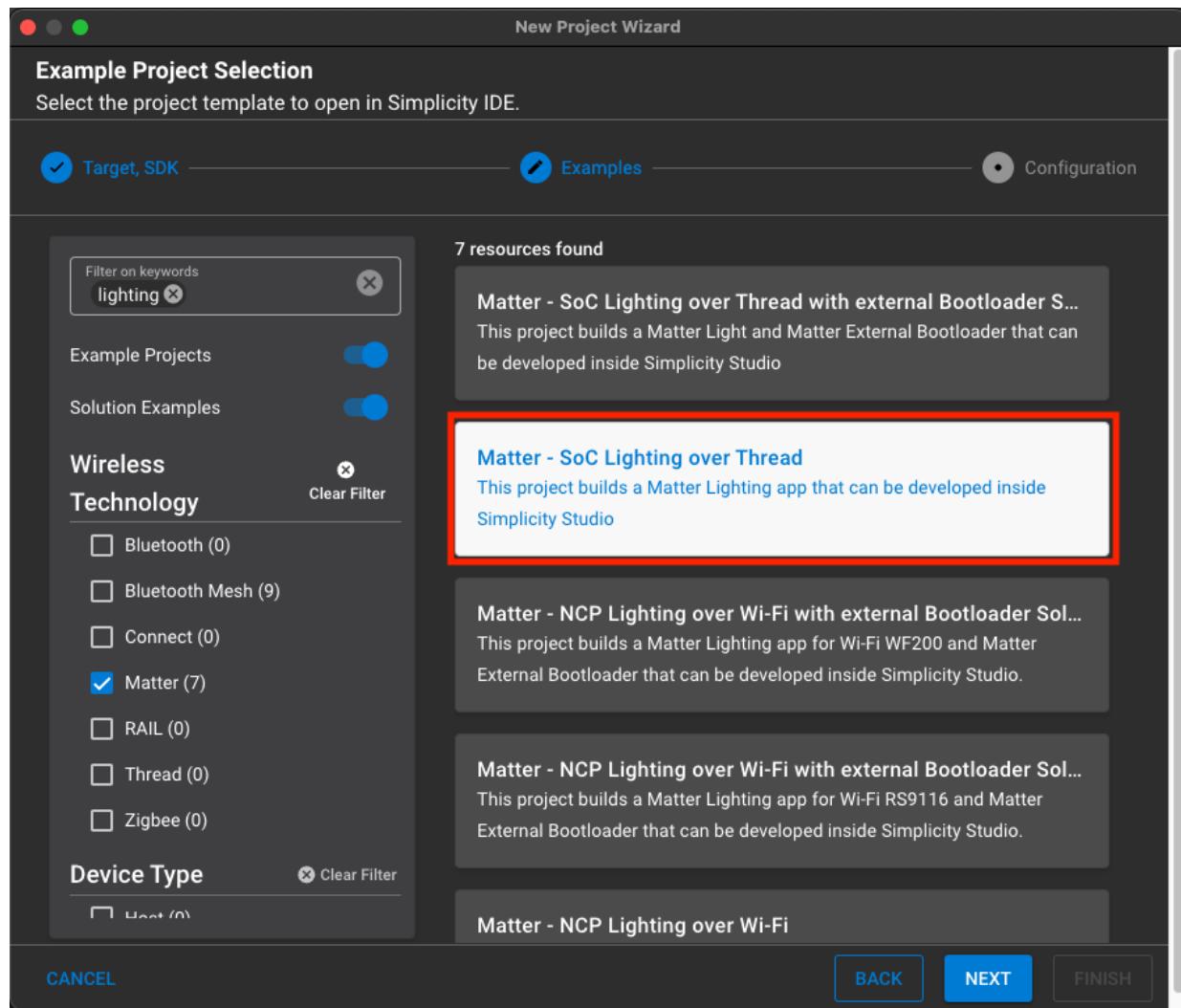
Parameter Name	Define	Description	Default Value	Maximum Allowed Value
SlowPollInterval	CHIP_DEVICE_CONFIG_ICD_SLOW_POLL_INTERVAL (SL_OT_IDLE_INTERVAL)	Interval, in milliseconds, at which the thread radio will poll its network in idle mode.	15000 ms	<= IdleModelInterval
FastPollInterval	CHIP_DEVICE_CONFIG_ICD_FAST_POLL_INTERVAL (SL_OT_ACTIVE_INTERVAL)	Interval, in milliseconds, at which the thread radio will poll its network in active mode.	200 ms	< ActiveModelInterval

For Matter configuration, see the [Matter ICD](#) documentation.

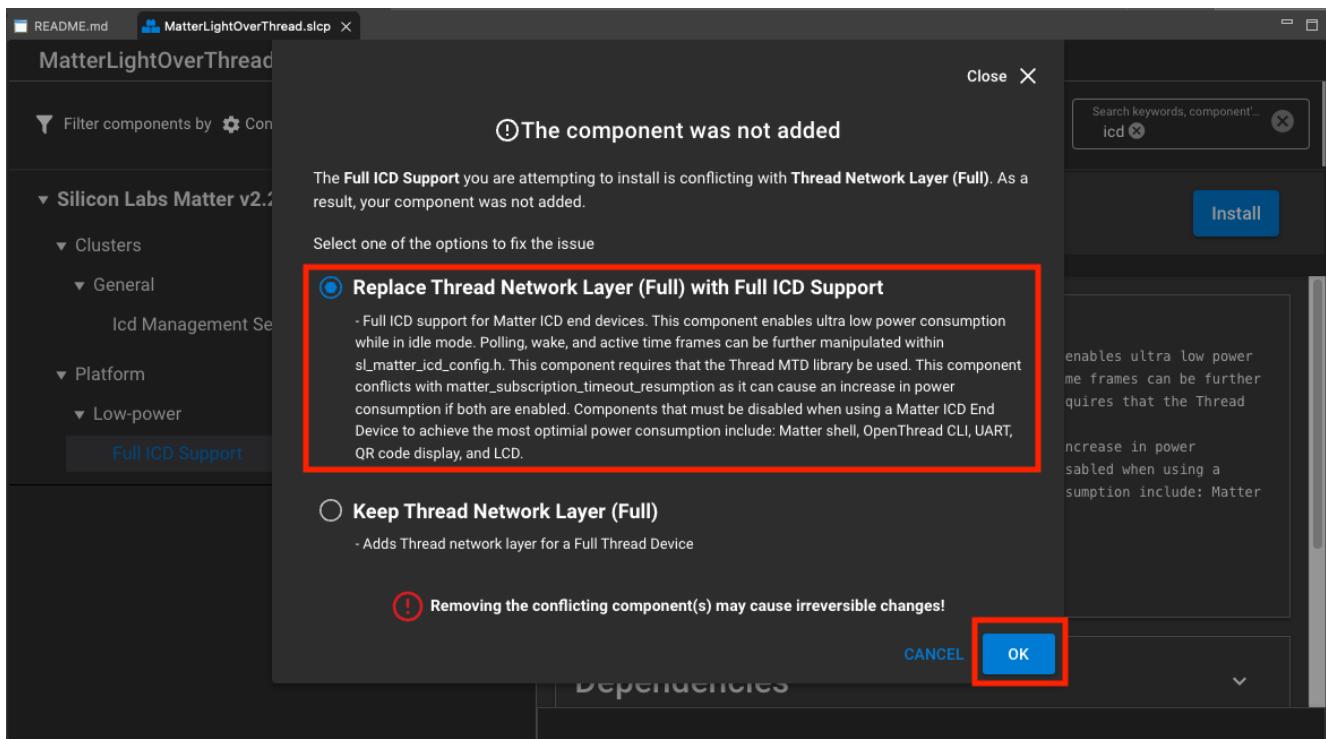
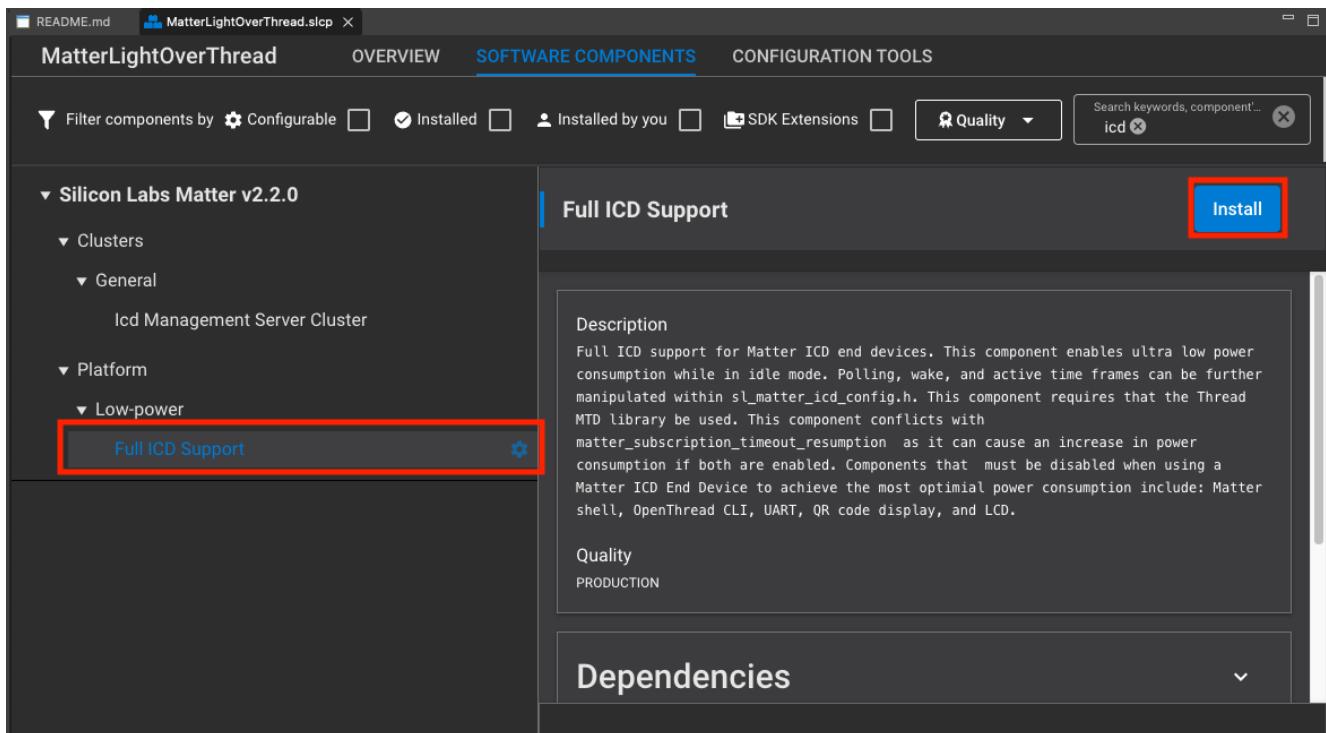
Building

Enabling/Building

1. To begin creating an OpenThread ICD example, create a generic Matter over Thread example via the [New Project Wizard](#). Lighting example will be used for demonstration purposes. Lock and Light-Switch applications come out-of-box with ICD enabled.



1. Once the project is generated, navigate to the software components section and install the Matter ICD component. Replace all subsequent conflicting components via the ensuing pop-up options (See below). This will install the necessary Thread Network Layer (MTD) component and ICD source code. This will also remove the conflicting Thread Network Layer (FTD) component.



1. ICD functionality should be installed and ready to build. Build the project as you would a normal example and flash the resulting binary to your specified end device. You should be able to commission the device the same way as non-ICD examples using the QR code URL (generated within the RTT logs at startup/BTN0 press).

Minimal Power Consumption

The Lower Power Mode component is optional for low-power builds with the component `matter_platform_low_power`.

The Lower Power Mode component will disable:

- Matter Shell
- OpenThread CLI
- LCD and QR Code

Matter Serial Port Communication (Matter Shell)

Serial Port Communication on the Silicon Labs Platform

The matter-shell exposes the configuration and the management APIs via the matter command line interface (matter CLI). This interface can be used to change the state of the device.

Hardware Requirements

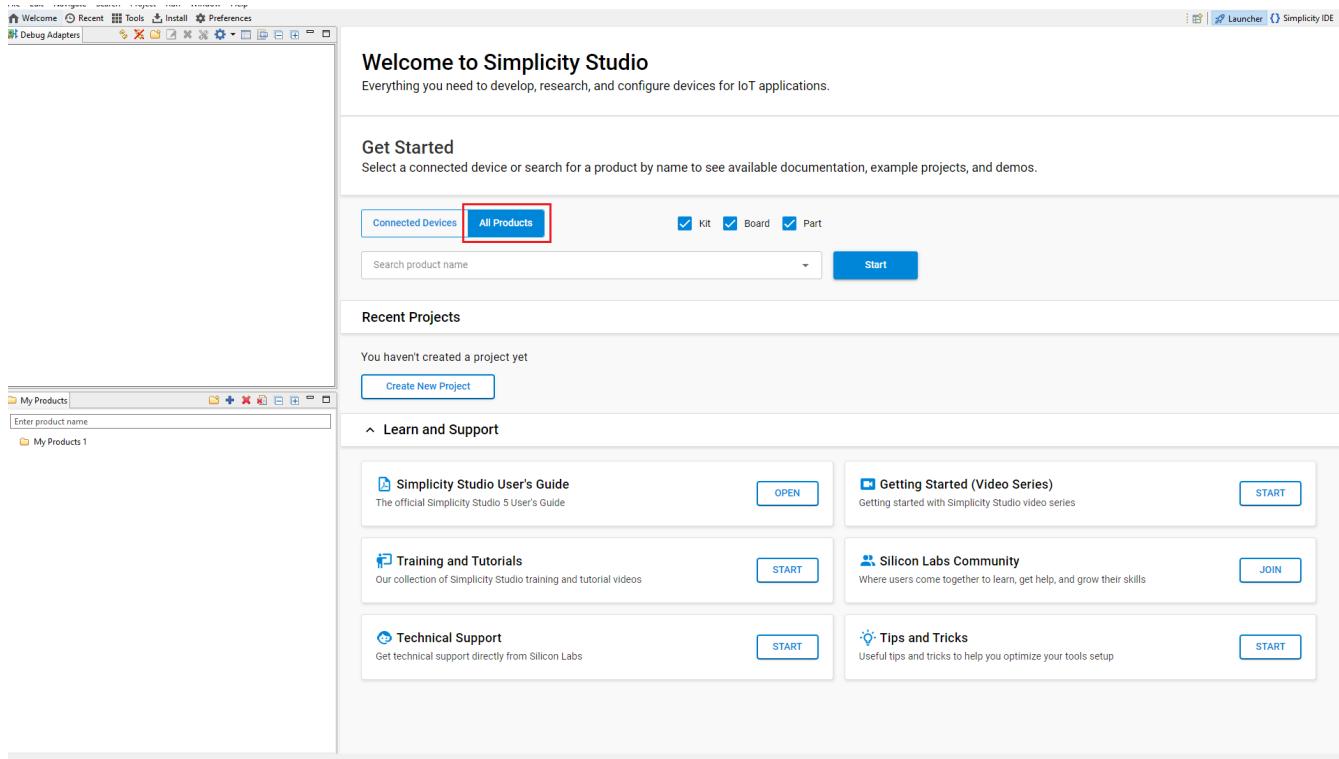
- To run matter shell on the Silicon Labs Platform, refer to the [Hardware Requirements](#).

Software Requirements

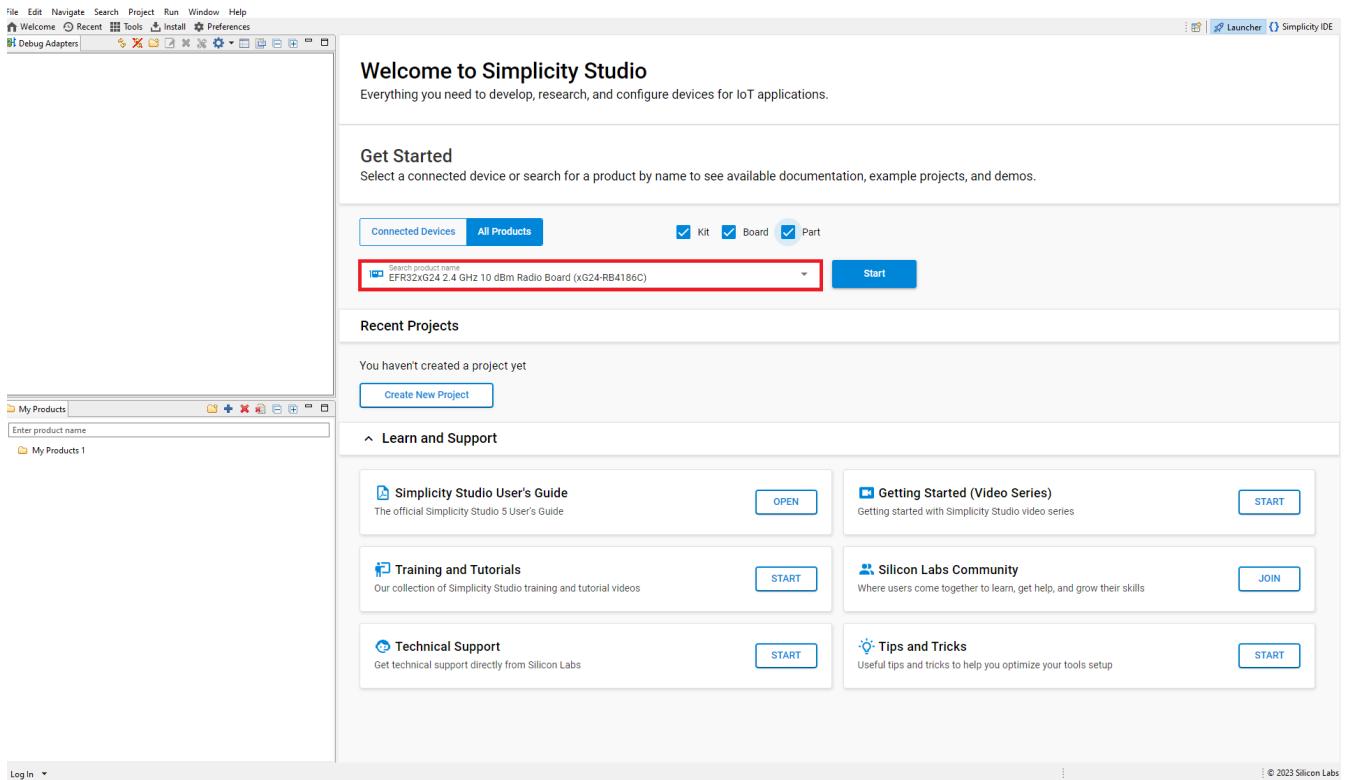
- To run matter shell on the Silicon Labs Platform, refer to the [Software Requirements](#).

Execute Matter Shell on Silicon Labs Platform

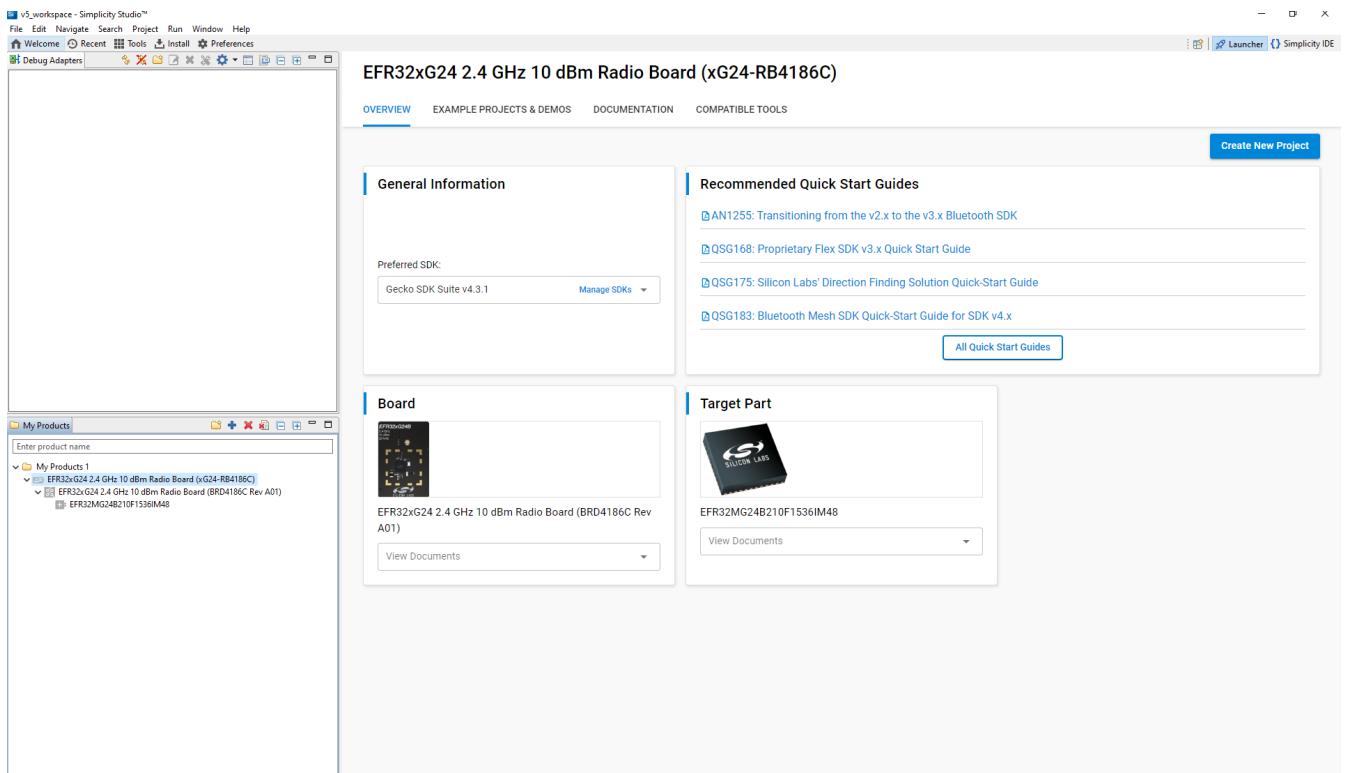
- Download and Install Simplicity Studio.
- To install the software packages for Simplicity Studio, refer [Software Package Installation](#).
- Log in to Simplicity Studio and connect the EFR32MG2x or SiWx917 SOC board to the computer.
- Go to the All Products section.



- Type and Select the radio board from the displayed list and select Start.



6. The Launcher page will display the selected radio board's details.



7. Verify the following in the General Information section:

- o The Debug Mode is Onboard Device (MCU).
- o The Preferred SDK is the version you selected earlier.

File Edit Source Refactor Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences

Debug Adapters



> EFR32xG21 2.4 GHz 20 dBm RB (ID:440192814)

EFR32xG21 2.4 GHz 20 dBm RE

[OVERVIEW](#)[EXAMPLE PROJECTS & DEMOS](#)[DOCUMENTATION](#)

General Information

Connected Via: J-Link Silicon Labs

[Configure](#)Debug Mode: **Onboard Device (MCU)** Adapter FW: **1v4p6b1171** [Latest](#)Secure FW: **1.2.1** [Update to 1.2.11](#) | [Changelog](#)

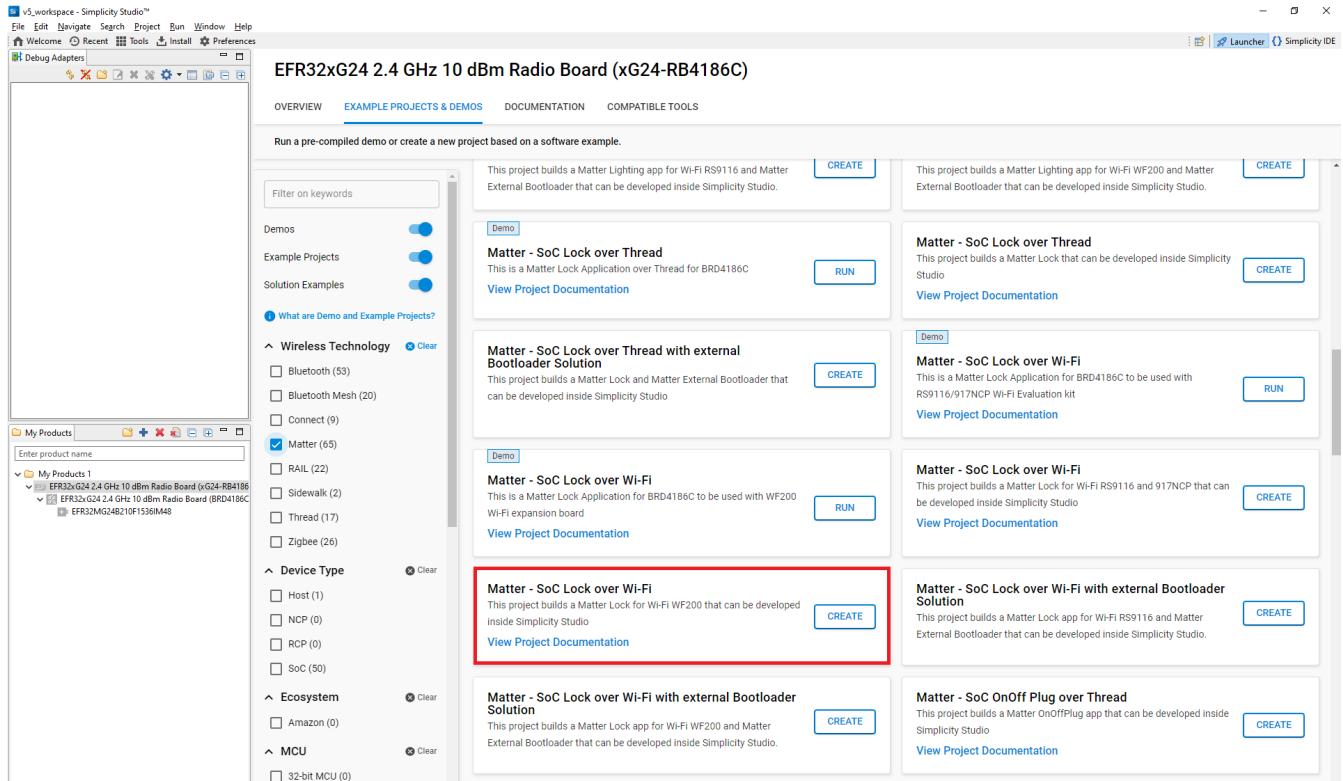
Preferred SDK:

Gecko SDK Suite v4.3.1 [Manage SDKs](#) ▾

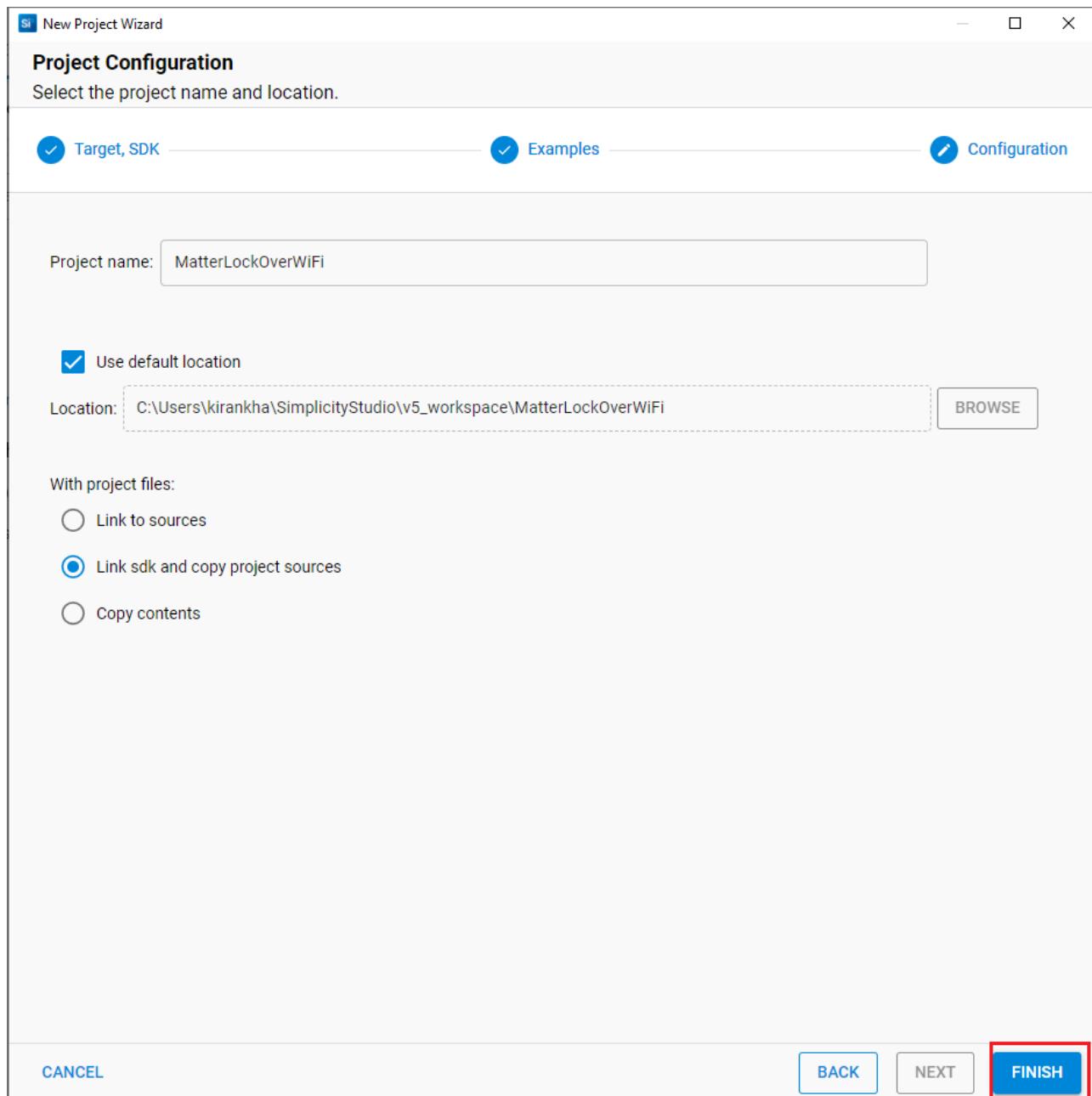
Board



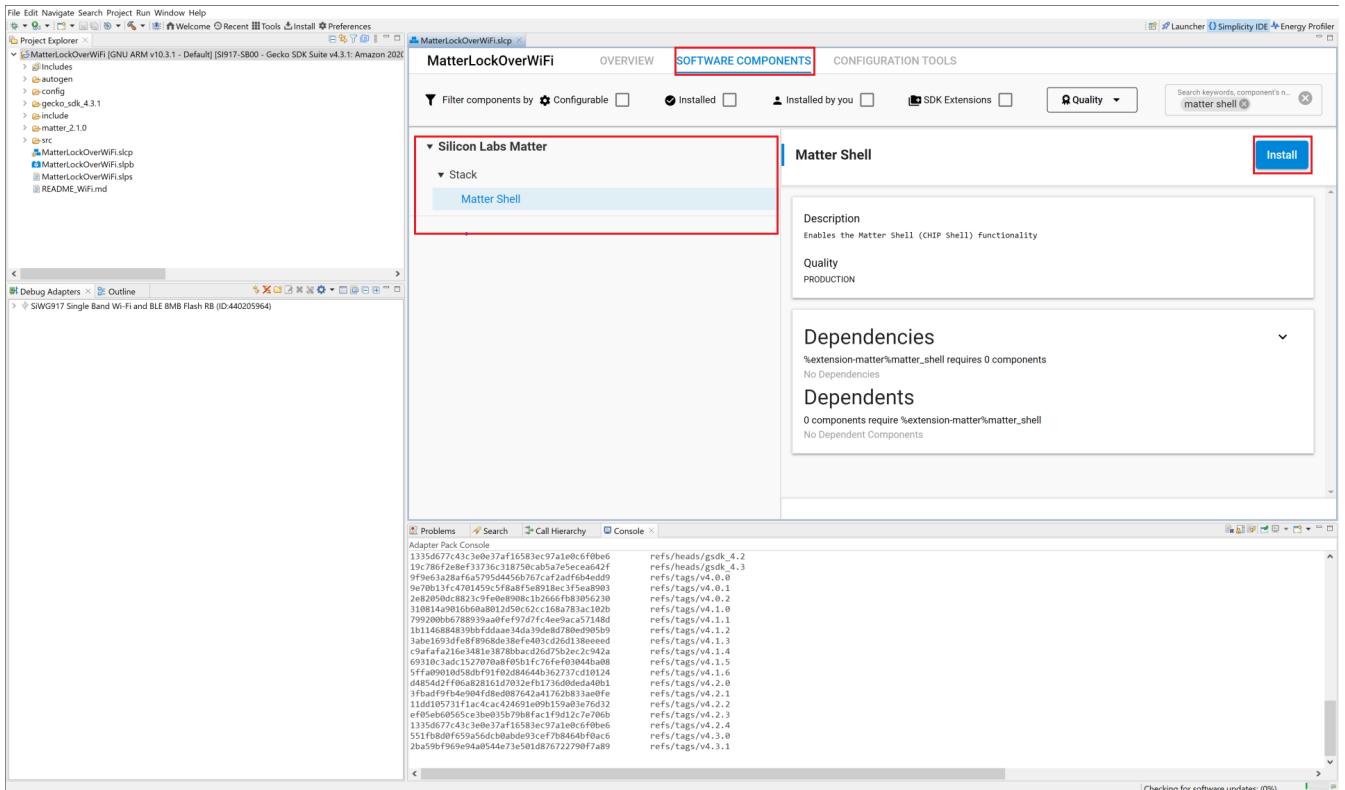
8. Open the Example Projects and Demos tab and create a project for Matter Lock Application.



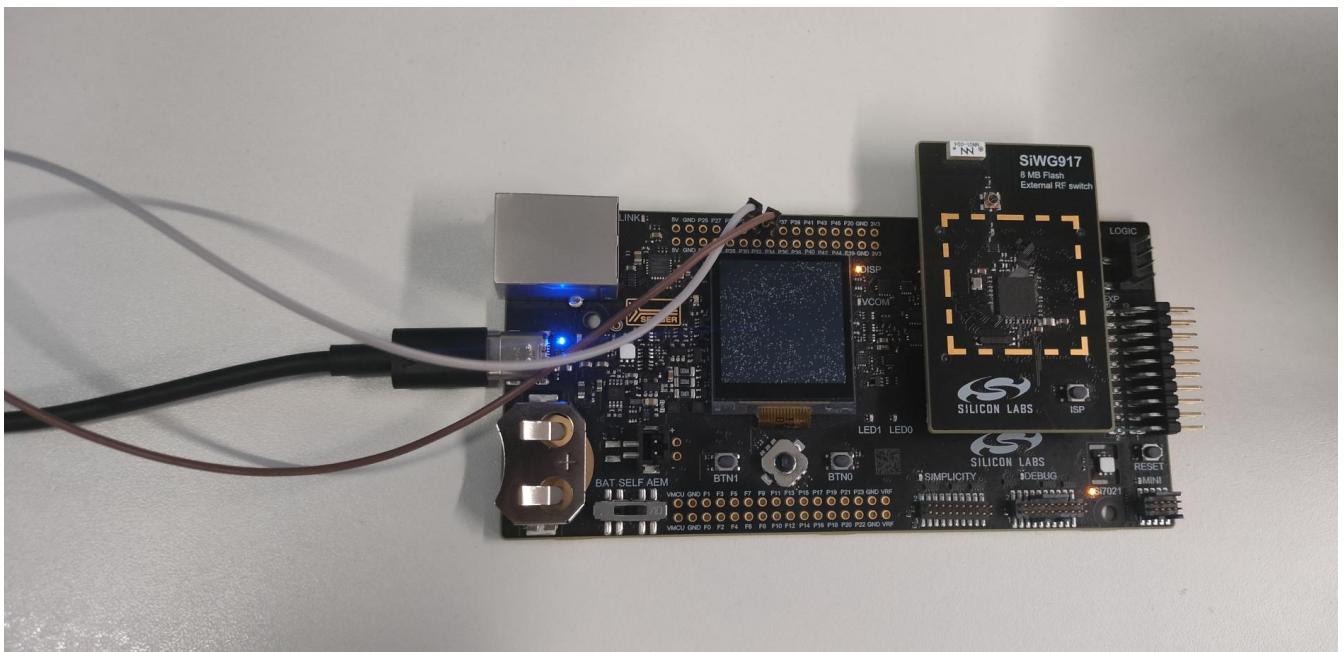
9. In the New Project Wizard window, click **Finish**.



10. After creation of project, open the Software Components tab and in search bar type **Matter Shell** and install it.

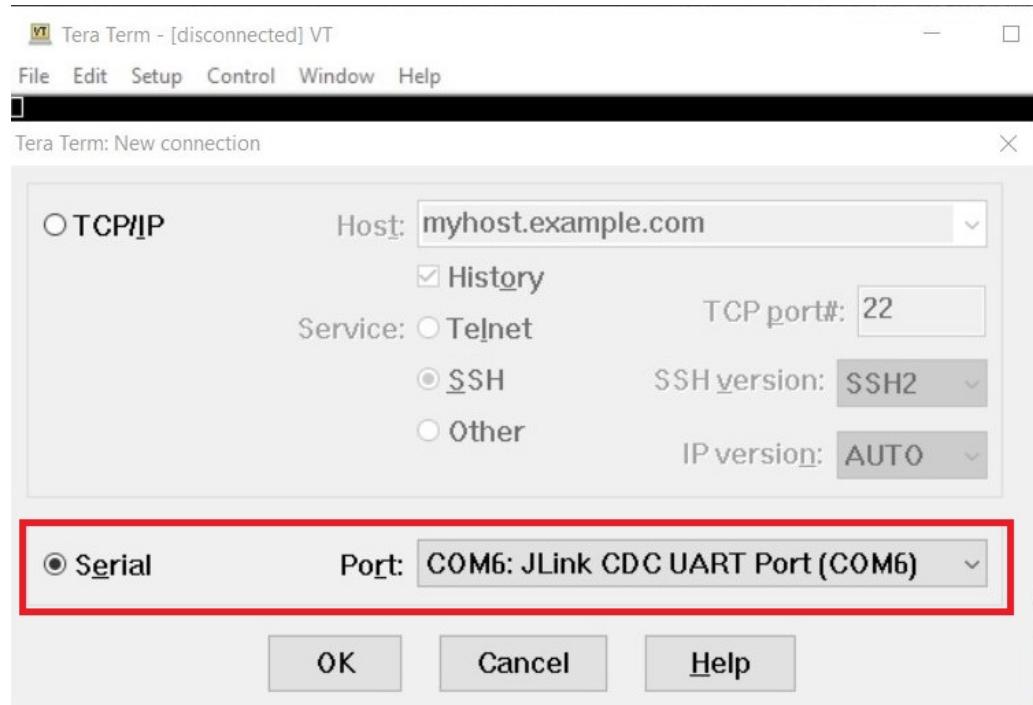


11. Build the project after enabling **Matter Shell** component.
12. After a successful build, commission the device as described in [Commission Matter Platform](#).
13. For SiWx917 SOC Connect the TTL cables with Radio Board to execute **Matter Shell**.

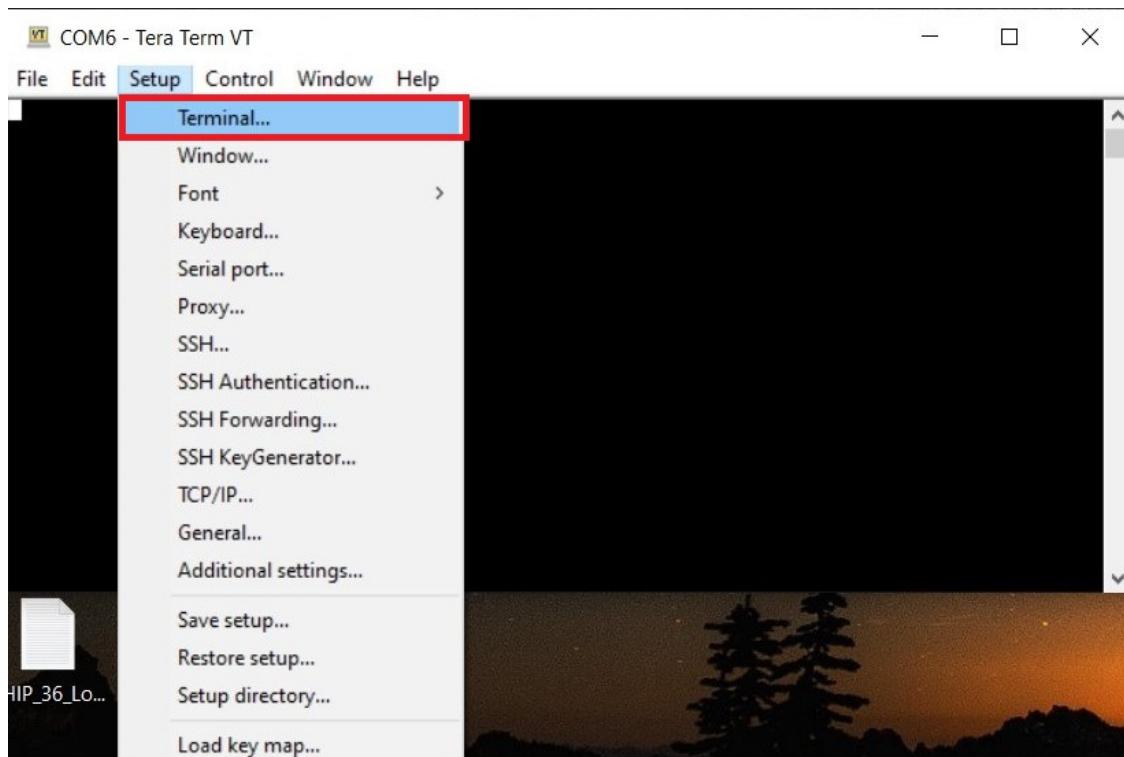


Note:- For EFR32MG2x Devices TTL Cable support is not required.

14. Open Tera Term and under New Connection, under Serial Port, select JLink port and click OK.

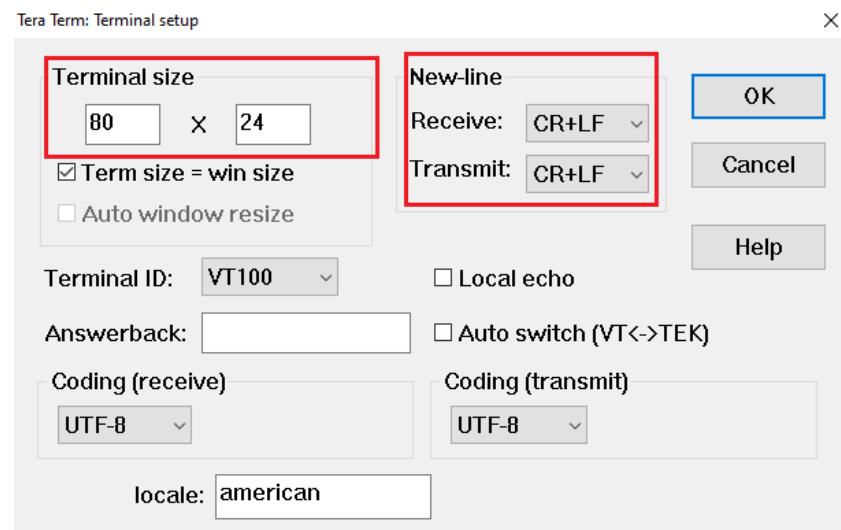


15. Click on Setup from Menu bar and change the value to 115200 under Speed category, then click on New setting.

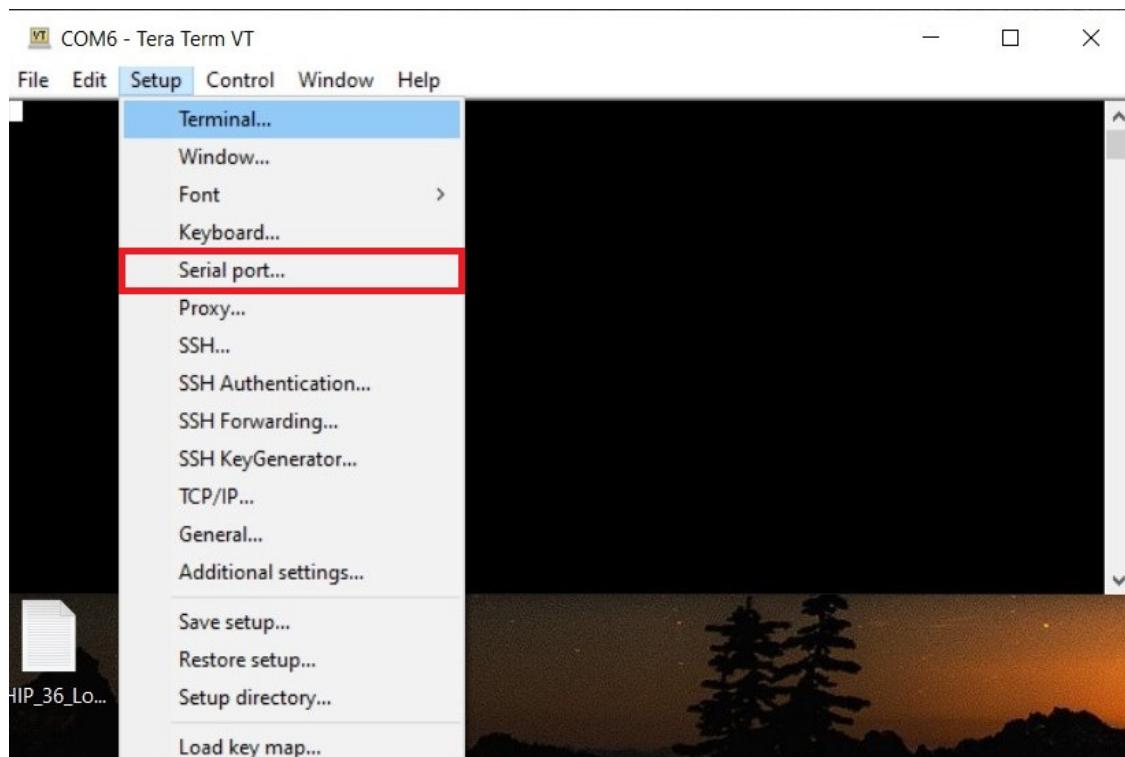


16. Inside Terminal Set the below values and click OK.

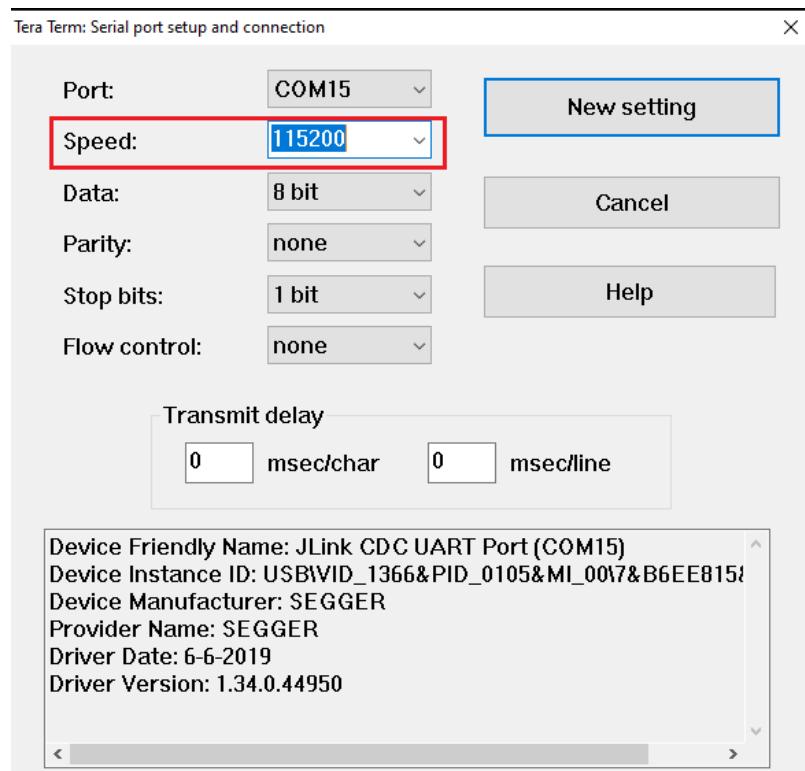
- Terminal Size : 80 * 24
- New-Line
 - Receive : CR+LF
 - Transmit : CR+LF



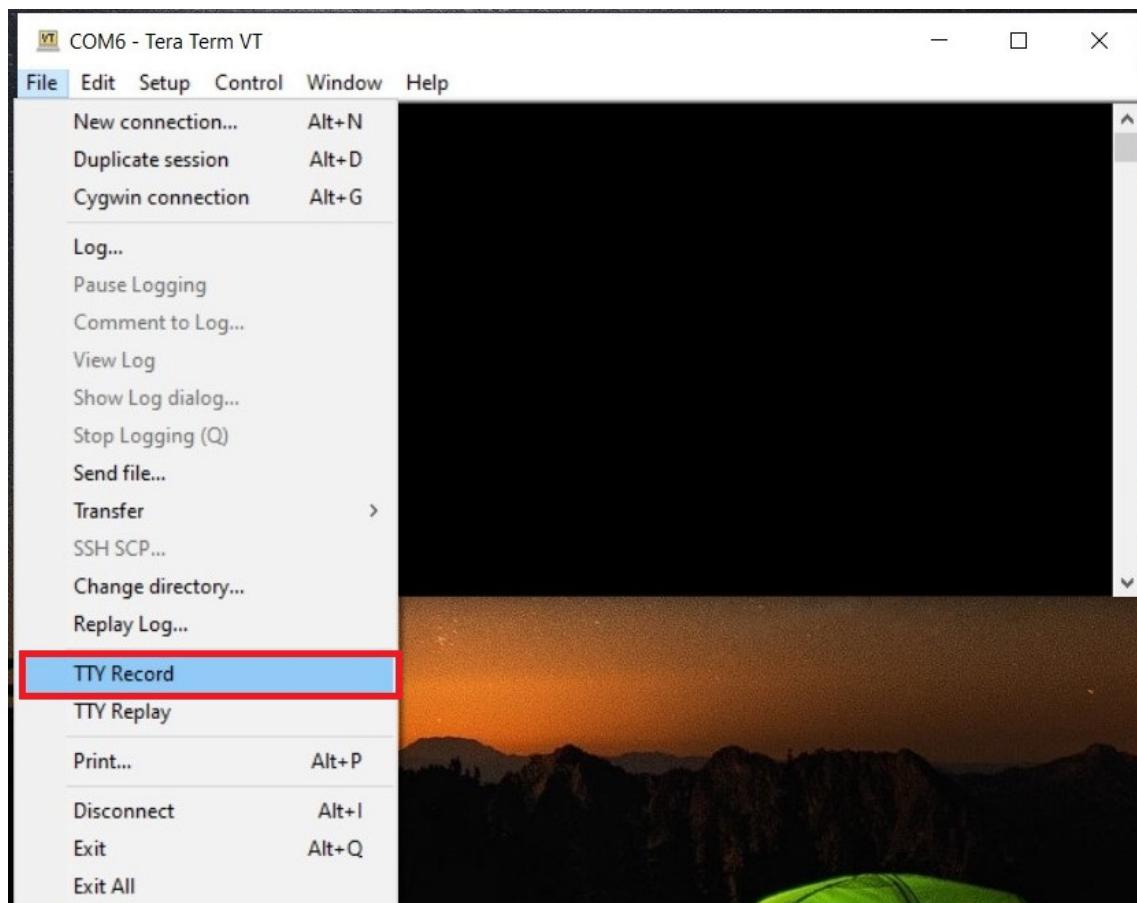
17. Click on File from Menu bar again, select Serial Port option.



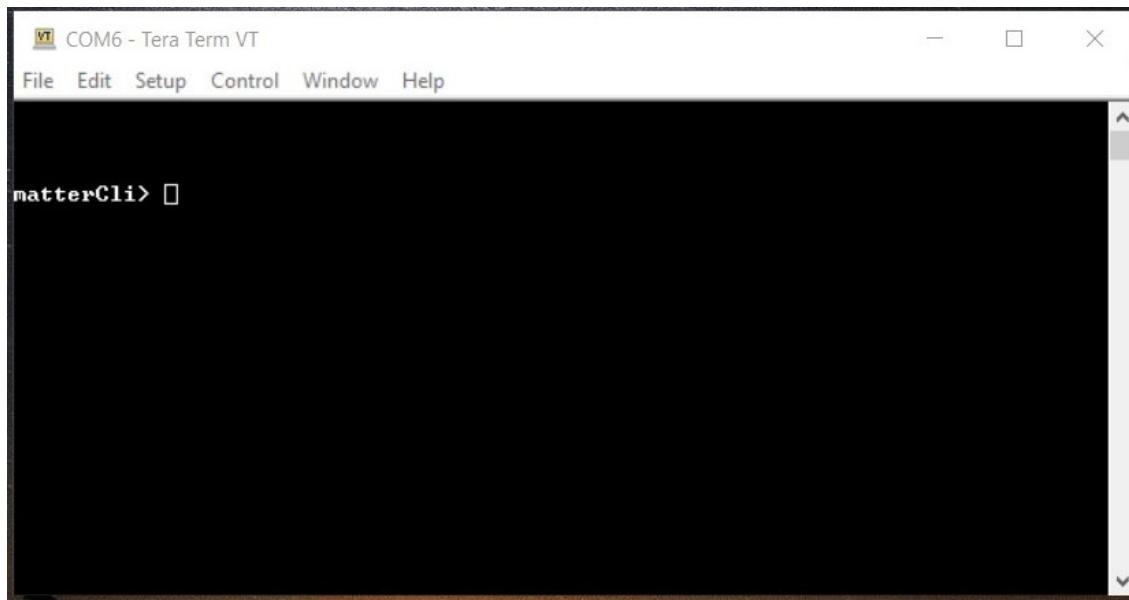
18. Increase the speed to 115200 and click on New setting.



19. Click on File from Menu bar, select TTY Record. Create any empty file with extension ".tty" and click on save.

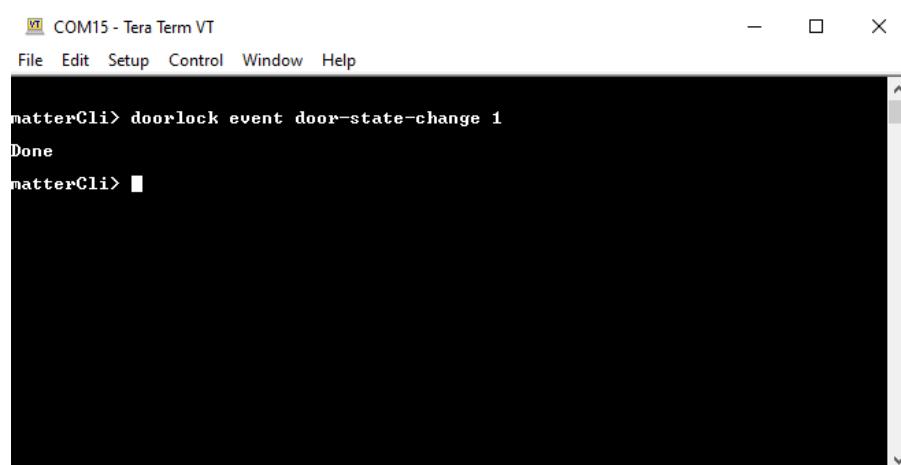


20. After creating tty file just click on **Enter** button from Keyboard then it will show you **matterCli** terminal.



21. Send any command through **matterCli** terminal, from the below list of commands:

- doorlock event door-state-change "DoorState"
 - Door State List
 - DoorOpen = 0
 - DoorClosed = 1
 - DoorJammed = 2
 - DoorForcedOpen = 3
 - DoorUnspecifiedError = 4
 - DoorAjar = 5
- doorlock event lock-alarm "AlarmCode"
 - Alarm Code List
 - LockJammed = 0
 - LockFactoryReset = 1
 - LockRadioPowerCycle = 3
 - WrongCodeEntryLimit = 4
 - FrontEsceutcheonRemoved = 5
 - DoorForcedOpen = 6
 - DoorAjar = 7
 - ForcedUser = 8
- onboardingscodes ble, command will show QR Code.



22. After changing DoorState and AlarmCode in **matterCli**, run the commands below using chip-tool on Raspberry PI to verify the event.

- To Read Door State
./chip-tool doorlock read-event door-state-change "node_id" "endpoint"
- To Read Alarm Code
./chip-tool doorlock read-event door-lock-alarm "node_id" "endpoint"

Note: Type **help** in matterCli terminal for more information about supported features.

Matter SLC CLI Setup and Build Instructions

Creating Matter Applications using SLC CLI

The Silicon Labs Configurator (SLC) offers command-line access to application configuration and generation functions. [Software Project Generation and Configuration with SLC-CLI](#) provides a complete description and instructions on downloading and using the SLC-CLI tool.

This guide lists the steps to create and build a Silicon Labs Matter SLC project using SLC-CLI and `make`. These scripts are evaluation quality and have been verified to work on Ubuntu 22.04.3 LTS, MacOS Version 13.5.1, and Windows 10.

Setting up the Environment

Clone Gecko SDK:

```
git clone https://github.com/SiliconLabs/gecko_sdk.git
```

Create a directory named `extension` inside the Gecko SDK directory.

Clone the Matter GSDK Extension inside the `extension` directory:

```
git clone https://github.com/SiliconLabs/matter_extension.git
```

Your path to the Matter extension should look like

```
<Path/To/Gsdk/Download>/extension/matter_extension
```

Install the following python packages:

```
pip3 install dload
```

Change directory to cloned extension directory and run the `sl_setup.py` script. This will install the ARM gcc toolchain, SLC-CLI, ZAP, Simplicity-commander, and Java.

For Mac and Linux:

```
cd extension/matter_extension
python3 slc/sl_setup_env.py
```

For Windows:

```
cd extension\matter_extension
python slc\sl_setup_env.py
```

The `sl_setup_env.py` script creates an `.env` file to be used to set the environment variables needed for the installed tools, ARM toolchain, SLC-CLI, Java ZAP, Simplicity-commander, and Java.

Creating an Application Project

Run the `sl_create_new_app.py` script to create a BRD4161A project with name `MyNewApp` starting from the `lighting-app-thread.slcp` example application project file:

The script will ask user permission to trust the `gecko_sdk` and `matter_extension` before generating.

For Mac and Linux:

```
python3 slc/sl_create_new_app.py MyNewApp slc/sample-app/lighting-app/efr32/lighting-app-thread.slcp brd4161a
```

For Windows:

```
python slc\sl_create_new_app.py MyNewApp slc\sample-app\lighting-app\efr32\lighting-app-thread.slcp brd4161a
```

Building an Application Project

After a project is created the `sl_build.py` script can be used to re-generate the `MyNewApp` project and build it:

For Mac and Linux:

```
python3 slc/sl_build.py MyNewApp/lighting-app-thread.slcp brd4161a
```

For Windows:

```
python slc\sl_build.py MyNewApp\lighting-app-thread.slcp brd4161a
```

Alternately, one can use SLC-CLI commands directly to generate the project and then use `make` to build it.

Windows users will need to install `make` in their system. You can use your own or follow these steps to get `make`.

1. Install the MSYS terminal, which provides a Unix-like environment on Windows.
2. Open the MSYS terminal and install `make` using the command `pacman -S make`.
3. Run command `where make`, copy the path, and add it to the PATH environment variable.
4. Restart your command line terminal and run `slc/sl_build.py` or run `make` directly. You might need to reboot.

Note: In rare cases, the build may fail due to missing files in the `zap-generated/` directory. The workaround is to delete the `.zap` folder in the home directory.

Modifying an Application Project

The resulting user project can be modified like any other SLC project: software components can be added or removed by modifying the project's `.slcp` file, configuration can be applied by modifying the files in the `config` directory, the application logic can be managed through the files in the `src` directory. Various SLC-CLI commands can be used to examine, validate, or re-generate the project after a modification, see [Software Project Generation and Configuration with SLC-CLI](#) for more information.

For modifying Matter endpoints and clusters invoke the ZAP tool passing to it the application's ZAP file:

```
./scripts/tools/zap/run_zaptool.sh MyNewApp/config/common/lighting-thread-app.zap
```

Matter Solutions

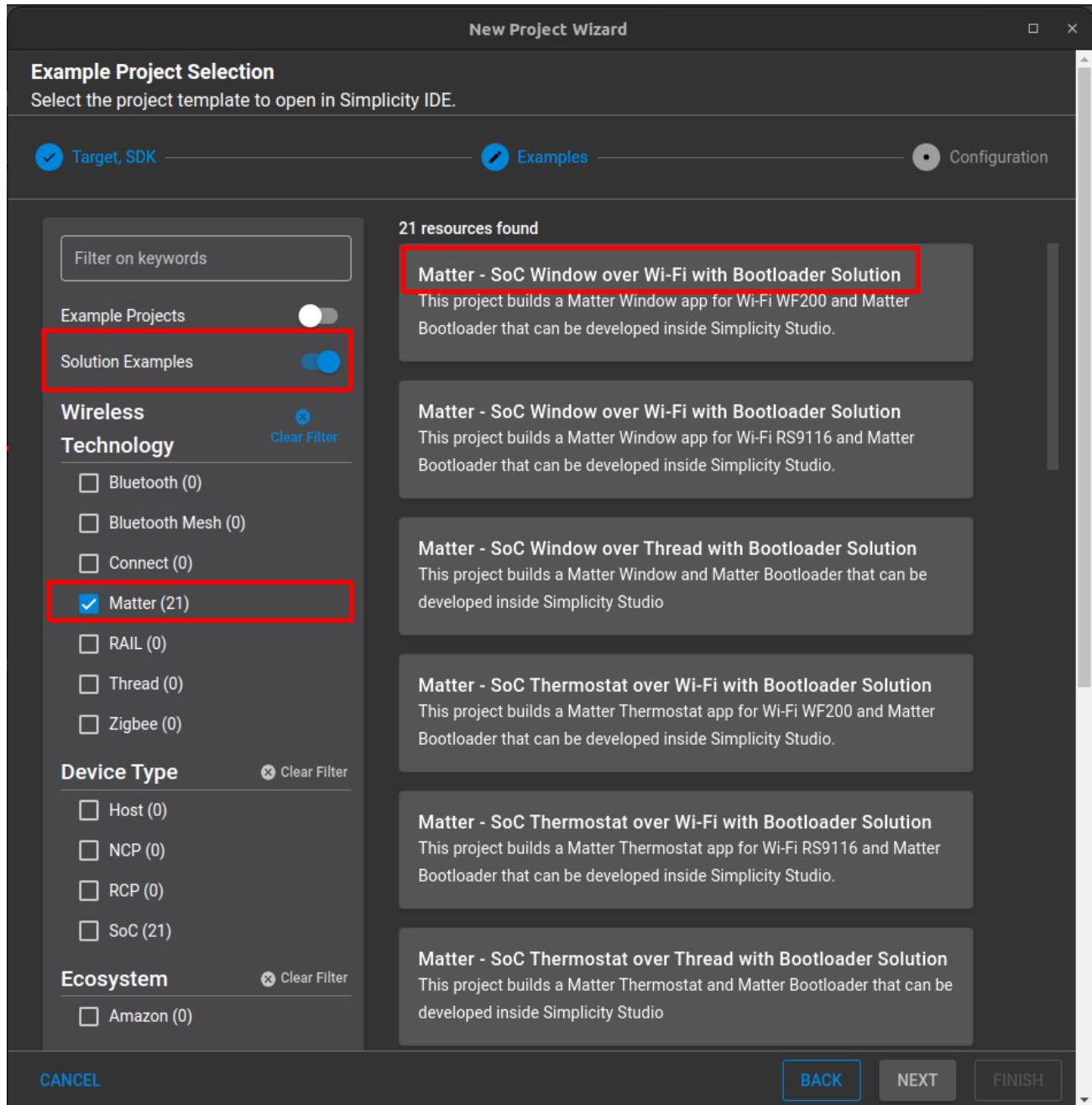
Solutions

General

Matter solutions allow the user to generate multiple projects at once, to generate a combined solution/binary and allow multiple post-build operations to provide flexibility and unison when developing a Matter example or application. For example, with the Matter Lighting over Thread with Bootloader Solution a user can generate a Matter Bootloader, a Matter Lighting example, and the combined Bootloader + Example production image. Solutions will also generate a .gbl binary for the users to make use of and create an .ota software image update binary (directions listed below).

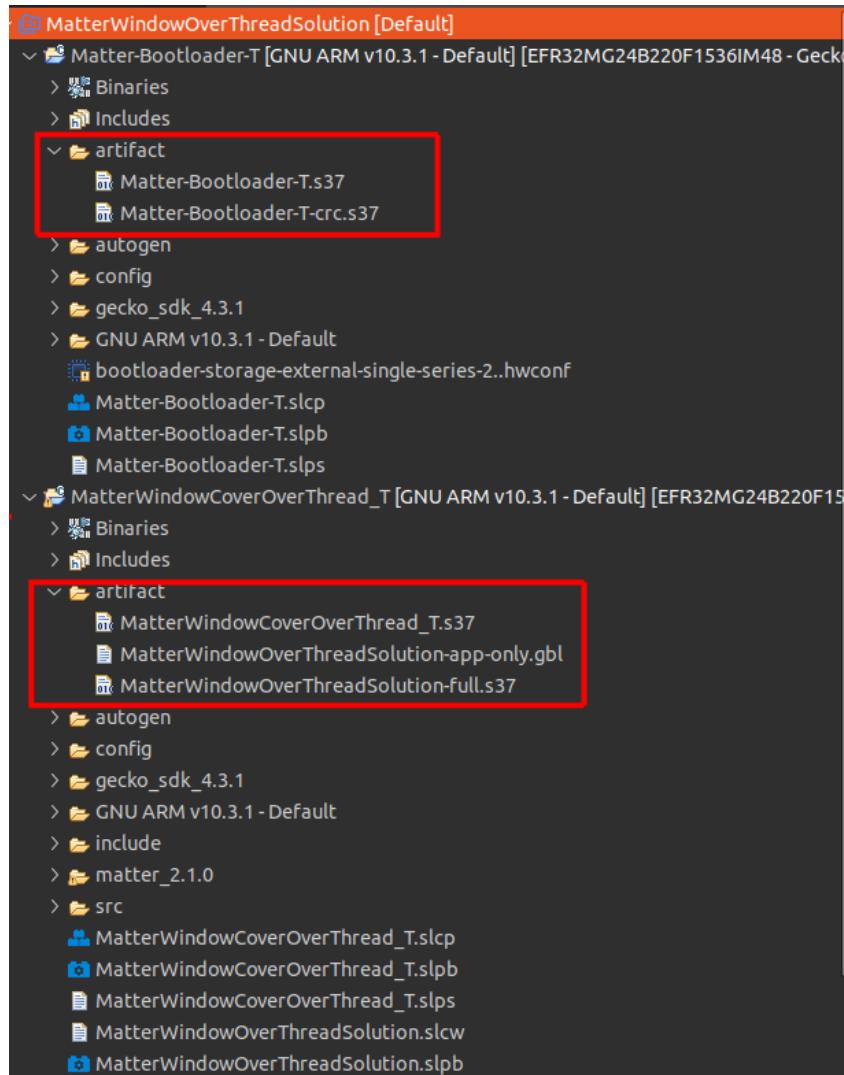
Solution Creation

To create a Matter Solution, proceed to the "Example Project Selection" section of Simplicity Studio's "New Project Wizard". On the left hand side, ensure "Example Projects" is turned off, "Solution Examples" is turned on, and "Matter" under the "Technology" section is checked. Then, select for which application example you wish to create a project from.



Solution Building

Building via solutions behaves just like a normal project. Just ensure the top-level solution is selected and build! Artifacts from the resulting projects can be found within the `artifact` directory under each distinct project within the solution.



OTA Creation

Due to certain limitations with the way our Matter examples are built within Simplicity Studio, OTA file generation must be conducted by the user via the command line.

To create an OTA file, first build a Matter example via solutions. Locate the resulting .gbl file within the `artifact` directory. This will be used as an argument to the GBL creation script. Then, locate the directory that holds the Matter Extension within the GSDK in Simplicity Studio. The location should be similar to:

`/Users/User/SimplicityStudio/SDKs/gecko_sdk/extension/matter_extension`. Once found, open a terminal at this location and run the command:

```
./src/app/ota_image_tool.py create -v 0xFFFF1 -p 0x8005 -vn <SoftwareVersion> -vs <SoftwareVersionString> -da sha256
<PathToArtifactDirectory>/<GblFile> <PathToArtifactDirectory>/<ResultingOtaFileName>
```

Where `SoftwareVersion` and the `SoftwareVersionString` correspond to the `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` and `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING` parameters the project has been compiled with. And where `ResultingOtaFileName` is the name of the OTA file you wish to generate, `PathToArtifactDirectory` is the location of your artifact directory within your project, and `GblFile` is the name of the GBL file that was produced after building the solution. The OTA file should now have been populated within the `artifact` directory in Simplicity Studio.

Matter OTA

Matter OTA

The Over The Air (OTA) Software Update functionality is enabled by default for all of the EFR32 and SiWx917 example applications. Its inclusion in an application is controlled by the OTA Requestor component in a Matter Studio project.

- [Matter OTA Bootloader](#)
- [Matter OTA Software Update](#)
- [Matter 917 SOC OTA Software Update](#)
- [Matter OTA WiFi Project](#)

Matter OTA Bootloader

Creating a Gecko Bootloader for Use in Matter OTA Software Update

The Matter OTA Software Update functionality on EFR32 devices requires the use of a Gecko Bootloader built with correct configuration parameters. The key parameters are the storage slot size and (in case of internal storage) storage slot address. The current document lists the steps required to build the Gecko Bootloader for Matter OTA Update and discusses the configuration parameter selection. For a detailed discussion of Gecko Bootloader refer to *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

The Gecko Bootloader is built with Silicon Labs Simplicity Studio. These instructions assume that you have installed Simplicity Studio 5, the Simplicity Commander tool (installed by default with Simplicity Studio), the GSDK and associated utilities, and that you are familiar with generating, compiling, and flashing an example application in the relevant version.

Bootloader Project In Studio

Creating the Project

In Simplicity Studio click on Project->New->Silicon Labs Project Wizard to create a new project. Select the correct Target Board, SDK and the Toolchain.

In the next screen select the example project the bootloader will be based on. For a bootloader using external storage select "Bootloader SoC SPI Flash Storage (single image with slot size of 1024K)". For a bootloader using internal storage select "Bootloader - SoC Internal Storage (single image on 512kB device)"

Configuring Storage Components and Parameters

In the newly created project select the project's .slcp file, click the "Software Components" tab, and select Platform->Bootloader->Storage. In the Bootloader Storage Slot component (it should be already installed) configure Slot 0's Start Address and Slot size.

- For external storage bootloaders the Start Address should be 0 and Slot size should be 1048576 -- both values are set by default
- For internal storage bootloaders see the "Internal Bootloader: Image Size, Selecting Storage Slot Address and Size" section below In the Common Storage component leave the "Start address of bootload info" at 0.

Configuring Other Components

It is recommended to install the "GBL Compression (LZMA)" component under Platform->Bootloader->Core: this allows the bootloader to handle compressed GBL files. This component is required for internal storage bootloaders.

At this point the project contains all the components necessary to support the Matter OTA Software Update functionality. Other components can now be added to support additional features such as Secure Boot. Refer to *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher* for the description of various Bootloader features and the steps to enable them.

Building and Flashing the Bootloader

Build the project by clicking on the hammer icon in the Studio toolbar. Flash the bootloader to the board using the "Upload Application" option from the Debug Adapters view.

Combined bootloader for MG12 boards

The MG12 boards (which are Series 1 EFR32 boards) require a combined bootloader image (first stage bootloader + main bootloader) the first time a device is programmed -- whether during development or manufacturing. For subsequent programming, if the combined bootloader had been previously flashed to the device use the regular version.

To create the combined bootloader follow this additional step (Step 6 in Section 6 of *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*) before clicking the build icon: Right-click the project name in the Project Explorer view and select Properties. In the C/C++ Build group, click Settings. On the Build Steps tab, in the Post Build Steps Command field enter

```
$ ./postbuild.sh "${ProjDirPath}" "${StudioSdkPath}" "${CommanderAdapterPackPath}"
```

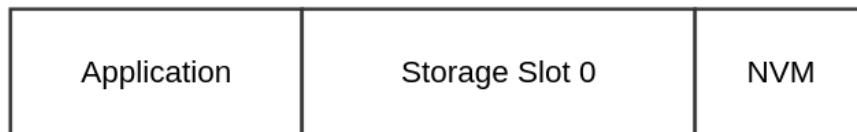
Click Apply and Close. Three bootloader images will be generated into the build directory: a main bootloader, a main bootloader with CRC32 checksum, and a combined first stage and main bootloader with CRC32 checksum. The main bootloader image is called [project-name].s37, the main bootloader with CRC32 checksum is called [projectname]-crc.s37, while the combined first stage image + main bootloader image with a CRC32 checksum is called [projectname]-combined.s37.

Internal Bootloader: Image Size, Selecting Storage Slot Address and Size

The internal storage bootloader for Matter OTA Software Update is supported on MG24 boards only. In this use case both the running image and the downloadable update image must fit on the internal flash at the same time. This in turn requires that both images are built with a reduced feature set such as disabled logging and Matter shell (see [here](#) for the list of features). Using LZMA compression when building the GBL file further reduces the downloaded image size.

When building an internal storage bootloader the two key configuration parameters are the Slot Start Address and Slot Size in the Bootloader Storage Slot component. The storage slot must not overlap with the running image and the NVM section of the flash. In other words, the slot start address must be greater than the end of the running image address and the sum of the start address and the slot size must be less than the address of the NVM section.

Internal Flash Layout



The simplest way to get the relevant addresses for the running image and NVM is by using the Simplicity Commander tool:

- Build the running image for the Matter application
- Erase the chip and flash the running image to it (For example: use Simplicity Studio's Debug Adapters view context menu to flash the application image and some bootloader valid for the device board. Make sure to select the "Erase chip before uploading image" option).
- In Simplicity Commander, select Device Info -> Flash Map. The blank area in the middle of the flash (between the running image in the beginning and NVM at the end) is available for the bootloader storage slot. Each block represents a flash page (8K on MG24 boards). Hovering the mouse over a block shows the block's start and end address.
- Set the Slot Start Address to be the address of the first available block. Calculate the Slot Size to be the difference between the end address of the last free block and the Slot Start Address. The Slot Size must be greater than the size of the GBL file for the update image.
- (Optional) It might be advisable to set the Slot Start Address to the beginning of the second or third available block to account for potential growth of the application image -- this way the bootloader won't have to be reconfigured for every increase in the image size. The storage slot must still be able to accommodate the GBL image for the update.

Another way to calculate the Storage Slot parameters is by examining the application's .map file:

- Build the running image for the Matter application
- In the application .map file find the highest address preceding the .data section, round it up to align on the 8K page boundary (e.g. 0x00000000080f1704 would round up to 0x00000000080f2000) and then add 0x2000 get the next page block address -- the result would be the Slot Start Address. The address of the .nvm section in the .map file is the end of the space available for the Storage Slot. The Slot Size is the difference of the .nvm address and the Slot Start Address.

Example

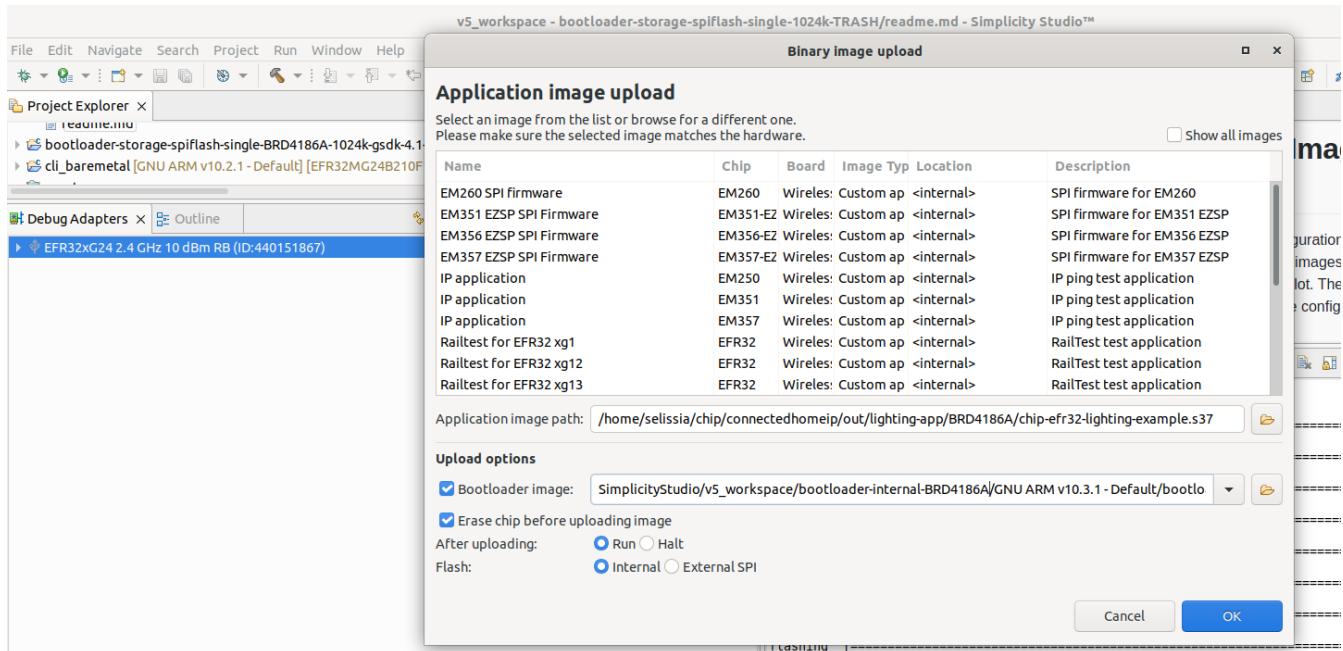
This example is for an internal storage bootloader for the Matter lighting app on BRD4186C.

- Build the application in Simplicity Studio after disabling all optional features such as the Matter QR Code, Matter Display, Matter Shell, OpenThread CLI components.
- Build the GBL file for the update image and note its size

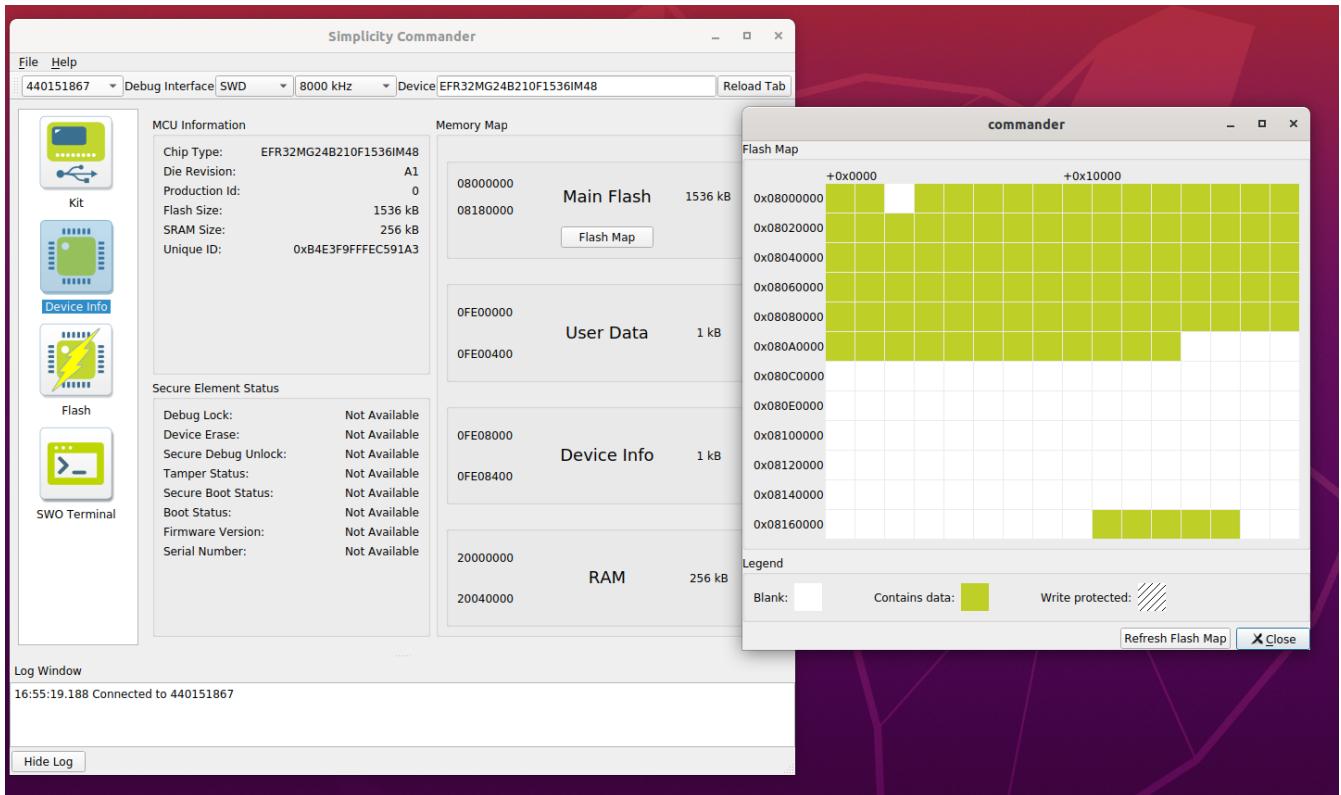
```
$ commander gbl create --compress lzma ~/chip/connectedhomeip/out/lighting-app/BRD4186A/chip-efr32-lighting-example.gbl --app
~/chip/connectedhomeip/out/lighting-app/BRD4186A/chip-efr32-lighting-example.s37

$ ls -la out/lighting-app/BRD4186A/chip-efr32-lighting-example.gbl
451176 Jul 19 16:39 out/lighting-app/BRD4186A/chip-efr32-lighting-example.gbl
```

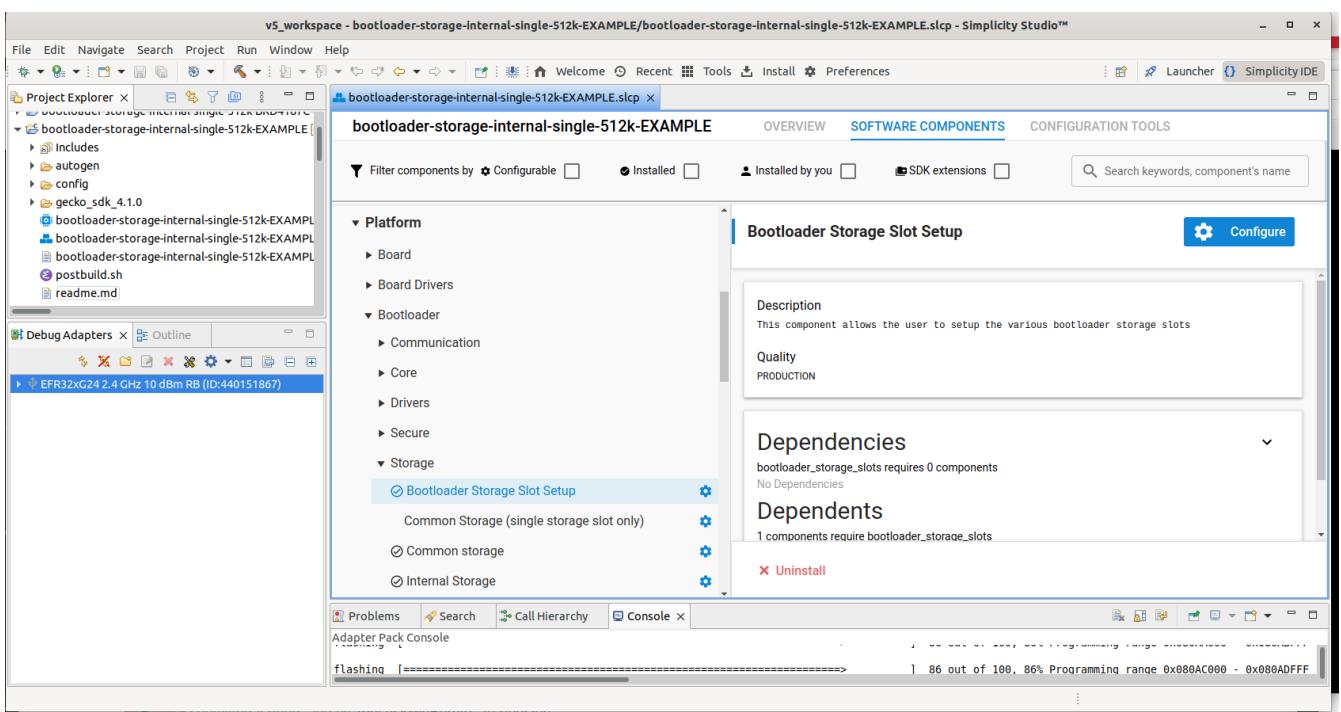
- Flash the application image, bootloader. Erase the flash.



- In Simplicity Commander display the flash map



- The address of the first available page is 0x080b8000, the end address of the last available block is 0x08172000. This means you can set the Slot Start Address to 0x080b8000 and the Slot Size to 761856 (761856 = 0x08172000 - 0x080b8000). The slot size is sufficient for our GBL file (451176 bytes)
- Create a project base on the "Bootloader - SoC Internal Storage (single image on 512kB device)" example. Configure the Bootloader Storage Slot component and set Slot Address and Slot Size.



- Enable the "GBL Compression (LZMA)" component.
- Build the project

Matter OTA Software Update

Matter OTA Software Update with Silicon Labs Example Applications

This page outlines the steps for a scenario that demonstrates the The Over The Air (OTA) Software Update functionality in Matter.

The Over The Air (OTA) Software Update functionality is enabled by default for all Silicon Labs example applications. Its inclusion in an application is controlled by the OTA Requestor component in a Matter Studio project.

Overview

The OTA Software Update scenario requires the following binaries:

- **OTA-A**, the running image: a regular application built with the default/older software version value. This application will be updated to the one with a higher software version. In the OTA Software Update process it acts as the OTA Requestor.
- **OTA-B**, the update image: a regular application built with a higher software version value.
- **Chip-tool**: the controller that announces the OTA-Provider's address to the application thus triggering the OTA Software Update.
- **OTA-Provider**: the server that carries the update image and from which the OTA Requestor will download the updated software.
- **Bootloader**: Silicon Labs Gecko Bootloader image that supports OTA; supports the external (SPI-flash) or the internal storage option.

Setting up the OTA Environment

Setting up chip-tool

The chip-tool binary is a part of the Silicon Labs' Matter Hub Raspberry Pi Image available as a part of the Release Artifacts page. If you are planning to run chip-tool on the Matter Hub you may skip the rest of this section.

If you have not downloaded or cloned this repository, you can run the following commands on a Linux terminal running on a Mac, Linux, WSL or Virtual Machine to clone the repository and run bootstrap to prepare to build the sample application images.

1. To download the [SiliconLabs Matter codebase](#), run the following commands.

```
$ git clone https://github.com/SiliconLabs/matter.git
```

2. Bootstrapping:

```
$ cd matter
$ ./scripts/checkout_submodules.py --shallow --recursive --platform efr32
$ . scripts/bootstrap.sh
# Create a directory where binaries will be updated/compiled called 'out'
$ mkdir out
```

To control the Matter application you will have to compile and run the chip-tool on either a Linux, Mac, or Raspberry Pi. The chip-tool builds faster on the Mac and Linux machines so that is recommended, but if you have access to a Raspberry Pi that will work as well.

3. Build the chip-tool

```
$ scripts/examples/gn_build_example.sh examples/chip-tool out/
```

Setting up OTA-Provider

The chip-ota-provider-app binary for a Raspberry Pi is a part of the Artifacts package available with the Matter Extension release. If you are planning to run the OTA-Provider on a Raspberry Pi there is no need to build it.

- To build the OTA-Provider app in Linux, run the following command in a Matter repository:

```
$ scripts/examples/gn_build_example.sh examples/ota-provider-app/linux out chip_config_network_layer_ble=false
```

Building Application Images Using Simplicity Studio

- The running image and the update image are regular Matter application images and are built using the standard procedure. The only additional configuration required is the use of a higher software version in the update image. See the following document for detailed steps: [build OTA application using studio](#).

Obtaining the Bootloader binary

- Build or download the Gecko Bootloader binary which can be obtained in one of the following ways:
 - Follow the instructions in [Creating the Bootloader for Use in Matter OTA](#)
 - Pre-built binaries (only valid for the external SPI-flash storage OTA update) are available on the [Matter Artifacts page](#).
 - Bootloader (only valid for the external SPI-flash storage OTA update) project can be built as a part of any Matter Solution in Studio
- Using the commander tool or Simplicity Studio, upload the bootloader to the device running the application.

Running the OTA Download Scenario

- Create a bootable image file (using the Lighting application image as an example):

```
$ commander gbl create chip-efr32-lighting-example.gbl --app chip-efr32-lighting-example.s37
```

- Create the Matter OTA file from the bootable image file:

```
$ commander ota create --type matter --input chip-efr32-lighting-example.gbl --vendorid 0xFFFF --productid 0x8005 --swstring "2.0" --swversion 2 --digest sha256 -o chip-efr32-lighting-example.ota
```

- In a terminal start the Provider app and pass to it the path to the Matter OTA file created in the previous step:

```
$ rm -r /tmp/chip_kvs_provider
```

```
./out/chip-ota-provider-app --KVS /tmp/chip_kvs_provider -f chip-efr32-lighting-example.ota
```

- In a separate terminal run the chip-tool commands to provision the Provider:

```
$ ./out/chip-tool pairing onnetwork 1 20202021
```

```
$ ./out/chip-tool accesscontrol write acl '[{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [112233], "targets": null}, {"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": null, "targets": null}]' 1 0
```

- For Matter over OpenThread, bring up the OpenThread Border Router and get its operational dataset, for Matter over WiFi bring up the AP.
- If the application device had been previously commissioned, hold Button 0 for six seconds to factory-reset the device.
- Commission the device. For Matter over OpenThread:

```
$ ./out/chip-tool pairing ble-thread 2 hex:<operationalDataset> 20202021 3840
```

where operationalDataset is obtained from the OpenThread Border Router.

For Matter over WiFi:

```
./out/chip-tool pairing ble-wifi "node_id" "SSID" "PSK" 20202021 3840
```

- Once the commissioning process completes enter:

```
$ ./out/chip-tool otasoftwareupdaterequestor announce-otaprovider 1 0 0 0 2 0
```

- The application device will connect to the Provider and start the image download. Once the image is downloaded the device will reboot into the downloaded image.

Internal Storage Bootloader

Internal storage bootloader for Matter OTA software update is supported on MG24 boards only. In this use case both the running image and the downloadable update image must fit on the internal flash at the same time. This in turn requires that both images are built with a reduced feature set, such as disabled logging and Matter shell. See [Creating the Bootloader for Use in Matter OTA](#) for more details.

Installing the Lower Power Mode component in the project's Software Components tool in Studio will uninstall the following optional components and reduce the image size:

```
Matter QR Code Display,  
Matter Display,  
Matter Shell,  
OpenThread CLI.
```

Disabling logging in the configuration of the Matter Core Components component also helps to reduce the image size.

Using LZMA compression when building the .gbl file (passing `--compress lzma` parameter to the `commander gbl create` command) further reduces the downloaded image size.

When building an internal storage bootloader the two key configuration parameters are the Slot Start Address and Slot Size in the Bootloader Storage Slot component. The storage slot must not overlap with the running image and the NVM section of the flash. In other words, the slot start address must be greater than the end of the running image address and the sum of the start address and the slot size must be less than the address of the NVM section. The simplest way to get the relevant addresses for the running image and NVM is by using the Silicon Labs Simplicity Commander (Device Info->Main Flash->Flash Map).

The pre-built bootloader binaries are configured with slot start address of 0x080EC000 and slot size of 548864

Managing the Software Version

In order for the Provider to successfully serve the image to a device during the OTA Software Update process the Software Version parameter that the .ota file was built with must be greater than the `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` parameter set in the application's `sl_matter_config.h` file which is a config file for the Matter Core Components component in the Matter Studio project. The Software Version parameter is set by the `-vn` parameter passed to the `commander ota create` command. For example, if the application's running image was built with `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` set to 1 and if the `.ota` file is built with `-vn 2` then the Provider will serve the update image when requested.

In order for the OTA Software Update subsystem to consider an update to be successful and for the `NotifyUpdateApplied` command to be transmitted the `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` in the updated image must exceed the software version of the running image (continuing the above example, the image for the update must be built with `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` set to 2).

Managing the Vendor and Product ID

Starting the `ota-provider-app` with the `--otalmageList` command line option allows the user to supply a JSON file specifying the Software Version, Vendor and Product ID that identify the image served by the Provider, see the `ota-provider-app` for Linux in examples directory.

Example provider configuration file:

```
{ "foo": 1, // ignored by parser
  "deviceSoftwareVersionModel": [
    [
      {
        "vendorId": 65521, "productId": 32773, "softwareVersion": 1, "softwareVersionString": "1.0.0", "cDVersionNumber": 18,
        "softwareVersionValid": true, "minApplicableSoftwareVersion": 0, "maxApplicableSoftwareVersion": 100, "otaURL": "chip-efr32-lighting-example.ota"
      }
    ]
  }
}
```

Additional Info

Developers can find more resources on [Silicon Labs Matter Community Page](#).

Matter 917 SOC OTA Software Update

Matter OTA For 917 SOC

The scope of this page describes the MATTER OTA upgrade on 917 SoC mode for combined image(TA+M4) as well as single M4 and TA image upgrade.

Hardware Requirements

- To run matter ota on Silicon Labs Platform, refer to [Hardware Requirements](#).

Software Requirements

To run matter ota on Silicon Labs Platform, refer to [Software Requirements](#).

Setting up OTA Environment

To run OTA on Matter over Wi-Fi, Need to build two different application below:

- **OTA-A** is a normal application with default or older software version. It acts as **ota-requestor** where it needs to update latest software version.
- **OTA-B** is a normal application with updated software version.
- **Chip-tool** is a controller for sending commands to ota-requestor to update the software version and receiving commands from device.
- **OTA-Provider** is the server who has the latest software version and from which ota-requestor will download the updated software.

Building OTA Application Using Simplicity Studio For 917 SOC

To create and build Matter OTA using Simplicity Studio, refer to [build OTA application using Simplicity Studio](#).

Combined Image Upgrade

For 917 SoC, storing a single Matter combined upgrade image(TA+M4) and then providing sample code that can transfer the image to the co-processor and rewrite the 917 firmware as well as M4 firmware Image then boot loading with the upgraded TA processor image and the M4 processor image.

Host will initiate OTA download to receive combined image (TA+M4) on to host. Host will store M4 and TA image on flash backup location.

Use Case Of OTA

- Combined image will be created and uploaded onto Raspberry Pi which provides the firmware image chunk by chunk to the device.
- Host will initiate the OTA download and provider app will start the OTA image transfer.
- Host will receive combined image and host will transfer the M4 and TA firmware images on to TA chunk by chunk, TA will write the TA image onto TA flash backup location.
- Once both images are downloaded, the device will reboot into the downloaded image.

Create Combined (TA+M4) Firmware Image

- The first step is to create a combined Image that contains both the firmware (TA & M4).
- This image is created by combining the binary images of both images.

- For Matter OTA file, create a bootable image file (using the Lighting application image as an example) and then create the Matter OTA file from the bootable image file using commands provided below.
- Once combined image .ota file is created , the same will be uploaded onto raspberry pi where OTA provider application is running.

Generating The Combined OTA image

- Create TA image (.rps) with combined image flag set by using command.

```
commander rps convert <ta_image_combined.rps> --taapp <ta_image.rps> --combinedimage
```

- Create M4 .rps file from .s37 using below command.

```
- commander rps create <m4_image.rps> --app <m4_image.s37>
```

- Create M4 (.rps) with combined image flag set by using command.

```
commander rps convert <m4_image_combined.rps> --app <m4_image.rps> --combinedimage
```

- Create combined image from the above created TA and M4 images.

```
commander rps convert "combined_image.rps" --app "m4_image_combined.rps" --taapp "ta_image_combined.rps"
```

- Create the Matter OTA file from the bootable image file.

```
./src/app/ota_image_tool.py create -v 0xFFFF1 -p 0x8005 -vn 2 -vs "2.0" -da sha256 combined_image.rps combined_image.ota
```

Note: For TA(alone) OTA firmware upgrade, follow the same steps as [combined image](#)

Running OTA Provider

- Locate **ota-provider** terminal. Run the Provider app along with the Matter OTA file created in the previous step.

```
rm -r /tmp/chip_*
./out/debug/chip-ota-provider-app -f combined_image.ota
```

Setting Up OTA-Requestor

- Before running **ota-requestor** app, flash the bootloader binary images for Silicon Labs Devices.

Running OTA-Requestor

- Enhancements to the Wi-Fi sdk IOT firmware upgrade application for MATTER OTA combined firmware application.

1. In a separate terminal, locate the chip-tool and ota-requestor and run the chip-tool commands to provision the Provider.

```
./out/chip-tool pairing onnetwork 1 20202021
./out/chip-tool accesscontrol write acl [{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [112233], "targets": null}, {"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": null, "targets": null}]' 1 0
```

2. If the application device had been previously commissioned, hold Button 0 for six seconds to factory-reset the device.

3. In the chip-tool terminal, commission the Device by passing below command.

```
./out/chip-tool pairing ble-wifi "node_id" "SSID" "PSK" 20202021 3840
```

where SSID and PSK are AP username and password.

4. Once the commissioning process completes in the same terminal, run below requestor command to start downloading the image.

```
./out/chip-tool otasoftwareupdaterequestor announce-ota-provider 1 0 0 0 2 0
```

- The application device will connect to the Provider and start the image download. Once the image is downloaded, the device will reboot into the downloaded image.

Note: once image download is done, disconnect jlink. it will reboot automatically with new image.

Matter Software Update with SOC M4 Example Applications

- Host will initiate OTA download to receive M4 image on to host. Host will store M4 image on flash backup location.

Use Case M4(alone) OTA

- OTA image will be created and uploaded onto Raspberry Pi which provides the firmware image chunk by chunk to the device.
- Host will initiate the OTA download, and provider app will start the OTA image transfer.
- Host will receive M4 image and host will transfer the M4 image on to flash chunk by chunk.
- TA will write the M4 image onto flash backup location.
- Once image is downloaded, the device will reboot into the upgraded M4 image.

Generating The M4 OTA image

- Create M4 (.s37) image to (.rps) image using below command.

```
commander rps create <m4_image.rps> --app <m4_image.s37>
```

- Create the Matter OTA file from the bootable image file

```
./src/app/ota_image_tool.py create -v 0xFFFF1 -p 0x8005 -vn 2 -vs "2.0" -da sha256 m4_image.rps m4_image.ota
```

Running OTA Provider

- Locate **ota-provider** terminal, Run the Provider app along with the Matter OTA file created in the previous step.

```
rm -r /tmp/chip_*
./out/debug/chip-ota-provider-app -f m4_image.ota
```

Setting Up OTA-Requestor

- Before running **ota-requestor** app, flash the bootloader binary images for Silicon Labs Devices.

Running OTA-Requestor

- Enhancements to the Wi-Fi sdk IOT firmware upgrade application for matter OTA combined firmware application.

- In a separate terminal, locate the **chip-tool** and **ota-requestor**, and run the chip-tool commands to provision the Provider.

```
./out/chip-tool pairing onnetwork 1 20202021
./out/chip-tool accesscontrol write acl "[{"fabricIndex": 1, "privilege": 5, "authMode": 2, "subjects": [112233], "targets": null}, {"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": null, "targets": null}]' 1 0
```

- If the application device had been previously commissioned, hold Button 0 for six seconds to factory-reset the device.

- In the chip-tool terminal, commission the Device by passing below command.

```
./out/chip-tool pairing ble-wifi "node_id" "SSID" "PSK" 20202021 3840
```

where SSID and PSK are AP username and password.

- Once the commissioning process completes in the same terminal, run below requestor command to start downloading the image.

```
./out/chip-tool otasoftwareupdaterequestor announce-ota-provider 1 0 0 0 2 0
```

- The application device will connect to the Provider and start the image download. Once the image is downloaded, the device will reboot into the downloaded image.

Note: once image download is done, disconnect Jlink. it will reboot automatically with new image.

Matter OTA WiFi Project

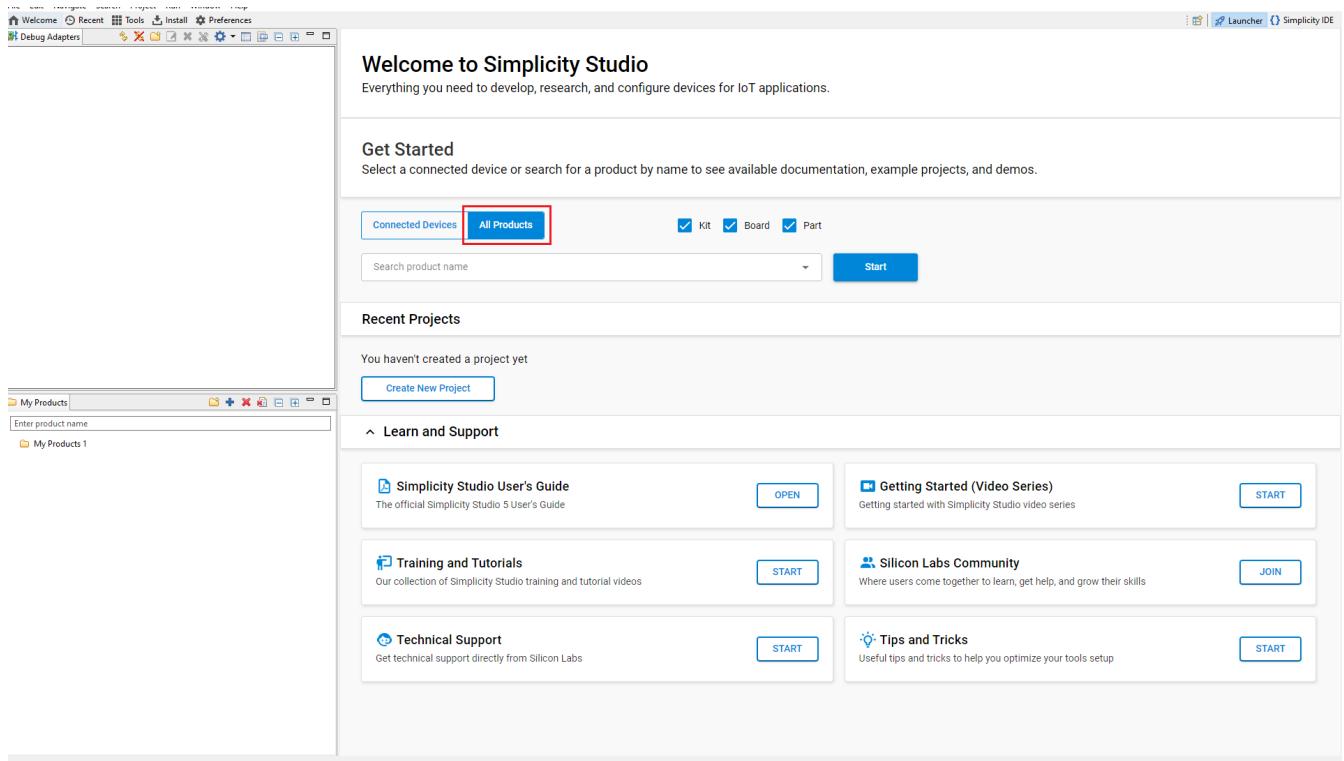
Building Matter applications for OTA Software Update

In Matter OTA Software Update scenario the running image (OTA-A) and the update image (OTA-B) are regular Matter application images and are built using the standard procedure, the only additional configuration required is the use of a higher software version in the update image. This document gives information about creation of OTA-A and OTA-B Application for EFR32 and 917 SOC Boards.

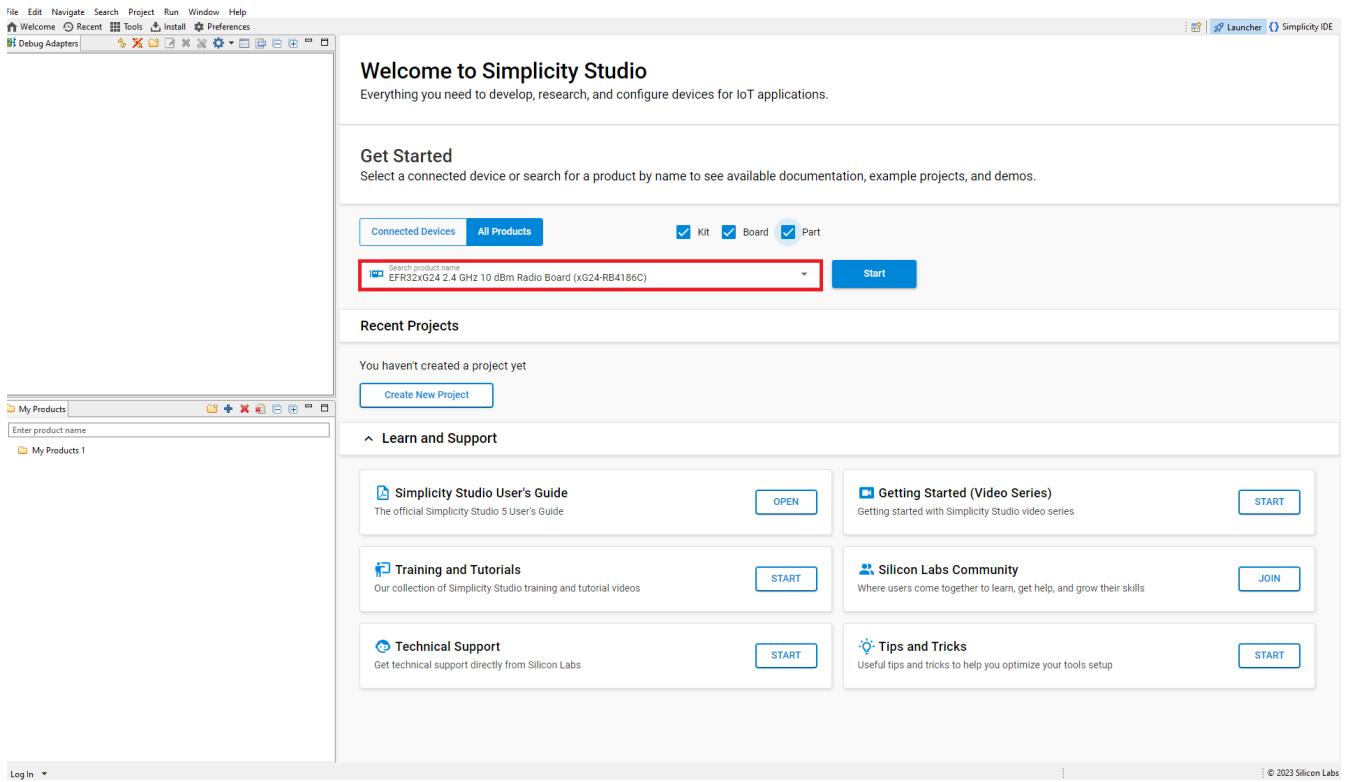
Note: Examples used in this document for EFR32. Select BRD4338A Board to create OTA-A and OTA-B application for 917 SOC.

Create and Build Project for Matter OTA-A Application

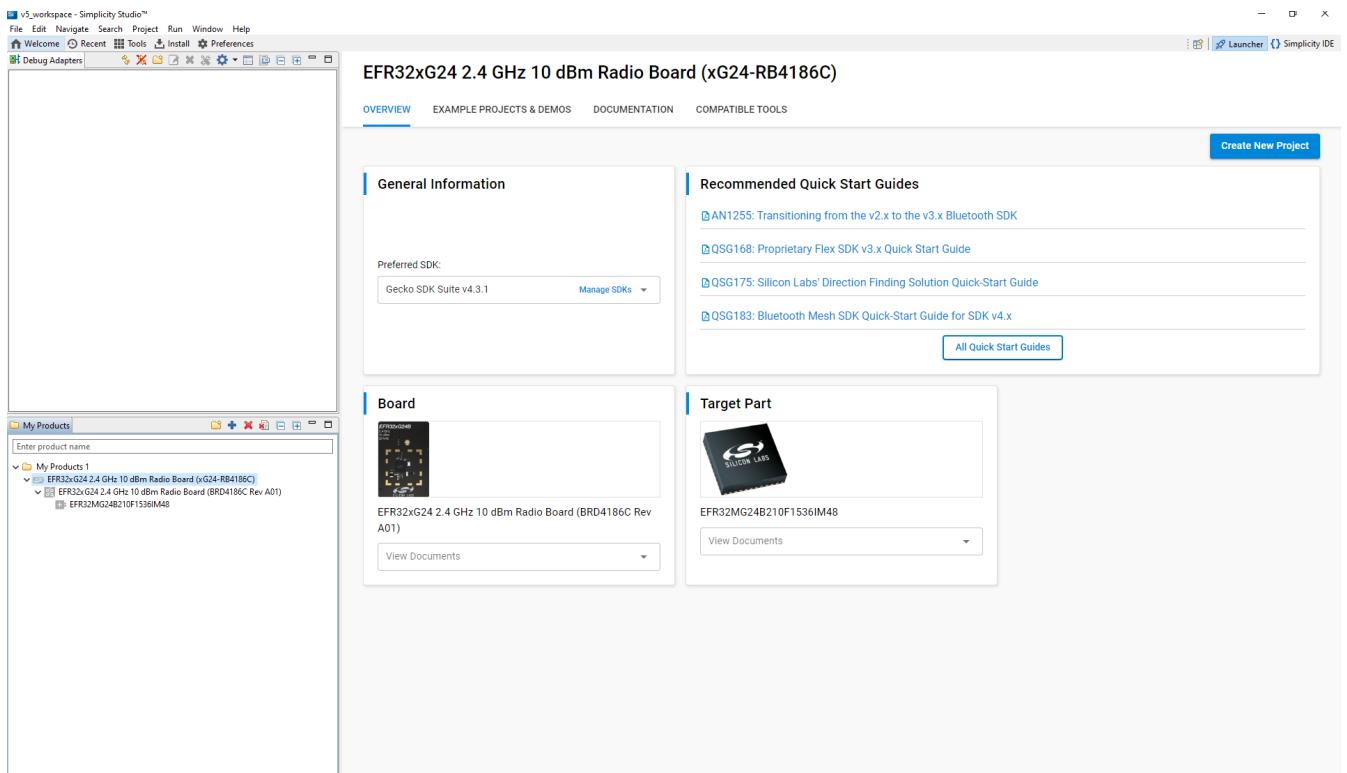
1. [Download](#) and Install Simplicity Studio.
2. To install the software packages for Simplicity Studio, refer [Software Package Installation](#)
3. Log in to Simplicity Studio and connect the board to the computer.
4. Go to the All Products section.



5. Type and Select the radio board from the displayed list and select Start.



6. The Launcher page will display the selected radio board's details.



7. Verify the following in the General Information section:

- o The Debug Mode is Onboard Device (MCU).
- o The Preferred SDK is the version you selected earlier.

File Edit Source Refactor Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences

Debug Adapters



> EFR32xG21 2.4 GHz 20 dBm RB (ID:440192814)

EFR32xG21 2.4 GHz 20 dBm RE

[OVERVIEW](#)[EXAMPLE PROJECTS & DEMOS](#)[DOCUMENTATION](#)

General Information

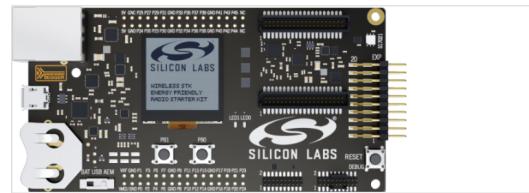
Connected Via: J-Link Silicon Labs

[Configure](#)Debug Mode: **Onboard Device (MCU)** Adapter FW: **1v4p6b1171** [Latest](#)Secure FW: **1.2.1** [Update to 1.2.11](#) | [Changelog](#)

Preferred SDK:

Gecko SDK Suite v4.3.1[Manage SDKs](#) ▾

Board



8. Click on Example Projects and Demos Option and Create Project.

EFR32xG24 2.4 GHz 10 dBm Radio Board (BRD4186C Rev A00)

OVERVIEW EXAMPLE PROJECTS & DEMOS DOCUMENTATION COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos

Example Projects

Solution Examples

What are Demo and Example Projects? [View Project Documentation](#)

Wireless Technology Clear

- Bluetooth (51)
- Bluetooth Mesh (20)
- Connect (9) 2
- Matter (45) 2
- RAIL (22)
- Thread (17)
- Zigbee (26)

Device Type Clear

- Host (1)
- NCP (0)
- RCP (0)
- Soc (30)

Ecosystem Clear

- Amazon (0)

MCU Clear

- 32-bit MCU (0)
- Bootloader (0)

Matter - SoC Light Switch over Wi-Fi
This project builds a Matter Light Switch for WI-FI WF200 that can be developed inside Simplicity Studio [CREATE](#)

Matter - SoC Lighting over Thread
This is a Matter Lighting Application over Thread for BRD4186C [RUN](#) [CREATE](#)

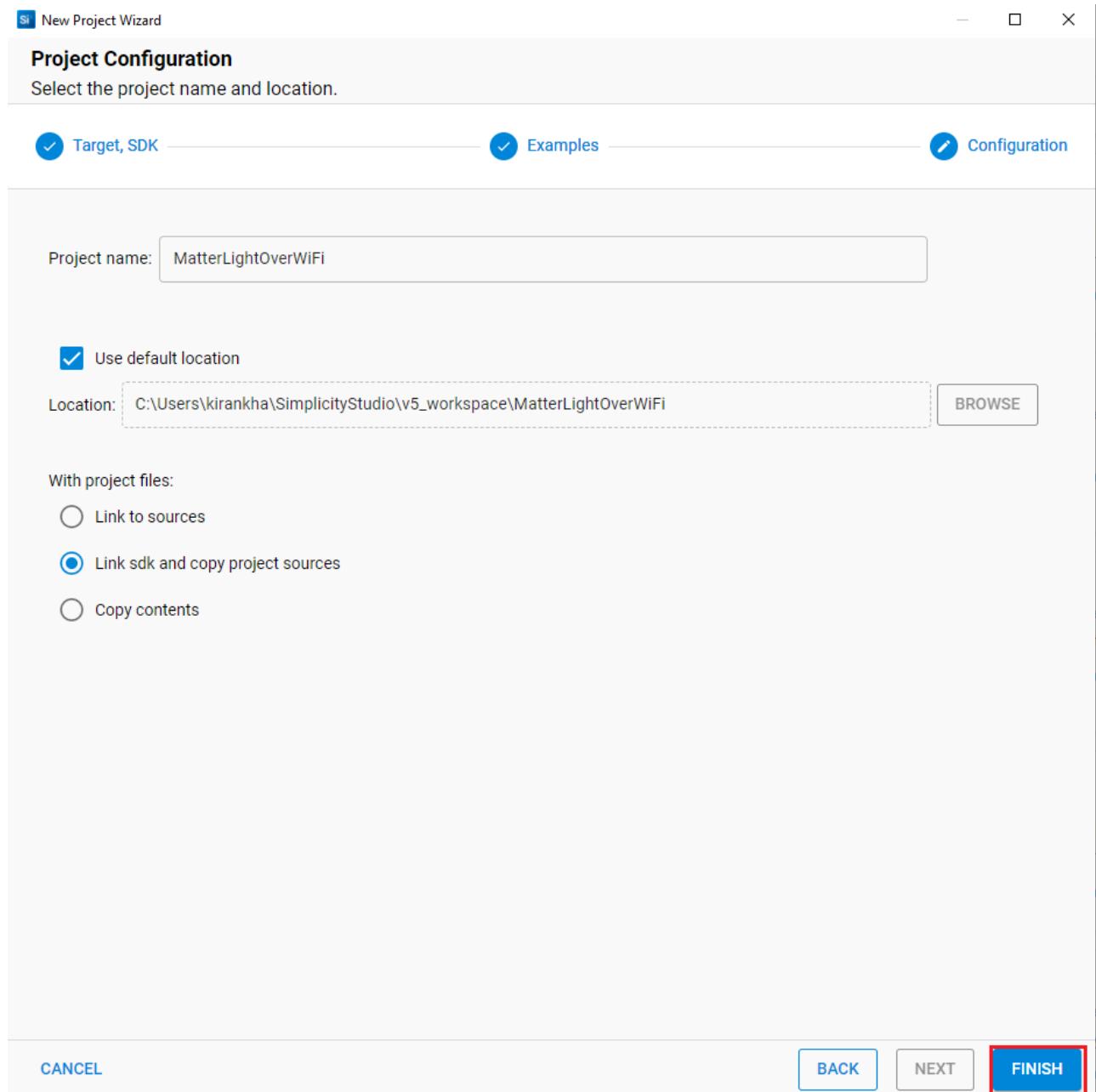
Matter - SoC Lighting over Wi-Fi
This project builds a Matter Lighting app for WI-FI WF200 that can be developed inside Simplicity Studio [CREATE](#) 3

Matter - SoC Lighting over Wi-Fi
This is a Matter Lighting Application for BRD4186C to be used with RS9116/917NCP WI-FI Evaluation kit [RUN](#) [CREATE](#)

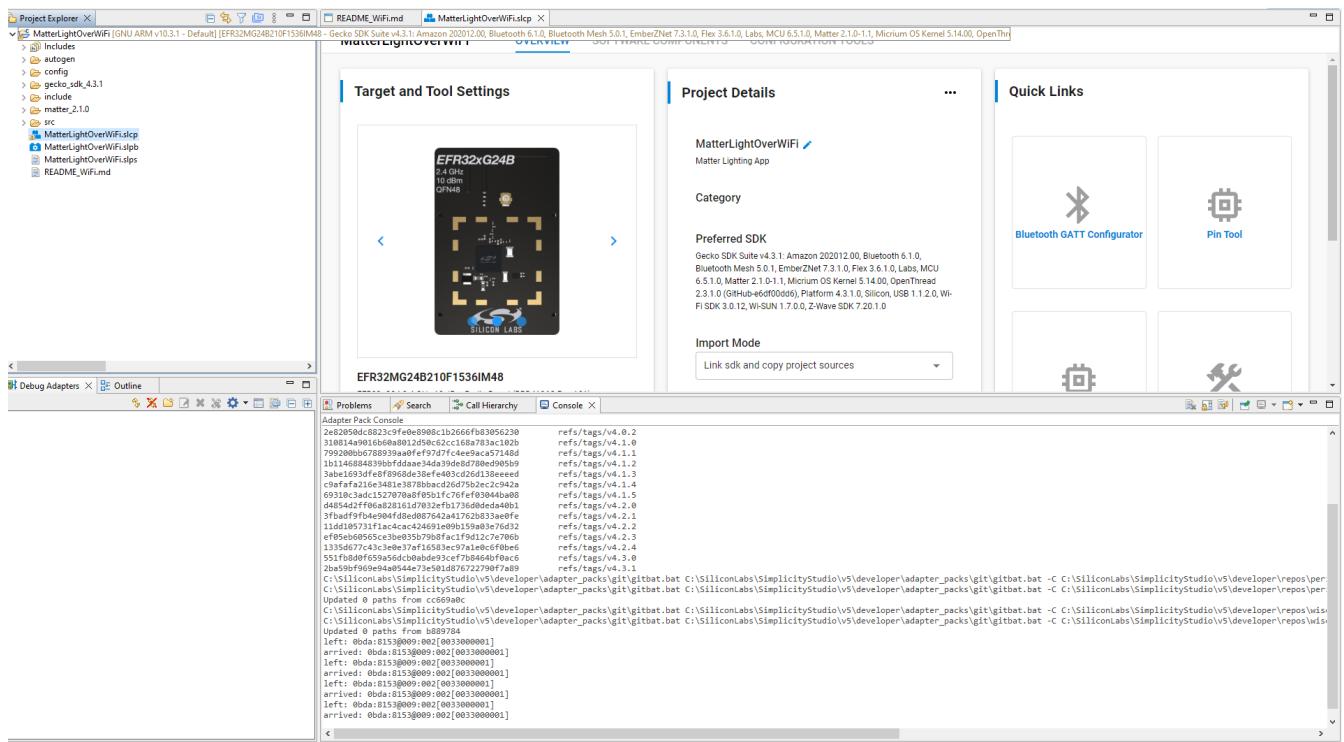
Matter - SoC Lock over Thread
This project builds a Matter Lock that can be developed inside Simplicity Studio [CREATE](#)

Matter - SoC Lock over Wi-Fi

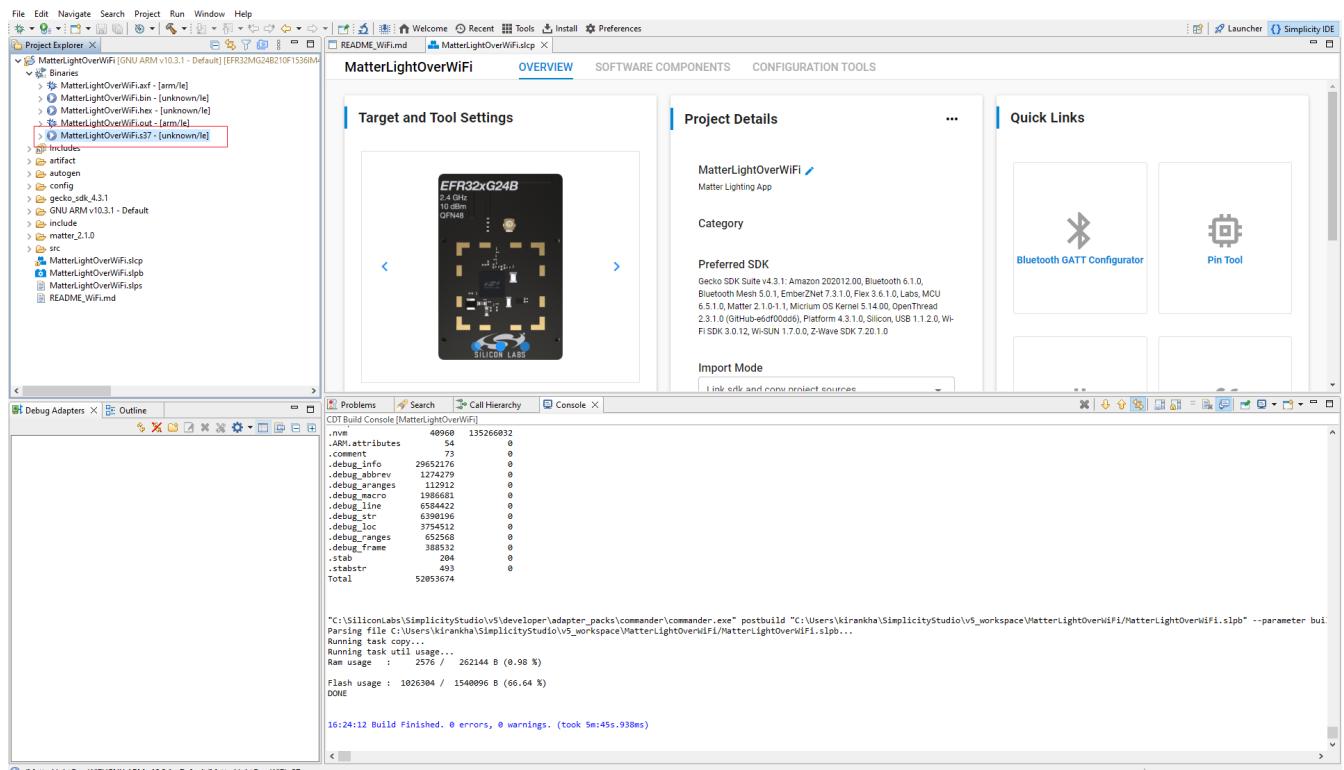
9. In the New Project Wizard window, click Finish.



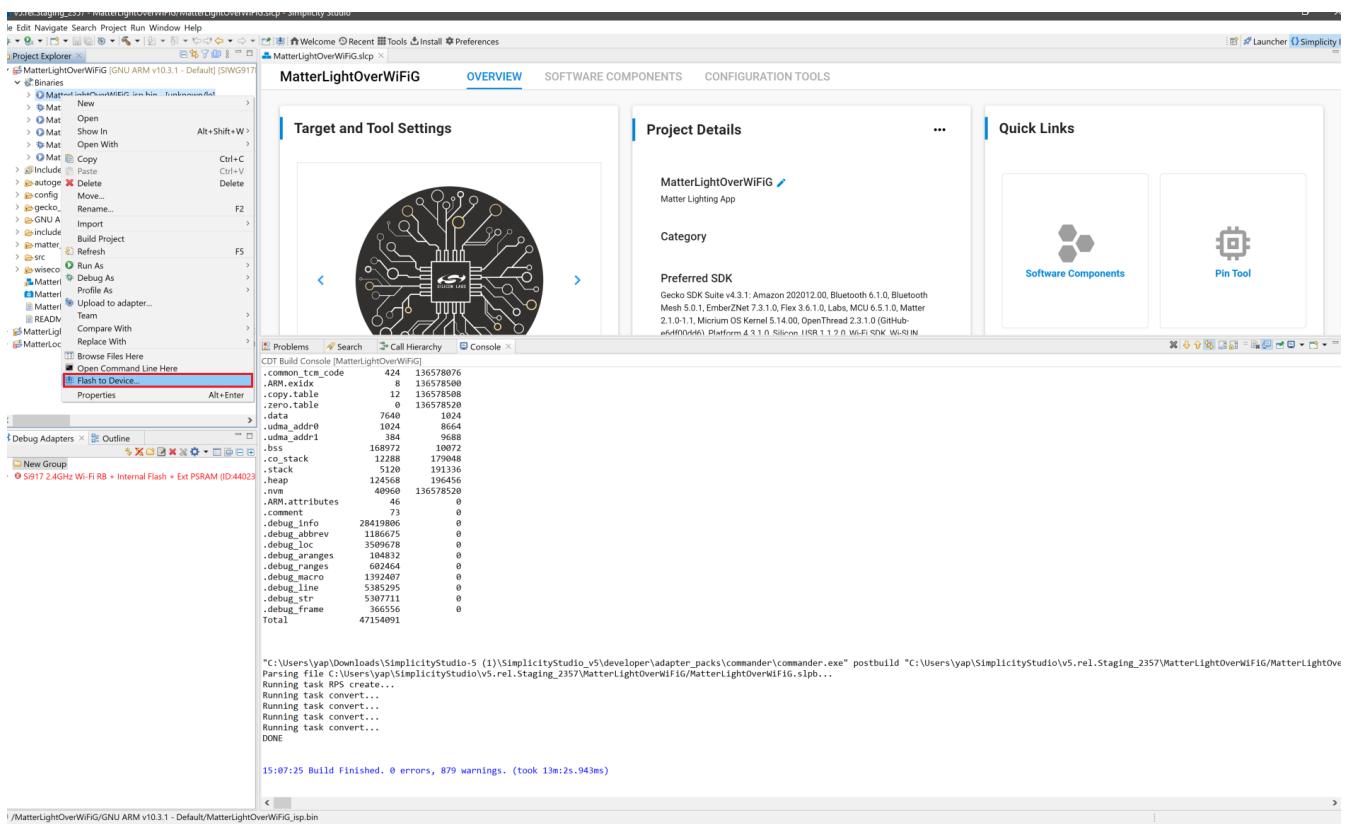
10. Once the project is created, right-click on the project and select *Build Project* in the Project Explorer tab.



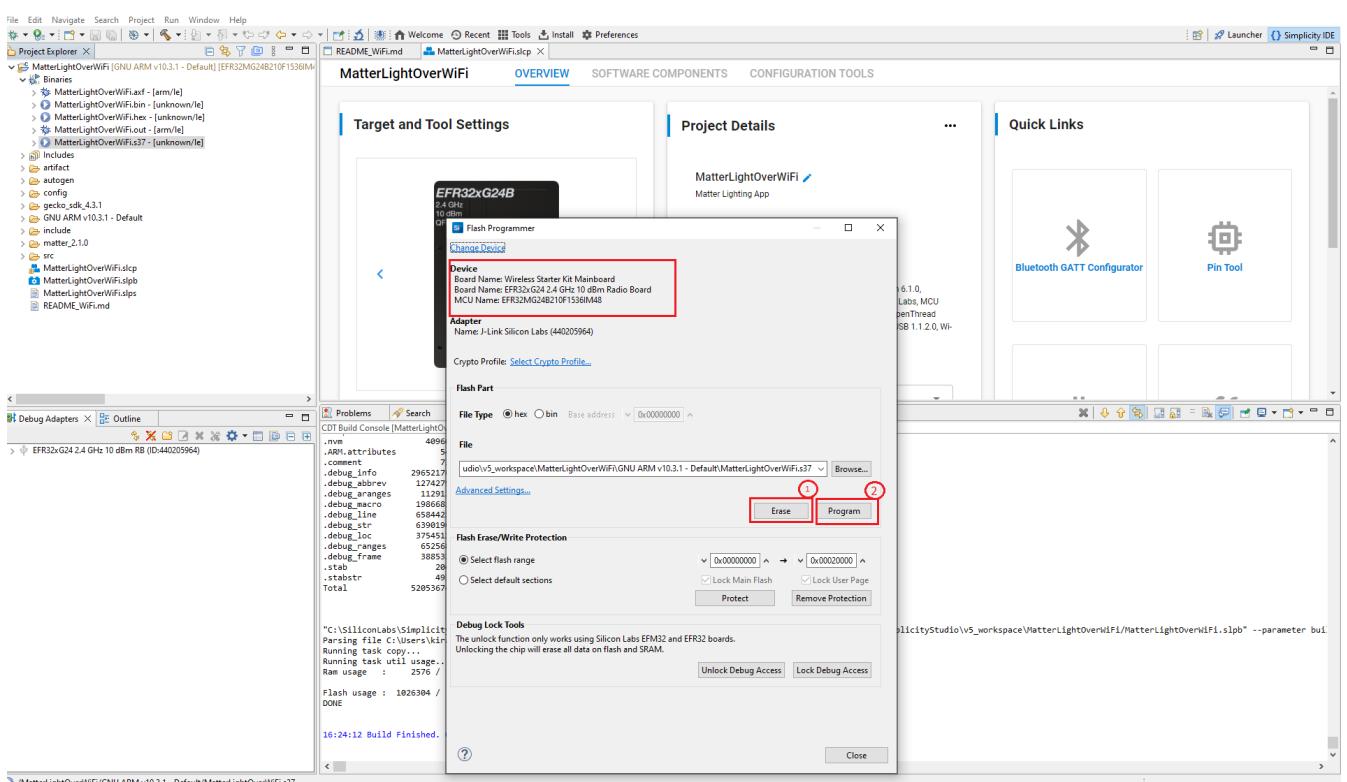
11. Once the project is compiled successfully, Go to the Project Explorer view and expand binaries folder to flash the binary.



12. Right-click on the selected .s37 binary and click on *flash to device*.



13. Flash programmer window will be opened, Click on *Erase button* and then *Program button* to start the flashing.

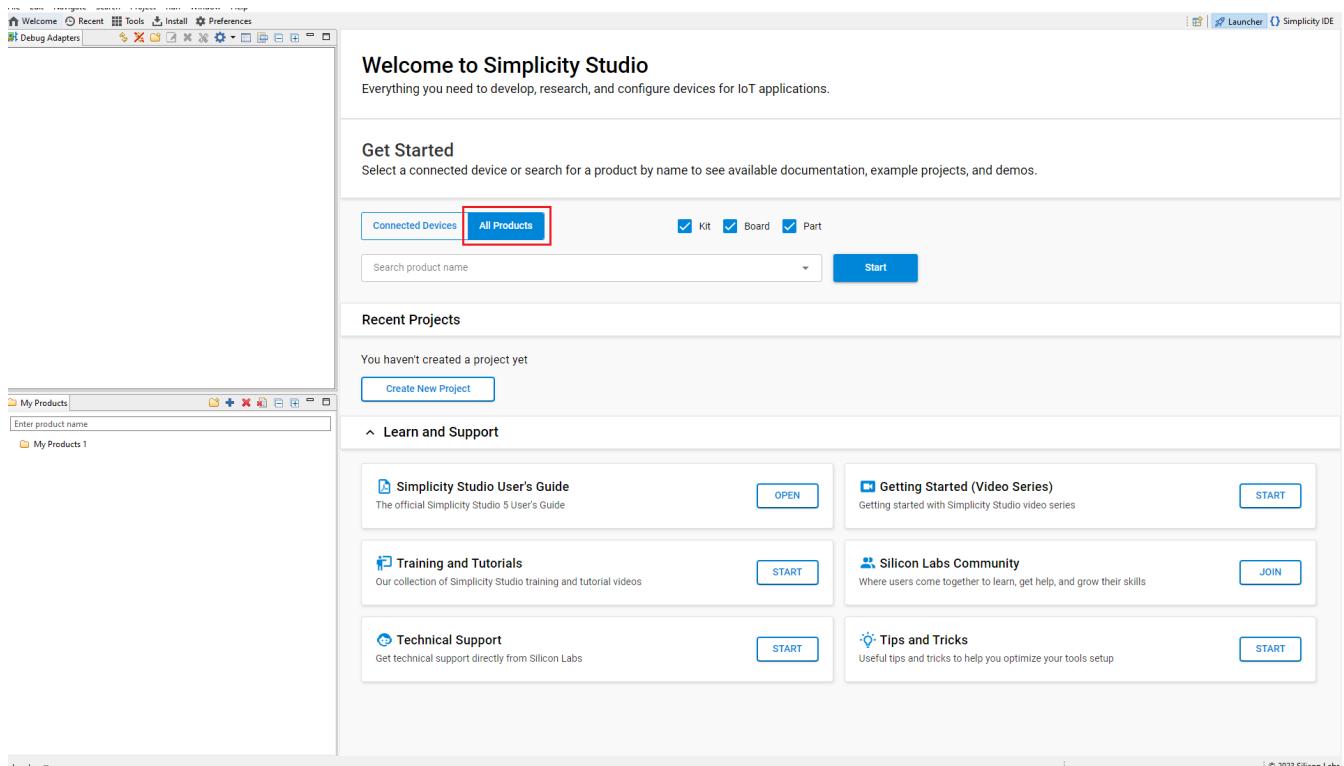


Note: Output of the EFR32 NCP Host application will be displayed on the J-Link RTT Viewer.

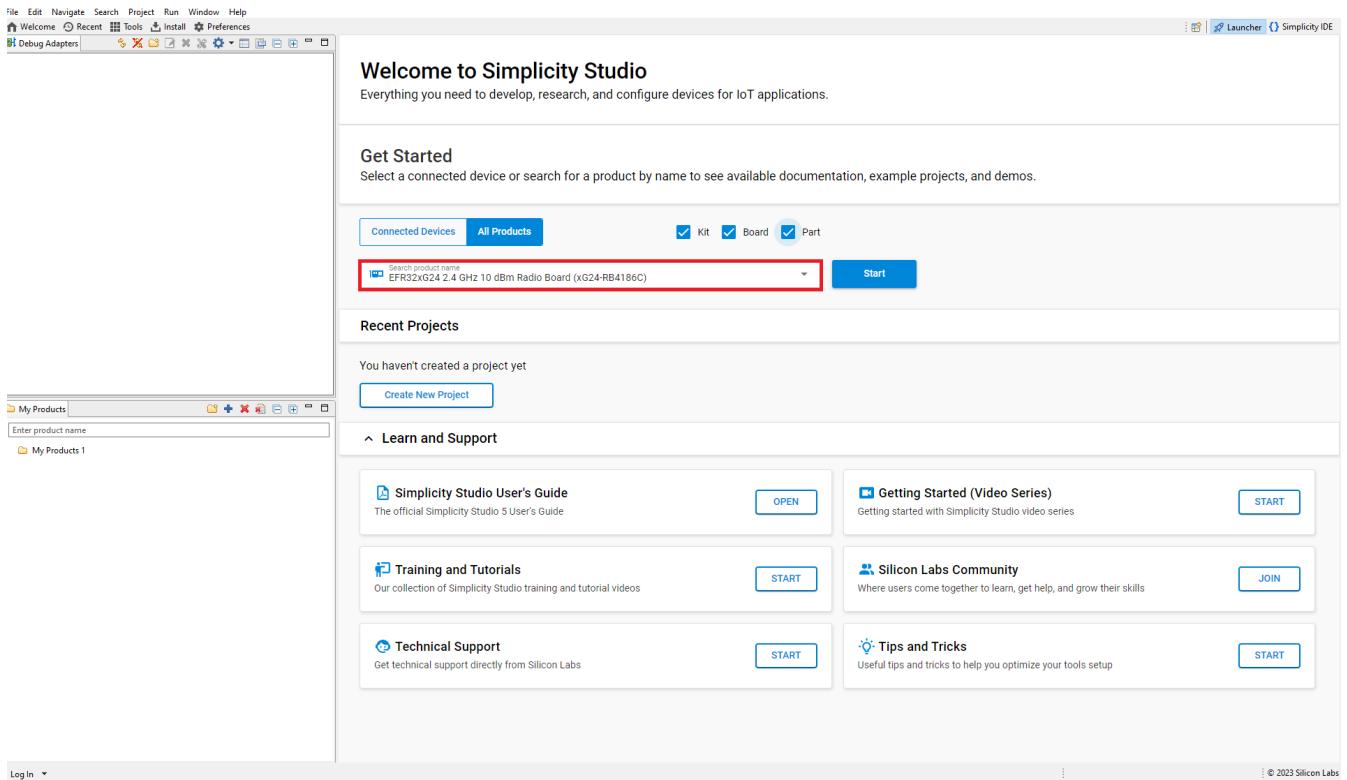
Create and build Project for matter OTA-B application

- Matter OTA-B application will be used to create gbl for EFR32MG2x & .rps for SiWx917 SOC OTA file and OTA-A will be used to flash to the matter device.
- For Matter OTA-B application need to change Version in sl_matter_config.h file before building.

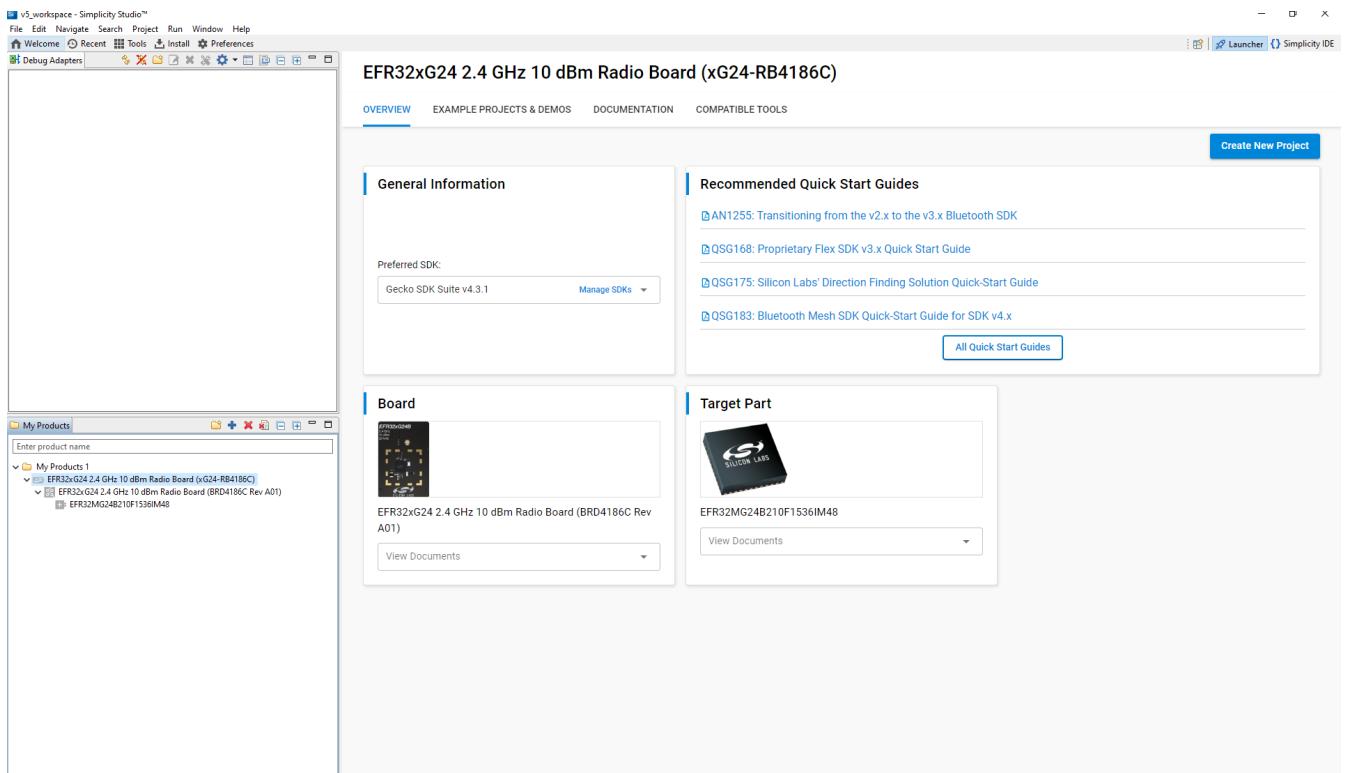
1. [Download](#) and Install Simplicity Studio.
2. To install the software packages for Simplicity Studio, refer [Software Package Installation](#)
3. Log in to Simplicity Studio and connect the board to the computer.
4. Go to the All Products section.



5. Type and Select the radio board from the displayed list and select Start.



6. The Launcher page will display the selected radio board's details.



7. Verify the following in the General Information section:

- o The Debug Mode is Onboard Device (MCU).
- o The Preferred SDK is the version you selected earlier.

File Edit Source Refactor Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences

Debug Adapters



> EFR32xG21 2.4 GHz 20 dBm RB (ID:440192814)

EFR32xG21 2.4 GHz 20 dBm RE

[OVERVIEW](#)[EXAMPLE PROJECTS & DEMOS](#)[DOCUMENTATION](#)

General Information

Connected Via: J-Link Silicon Labs

[Configure](#)Debug Mode: **Onboard Device (MCU)** Adapter FW: **1v4p6b1171** [Latest](#)Secure FW: **1.2.1** [Update to 1.2.11](#) | [Changelog](#)

Preferred SDK:

Gecko SDK Suite v4.3.1[Manage SDKs](#) ▾

Board



8. Click on Example Projects and Demos Option and Create Project.

EFR32xG24 2.4 GHz 10 dBm Radio Board (BRD4186C Rev A00)

OVERVIEW EXAMPLE PROJECTS & DEMOS DOCUMENTATION COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos

Example Projects

Solution Examples

What are Demo and Example Projects? [View Project Documentation](#)

Wireless Technology Clear

- Bluetooth (51)
- Bluetooth Mesh (20)
- Connect (9) 2
- Matter (45) 2
- RAIL (22)
- Thread (17)
- Zigbee (26)

Device Type Clear

- Host (1)
- NCP (0)
- RCP (0)
- Soc (30)

Ecosystem Clear

- Amazon (0)

MCU Clear

- 32-bit MCU (0)
- Bootloader (0)

Matter - SoC Light Switch over Wi-Fi
This project builds a Matter Light Switch for WI-FI WF200 that can be developed inside Simplicity Studio [CREATE](#)

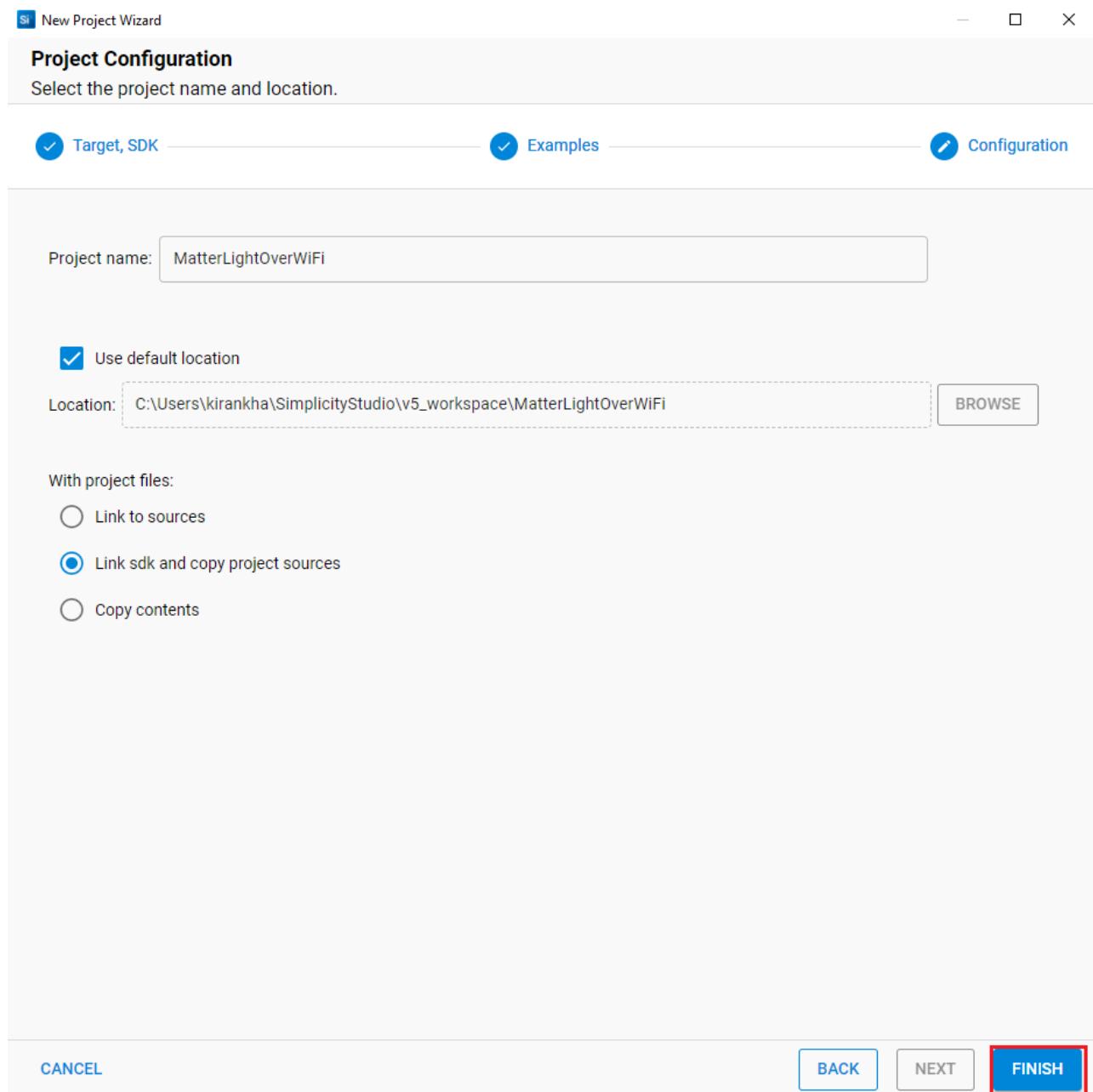
Matter - SoC Lighting over Thread
This is a Matter Lighting Application over Thread for BRD4186C [RUN](#) [CREATE](#)

Matter - SoC Lighting over Wi-Fi
This project builds a Matter Lighting app for WI-FI WF200 that can be developed inside Simplicity Studio [CREATE](#) 3

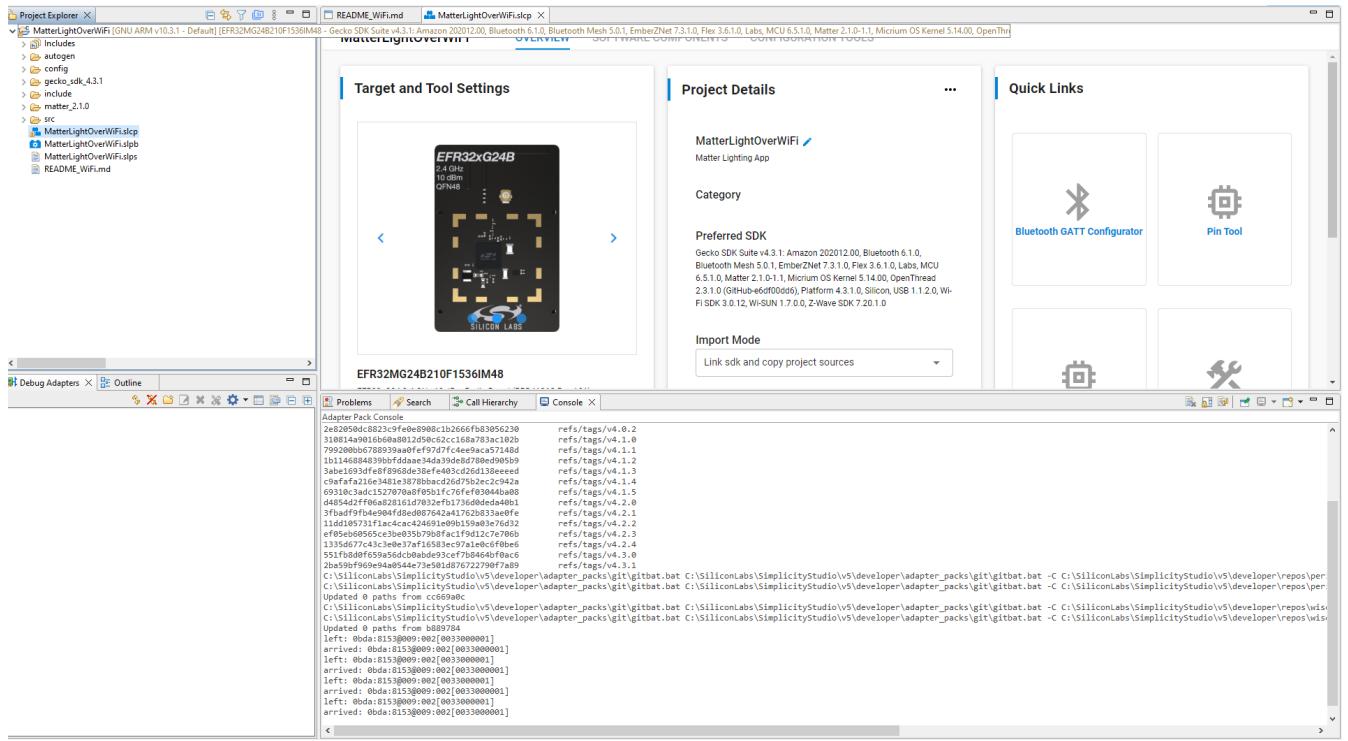
Matter - SoC Lock over Thread
This project builds a Matter Lock that can be developed inside Simplicity Studio [CREATE](#)

Matter - SoC Lock over Wi-Fi

9. In the New Project Wizard window, click Finish.



10. In Project Explorer section , open sl_matter_config file which is present in the config folder. Modify the **CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION 2** and **CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING "1"**
Note: Make sure always CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION should be greater than CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING
11. Once the modification is done for Software version, right-click on the project and select *Build Project* in the Project Explorer tab.



12. Once the project is compiled successfully, Go to the Project Explorer view and expand OTA-B project binaries folder , using application .s37 file Need to Create .gbl file using Simplicity commander.
13. After Creation of OTA-B Application run the OTA Scenario.

Introduction

Introduction

The Matter Production Guide includes:

- [**Device Development Prerequisites**](#) provides the prerequisites and next steps to facilitate your production journey through Matter.
- [**Custom Part Manufacturing Services**](#) describes the Matter support that Silicon Labs offers through our Custom Part Manufacturing Services (CPMS).
- [**Device Attestation**](#) describes this step in the Commissioning process for Matter devices to be commissioned into a Matter network.

Device Development Prerequisites

Matter Device Development Prerequisites

If you plan to develop a Matter end product, this page lists the prerequisites and next steps to facilitate your production journey through Matter.

Become a CSA Member

Your organization must be an associate member or better to get your product certified by a CSA-approved testing facility. As a member, your organization will receive membership perks:

- Official resources to assist you in developing Matter products.
- Authorization to contribute to the [Matter Github repository](#).
- 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](#) (DCL).
- [Matter Certification tool access](#)
 - Allows you to evaluate your product for certification before the official certification process.

Become a member at [CSA Membership](#). You can see a list of the different memberships offered at CSA.

Develop Your Matter Application

Creating a Matter Application is an exciting journey. To start your journey, you need to understand the [Matter Fundamentals - Silabs](#). We also provide two development paths that you can choose from. Refer to [Developing with Silicon Labs Matter](#) to help you choose these paths.

Once you are ready to develop your Matter Application, you should review the [Matter Developer's Guide](#), which provides detailed background and instructions for Matter developers working in either the Thread or Wi-Fi models. The Developer's Guide contains a deeper dive into development.

Additionally, Silicon Labs recommends that you become familiar with the Matter Specification documents:

- Matter Core Specification
- Matter Device Library Specification
- Matter Application Cluster Specification

These specifications can be downloaded from the [CSA's Specifications Download Request](#) website.

CSA Certification

Once a device/product is ready for production, you must have your product certified by a CSA-approved testing facility to certify compliance with the Matter Standard. Review the [CSA Certification Process](#) to ensure a smooth and prompt Matter product certification.

By becoming a member of the CSA, access to their [Matter Certification Tool](#) is granted. The Matter Certification Tool facilitates applying to certify your Matter product, after which you can create your application and upload your images and required documents. For additional resources on the certification process, refer to the [Certifying your Matter Product, an Overview](#) KBA. This KBA provides information on:

- Overview of Matter Certification Steps and milestones
- Thread Certification

- Matter Test Harness Guide
- Important Matter CSA Resources and a Q&A

Once your device is approved, your organization will be issued an official Certification Declaration (CD). This CD is a cryptographic document issued by CSA that confirms that your device has been certified. The CD is included in the attestation process sent by the Commissionee during device attestation. To view a list of authorized test facilities, check out the [CSA Authorized Testing Providers](#) summary. This CD is required to be injected during the manufacturing process if you are using the Silicon Labs CPMS process. This certification process can take some time to complete. You should ensure that this time is accounted for during your development lifecycle to keep your product on schedule.

Helpful CSA Links

- [CSA Membership](#)
- [CSA Specifications Download Request page](#)
- [CSA Pre-Certification Tool](#)
- [CSA Authorized Testing Providers](#)
- [CSA Distributed Compliance Ledger \(DCL\)](#)
- [CSA Test Distributed Compliance Ledger \(DCL\)](#)
- [PAA Providers](#)

Standards Development Organizations (SDO) Membership

Matter is an application layer that works on top of other proven network technologies. Your organization may also be required to be a member of an SDO and be able to pass the certification processes that these organizations require. Examples of these SDOs are Bluetooth, WiFi, Thread, and others. Be sure to take these organizational requirements into account when planning your products. Refer to the [Certifying your Matter Product, an Overview](#) KBA.

Ready for Production

Silicon Labs is the only IoT embedded solution provider offering a Custom Part Manufacturing Service (CPMS) to device makers. This secure provisioning service allows IoT device makers to order customized hardware straight from the factory via the [CPMS web portal](#). CPMS removes the numerous complexities, time, and expense of custom provisioning Matter devices at scale. To provide this solution, Silicon Labs has partnered with Kudelski Security to provide scalable access to Device Attestation Certificates (DACs) for Matter Devices.

When moving to production, if you decide to provision your devices at scale using CPMS and would like to learn more about Kudelski, refer to their [Matter-compliant certificate service](#). To use the Silicon Labs CPMS solution, contact [Kudelski](#) to create an account. When creating a Kudelski account and using CPMS for production, you will need to specify Silicon Labs as a requestor and recipient of DACs for any PAIs that will be programmed in the Silicon Labs manufacturing facilities.

Custom Part Manufacturing Services

Using Custom Part Manufacturing Services (CPMS)

Silicon Labs offers Matter support through our Custom Part Manufacturing Services (CPMS). Your organization can order your Matter devices directly from Silicon Labs or a third-party vendor utilizing our CPMS services. Silicon Labs is one of the few providers that can program your information directly to silicon through secure automation with our partner, Kudelski Security.

What is CPMS?

CPMS allows you to customize Silicon Labs hardware – wireless SoCs, modules, MCUs – at the factory. The CPMS self-service web portal guides you through the customization process and its various customizable features and settings. You can place orders for customized test and production units to our factories securely via the CPMS portal.

Unlike traditional flash programming, CPMS is a secure provisioning service that enables you to customize your chips with highly advanced features. These include secure boot, secure debug, encrypted OTA, public, private, and secret keys, secure identity certificates, and more. The custom features, identities, and certificates are injected into the hardware securely, quickly, and cost efficiently through Silicon Lab's own factories.

Why CPMS?

Securing an IoT device is a highly complicated and costly process. You must generate public and private keys for secure boot and secure debug, sign code with a private key, store all the private keys in a Hardware Security Module (HSM), place the public keys for secure boot and secure debug in one-time-programmable (OTP) memory, flip OTP bits for secure boot and secure debug, and flash the encrypted code and identity certificates within the hardware. CPMS streamlines the programming part of this process for you. Even the most advanced security features, certificates, and identities can be programmed in a secure, fast, and cost-efficient way in Silicon Lab's factories.

How Does Matter Fit into the CPMS Equation?

Silicon Labs is the only IoT-embedded solution provider at this time offering a secure provisioning service for Matter devices at scale. Silicon Labs has partnered with [Kudelski Security](#) to provide scalable access to Device Attestation Certificates (DACs) for your Matter devices. Kudelski has "30+ years of experience securely provisioning more than 500 million devices". Rest assured that your secrets are stored in HSMs both on and offline to provide maximum security for your secret key material. Learn more about [Security](#).

CPMS allows you to configure your device and receive production samples for a minimal cost before making a full production order. To configure your Matter settings, there are two ways to accomplish this with Silicon Labs tooling.

If your organization uses [Simplicity Studio](#), Silicon Lab's IoT IDE, we have provided a built-in utility that will output a JSON formatted data file that can be uploaded directly into CPMS. This data file will fill out the necessary Matter information for you. This is the preferred method as it reduces the potential for errors and/or typos.

The second method is to simply provide the required information through the CPMS web forms. This is a minimal process that includes important attestation information such as your Vendor ID (VID), Product ID (PID), Certification Declaration, and other inputs required to generate the Matter certificate chain.

CPMS has automated integrations with Kudelski to obtain the unique DACs for each device at the time of manufacturing. All data remains encrypted throughout the entire process through secure channels between Kudelski and Silicon Labs.

I'm Ready to Get my Product to Market. What is Needed by CPMS?

CPMS will ask for various attributes about your device, but these are the primary elements that will be needed for proper certificate generation.

- Vendor ID (VID) - Your unique VID will be required by CPMS to properly generate the necessary PKI infrastructure to allow your device on the Matter network.
- Product ID (PID) - Your organization will need to provide a unique PID that will be used to identify this product on the network.
- Certification Declaration (CD) - This is a cryptographic document that is issued to you by CSA after your device has been successfully certified by a CSA-approved testing facility.

Pre-Production Checklist

1. Choose a Matter-capable part to develop your Matter application on.
2. Become a [CSA member](#) if your organization is not already a member. An associate-level membership or higher is required to obtain membership perks, certification, and a Vendor ID. See [Device Development Prerequisites](#). If you have not been through these steps, please ensure ample time to get this step done before you are ready to go to production.
3. If you are already a CSA member, make sure that you have been supplied a VID from CSA. If not, contact CSA to obtain a VID. The VID should also have been added to the [CSA Distributed Compliance Ledger \(DCL\)](#).
4. Confirm that your VID has been added to the [DCL](#).
5. As a device maker and CSA member, you should add information about your device to the ledger before shipping your device to the market. If this is not available at the time of release, your devices will not attest properly.
6. Your application has been developed and is ready for certification.
7. Using the [CSA Pre-certification tool](#), you can test your application for completeness before submitting your application for certification. Save your organization time and money by pre-certifying your application before submitting it for certification.
8. Submit your application for certification to a [CSA-approved testing facility](#) for your product type. Once certified, you will be issued a [Certification Declaration \(CD\)](#). This is a cryptographic document stating that your device has successfully been certified and is used in conjunction with the Matter certificate chain to attest to the Matter network. This file should be in a .der format.
9. Begin the process of setting up an account with Kudelski Security as a provider of DACs. Note: Kudelski provides DACs on the Test DCL for no charge. [Learn more about our partnership](#) with Kudelski Security for Matter devices.
10. Ensure that you have the CD in hand. This will need to be uploaded to CPMS.
11. You're ready to order samples with [CPMS](#)!

Choosing the Test DCL or Production DCL

There are two public ledgers available to developers known as the Matter Distributed Compliance Ledger (DCL). The DCL is a cryptographically secure ledger based on blockchain technology. This ledger preserves an immutable record that stores public information that can be retrieved by DCL clients. For more details, see the [CSA Matter DCL whitepaper](#). Each DCL contains five schemas that can be accessed by a client to retrieve information about a device.

- Vendor Info Schema - this schema provides public information about the device vendor such as the VID, Vendor Name, and Company Legal Name.
- Device Model Schema - this schema provides public information about the actual device such as the Product Name, PID, VID, and more.
- Device Software Version Model Schema - this schema provides public information about software-specific data about the device such as Release Notes URL, OTA software image URL, and more.
- Compliance Schema - this schema provides public information about the certification of a device such as the VID, PID, Software Version, CD Certificate ID, and more.
- PAA Schema - this schema provides information about valid Product Attestation Authority certificates for approved PAAs.

The **Test DCL**, as the name suggests, is a public Matter ledger that will allow vendors to test their devices in a test environment. Entries into the Test DCL are less rigorous than the Production DCL and can be used to test devices using test certificates provided by Matter or other valid vendors. These test certificates cannot be used on the production DCL. For the production case, you have to ensure that you have the proper certificate chain in place. For CPMS, Kudelski provides Test DCL DACs at **no additional charge**. Your organization needs to ensure that an account has been created with Kudelski to order these DACs through CPMS. [Learn more here](#).

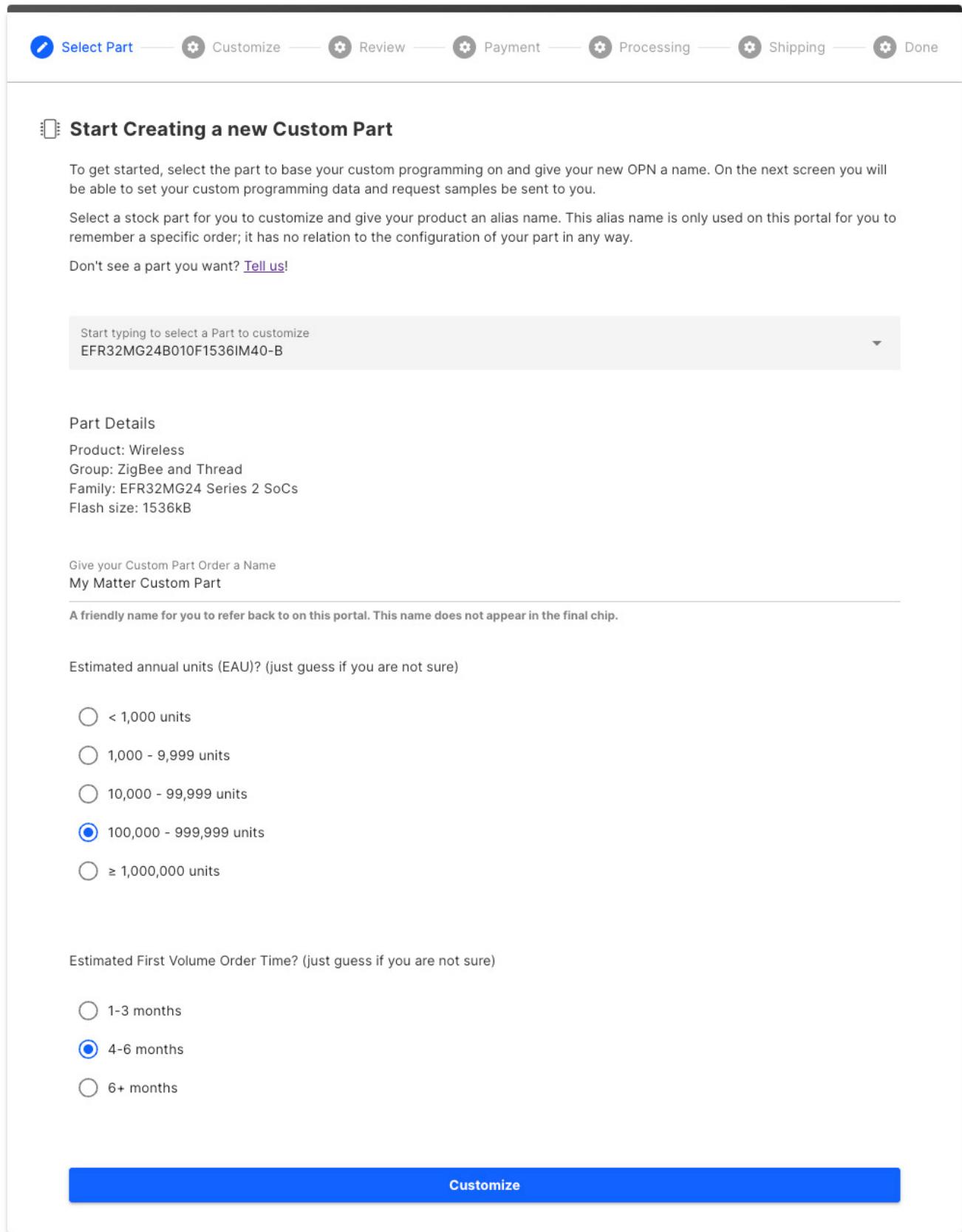
If you are ready to take your device to production, you have the option to select the **Production DCL**. This is the primary [Matter DCL](#) for production devices. For your device to properly commission onto the Matter fabric, the commissioner needs to be able to verify that a valid certificate chain is in place. The information needed must be publicly available in the

production DCL. The device needs to have a valid DAC signed by an approved PAI provider, and a root PAA provider. Your device also must contain a valid certification of the device, all available in the DCL. Silicon Labs partners with Kudelski Security as a PAA provider of choice. Kudelski also signs the Product Attestation Intermediate (PAI) certificate for our customers using CPMS. Each PAI is specific to our customer's products and is created when you set up a new product on your account with Kudelski. Production DCL samples must be approved even if you have already approved Test DCL samples before going to production.

CPMS Workflow

You've completed all of the items in the pre-production checklist and are ready to create samples. With CPMS, you get the benefit of receiving several actual samples of your product for your approval. This allows you to test the actual device before placing a large production run. Once you approve the sample, you have an Orderable Part Number (OPN) that can be used with Silicon Labs or other third-party distributors. The workflow involves the following steps:

1. To access CPMS, you need to register for an account with Silicon Labs. If you are using Simplicity Studio or other Silicon Labs tools, you probably already have this. If not, [register for a Silicon Labs account](#).
2. [Login to CPMS](#).
3. Create a new Custom Part.
4. Select the part on which you have built your Matter application. You will be asked a couple of questions about your future order. This helps Silicon Labs prepare for your eventual order and ensure that the factories are ready to go in the timeframe expected.



 Select Part  Customize  Review  Payment  Processing  Shipping  Done

Start Creating a new Custom Part

To get started, select the part to base your custom programming on and give your new OPN a name. On the next screen you will be able to set your custom programming data and request samples be sent to you.

Select a stock part for you to customize and give your product an alias name. This alias name is only used on this portal for you to remember a specific order; it has no relation to the configuration of your part in any way.

Don't see a part you want? [Tell us!](#)

Start typing to select a Part to customize
EFR32MG24B010F1536IM40-B

Part Details

Product: Wireless
Group: ZigBee and Thread
Family: EFR32MG24 Series 2 SoCs
Flash size: 1536kB

Give your Custom Part Order a Name
My Matter Custom Part

A friendly name for you to refer back to on this portal. This name does not appear in the final chip.

Estimated annual units (EAU)? (just guess if you are not sure)

< 1,000 units
 1,000 - 9,999 units
 10,000 - 99,999 units
 100,000 - 999,999 units
 ≥ 1,000,000 units

Estimated First Volume Order Time? (just guess if you are not sure)

1-3 months
 4-6 months
 6+ months

Customize

5. Click **Customize** to start configuring your device. With CPMS, you have a wide range of options to work with to customize your device. Matter is only one component of this. You have full control over other features of the part itself such as debug lock/unlock, secure boot, and many other security features depending on the part selected.

6. The Matter-specific configurations can be found in the Ecosystem Identities toggle. Select the toggle to view the available ecosystems supported by your device.

Ecosystem Identities

Add ecosystem identity configurations (i.e. Matter) to your custom part to enable it in production.

[Add Matter Ecosystem](#)

This product supports the Matter protocol. Configure your device information to add Matter to this custom part.

Flash Programming

Flash Programming involves the addition of customer specific code to a standard product. Customer code in INTEL HEX format is required.

7. Add the Matter Ecosystem to your part and you will be presented with the required Matter inputs to help secure the proper PAA/PAI/DAC certificates from Kudelski.
8. Upload your Certification Declaration. This is the file in .der format that you should have received after successful certification from a CSA-approved testing facility.

Certification Declaration * - required

Please provide a Certification Declaration (CD) in .der format. This could be an official CSA CD, a development CD provided by the Silicon Labs GSDK Matter provisioning tool, or other document which is created in the proper format.

 DRAG AND DROP OR CLICK HERE TO UPLOAD A FILE

9. (optional) If you used Simplicity Studio, use the Provisioning Tool to output your Matter information directly from the application. This tool outputs a cpms.json file that can be uploaded to help you quickly fill out this information.

Have a CPMS configuration file?

Did you know that you can generate a Matter / CPMS configuration file using provision.py from the GSDK? This will autofill the Matter inputs based on your development configuration. You may also remove this configuration once uploaded if you need to make changes. This will reset the inputs back to normal so that you can make manual edits.

 DRAG AND DROP OR CLICK HERE TO UPLOAD A FILE

10. Fill out the required Matter fields. This includes the VID, PID, and several additional inputs to help Silicon Labs generate the necessary Matter certificate chain. If you use the cpms.json file that is generated through the Silicon Labs Matter provisioning tool, these will be automatically filled in for you.

Matter Required Fields * - required**VID / Vendor ID ***

A 16-bit number that uniquely identifies a particular product manufacturer, vendor, or group thereof. Each Vendor ID is statically allocated by the Connectivity Standards Alliance. (eg. 0x1234)

PID / Product ID *

A 16-bit number that uniquely identifies a product of a vendor. (eg. 0x5678)

Hardware Version *

A 16-bit unsigned number that specify the version number of the hardware of the Node. The meaning of its value, and the versioning scheme, are vendor defined.

Hardware Version String *

A maximum of 64 characters that specify the version number of the hardware of the Node. The meaning of its value, and the versioning scheme, are vendor defined. It is used to provide a more user-friendly value than that represented by the Hardware Version attribute.

Key ID *

PSA Crypto Key Identifier. Range from 0x00000001 to 0x3FFFFFF. (eg. 0x00000002)

11. (optional) Fill out the Matter Optional Fields. These fields will also be automatically filled out for you if you use the cpms.json file referenced above.

Matter Optional Fields

^

Vendor Name

A maximum of 32 characters that specify a human readable (displayable) name of the vendor for the Node.

Product Name

A maximum of 32 characters that specify a human readable (displayable) name of the model for the Node such as the model number (or other identifier) assigned by the vendor.

Product Label

A maximum of 64 characters that specify a vendor specific human readable (displayable) product label. The ProductLabel attribute may be used to provide a more user-friendly value than that represented by the Product Name attribute. The Product Label attribute SHOULD NOT include the name of the vendor as defined within the Vendor Name attribute.

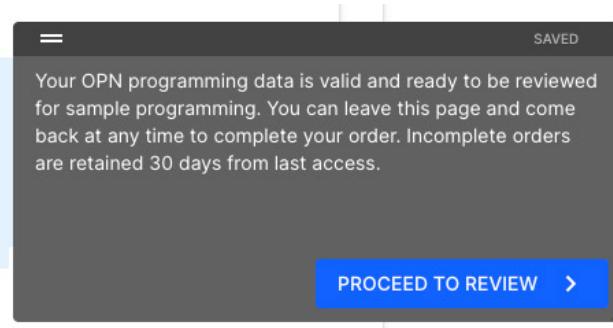
Product URL

A maximum of 256 characters that specify a link to a product specific web page. The syntax of the ProductURL attribute shall follow the syntax as specified in RFC 3986 [<https://tools.ietf.org/html/rfc3986>]. The specified URL should resolve to a maintained web page available for the lifetime of the product.

Part Number

A maximum of 32 characters that specify a human-readable (displayable) vendor assigned part number for the Node whose meaning and numbering scheme is vendor defined.

12. Once you have satisfied all of the required fields, you will be prompted to **Proceed to Review** to review the selections in your order.



13. Review your customizations and pricing information. You may also be asked for the shipping information if this is not on file with us already. The sample orders will be shipped to this address.
14. Submit for evaluation.
15. For Matter-specific parts, Silicon Labs works with Kudelski IoT to secure the DACs for your sample parts. These DACs are signed with the proper PAA/PAI certificate chains and delivered via Secure Vault Services integrations directly with Kudelski.
16. Once the DACs are available, the order will go into Silicon Labs manufacturing to be programmed and shipped to your address once the samples are complete.
17. You can then Approve or Reject the samples once your organization is able to test the sample parts. Silicon Labs recommends at this time that you test these samples with your device commissioner to ensure that the samples can properly attest to the Matter network.

18. Once approved, you will be able to order these parts, based on the OPN for that part. You can do this through Silicon Labs or through a third-party distributor. You may also opt to work with a Silicon Labs Field Application Engineer to help get this order executed.

Kudelski Security

Matter, CPMS, and Kudelski

Kudelski PKI Infrastructure

Silicon Labs has partnered with Kudelski Security to generate Matter Device Attestation Certificates (DAC) for our Custom Part Manufacturing Service (CPMS). "Kudelski IoT's Matter Product Attestation Certificate Service enables companies to get scalable access to Device Attestation Certificates, allowing each device to join the Matter ecosystem with confidence and ease". Kudelski is a certified CSA certificate authority that forms the root Product Attestation Authority (PAA) certificate in the CPMS Matter certificate chain. Each PAA owns a public and private key along with their self-signed PAA certificate which will be used to sign PAI certificates. Kudelski also manages the Product Attestation Intermediate (PAI) keys for CPMS used to sign the unique Device Attestation Certificates (DAC). PAIs are submitted in the form of a CSR request to the PAA to be signed. These three items, the PAA, PAI, and DAC as well as a Certification Declaration form the basis for a device to attest to the Matter network.

What is Required of My Organization with the Silicon Labs/Kudelski Security Partnership?

To have DACs generated by Kudelski, your organization will need to have an account created with Kudelski Security. Start at [Kudelski](#) to begin the process of creating this account.

After creating your account, your organization will want to create a Product Attestation Intermediary (PAI) certificate with Kudelski. This will be used to sign the DACs that will end up in your devices at manufacturing. If you use CPMS, you will need to specify Silicon Labs as a requestor and recipient of DACs for any PAIs that will be programmed in the Silicon Labs manufacturing facilities. This will allow CPMS Secure Vault Services to request those DACs on your behalf when you are ready to manufacture those products. Depending on your own internal processes, this may require reviews from within your own organization. Be sure to allow enough lead time for yourself when planning your Matter devices to account for this.

Why Not Just Do This Myself?

Kudelski IoT has done the heavy lifting for you. Kudelski is a well-established Public Key Infrastructure (PKI) provider that offers Matter certificates that allow you to be a part of the Matter ecosystem. Kudelski IoT uses its keySTREAM PKI-as-a-Service application to help manage your Matter PKI artifacts.

- Kudelski has done the heavy lifting of becoming a Connectivity Standards Alliance (CSA) authorized PAA for you.
- Kudelski has been creating device credentials for more than 30 years and is a trusted leader in PKI-as-a-Service.
- Cloud setup of your vendor-specific Product Attestation Intermediate (PAI)
- Managed generation and secure delivery of Matter Device Attestation Certificates (DAC)
- Kudelski and Silicon Labs have created an ultra-secure integration straight from the root CA to the factory.
- Spare yourself time, and get to market sooner with CPMS. CPMS already handles this for you!

Becoming your own PAA and issuing your own signed attestation certificates for Matter is no small undertaking. CSA requires very strict requirements for becoming a trusted certificate authority, as well as rigorous security and privacy requirements, hardware, infrastructure, governance, and much more. This is a costly endeavor that may include specialized hardware, such as on and offline Hardware Security Modules (HSM), specialized facility and operational needs, and much more that can quickly be out of reach of all but the largest of organizations, not to mention the time required to put these processes in place. In most cases, partnering with an established provider is far more economical for vendors and will get you up and running as quickly as possible.

What Should I Expect When I Create an Account with Kudelski?

There are a few processes in place to get you set up and working quickly to start receiving signed DACs from Kudelski:

1. Account Setup - If you are already a CSA member, you can start the setup of your account (fees may apply) on Kudelski's cloud application. If you are not a CSA member yet, review the CSA requirements. Per CSA requirements, Kudelski will conduct some manual background checks. Your organization will also need to sign the Kudelski Requestor Agreement Document (RAD) document before any PAIs are created.
2. New Product Family Setup - Once your Product Family is successfully certified by CSA, Kudelski can create your vendor-specific Product Attestation Intermediate (PAI) on keySTREAM for that Product Family.
3. Certificate Request - Once a PAI is created, you can request a batch of certificates for your devices. For the CPMS workflow, you will need to agree (with Kudelski) to allow Silicon Labs to be a requestor and receiver of DACs in order for the CPMS integrations to successfully request and receive those DACs during manufacturing.
4. Certificate Delivery - In the CPMS workflow, Silicon Labs will receive delivery of these DACs through secure services with Kudelski, which can then be programmed into your devices on the manufacturing line.
5. DACs are billed after your devices are manufactured and shipped.

Device Attestation

Matter Device Attestation

Matter Certificate Overview

For Matter devices to be commissioned into a Matter network, a Matter commissioner must verify that the devices are certified by the Connectivity Standards Alliance, this step in the Commissioning process is called Device Attestation. Each certified device must be configured with a unique Device Attestation Certificate (DAC) and its corresponding DAC private key, which will be checked by a commissioner to add this device to its Matter fabric. For a more conceptual overview of the Matter Certificates and Device Attestation Procedure, refer to [Matter Security](#).

Device Attestation Public Key Infrastructure and Certification Declaration

Device Attestation Certificates or DACs must be included in all commissionable Matter products and must be unique in each product. DACs are immutable, so they must be installed in-factory and must be issued by a Product Attestation Intermediate (PAI) which chains directly to a Product Attestation Authority (PAA), issued by specified root Certification Authorities (CAs). These root CAs are entities that have been approved by the CSA to issue digital Matter Certificates. Therefore, if you decide not to apply to become a Certification Authority, you will need to request the generation of the Matter Certificate Chain which is a Public Key Infrastructure. To request these certificates, you must meet the following requirements:

- Certify your Matter Product. CSA will issue a CD with a corresponding VID and PID.
- Select a Certification Authority where you will request your DACs. At Silicon Labs, we have partnered with Kudelski to offer the [Custom Manufacturing Service](#) to facilitate this process.

The PAA are root certificates (certificates of a Root Certificate Authority) and are used to sign PAIs (intermediate certificates). For the attestation process to succeed, the certificate chain must start from a trusted Root Certificate; for this purpose, Matter has a database, called [Distributed Compliance Ledger \(DCL\)](#), where the PAAs will reside.

Note: PAAs are not stored on the target devices.

The PAIs are intermediate certificates, signed by the PAA's private key, and are used to sign the DACs (device certificates). The PAI is stored on the target device and is sent to the Commissioner during the attestation process.

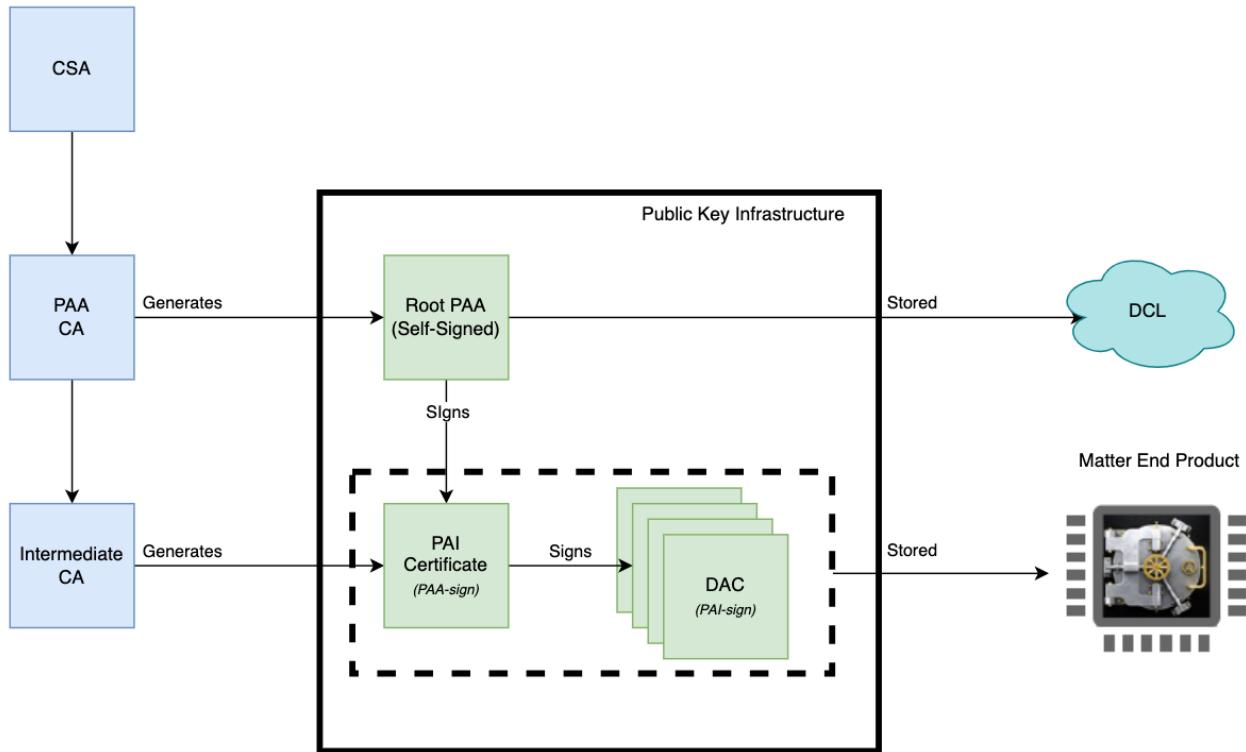
The DAC is the certificate that uniquely identifies the device itself. It is signed by the PAI, and like the PAI itself, it is sent to the Commissioner during the attestation process. The DAC public-key must match the device's private-key, which should be stored in the most secure location possible and is used to sign outgoing messages during commissioning.

The CD (Certification Declaration) is a file issued by CSA upon the firmware's certification process. It contains the Vendor ID (VID), and a list of Product IDs (PIDs), which should match the VID, and PID stored in the Subject field of both the PID, and DAC certificates. Along with the PID and DAC, the CD is stored on the device and sent to the Commissioner during the commissioning process.

Certification Authorities and Recommended Certificate Use

The Certification Authority's certificate chain is used to validate any certificate said authority has signed and confirm the Matter Device Identity. A Root Certification Authority is a CA whose certificates are self-signed. They are the root of trust. These are the most trusted and secured CAs, and their private keys are expected to be the most highly secured keys. Root CAs can generate the whole Certificate Chain (PAA > PAI > DAC). Intermediate Certificate Authorities are CAs whose certificates have been signed by a higher-level CA from which you can generate DACs that will be signed by these Intermediate CAs and saved on Matter Devices. These CAs will generate (PAI-DAC). A compromise in the private key of the

Root CA would jeopardize not only their issued Root Certificates, but also all the certificates in that chain including PAIs and DACs.



While the information in the certificate location is public, write access to the certificate location should be restricted. Usually the certificates are installed in-factory, but the exact procedure depends on the CAs involved, and the mechanisms available to secure the DAC private-key, and the certificate files. Ideally:

1. The device must run a small application in privilege mode that generates the key-pair in a secured environment. This procedure should run on-device.
2. The device issues a CSR, signed with its new private key, that is sent to the CA.
3. The CA issues the new certificate from the certificate Chain and signs it using its own PAI private key.
4. The newly created DAC is returned to the device.
5. The device stores the new DAC in a read-only section of non-volatile memory.
6. The device boots into normal operation mode, thus forbidding any further modification of either the DAC key-pair, and the DAC.
7. From this point on, the DAC private-key value should never be exposed, and the DAC should never be modified, unless it is compromised.

However, in some environments, the key-pair is generated outside the device. A CA may generate the DAC private keys on behalf of the device and use the keys to generate the DAC too. In this case, both the DAC key-pair, and the DAC are stored into the device on secured locations, and any external copy of the DAC private-key must be destroyed.

Device Attestation Procedure

When commissioning a device, the Commissioner must execute this procedure to validate if the device is indeed Matter certified and compliant, and if it has been produced by a certified manufacturer. The procedure is as follows:

1. The commissioner generates a random 32-byte nonce meant for attestation.
2. The commissioner sends the generated nonce to the commissionee (Matter Device being attested) and makes an Attestation Request.

3. The commissionee responds with an Attestation Response including an Attestation Information message signed with the DAC private key. This Attestation Information must be validated by the commissioner to accept the Matter Device into its Matter Fabric (Network).
4. The commissioner will retrieve the Matter Certificate Chain in order to attest the Matter Device.

Attestation Information to validate:

- The commissioner validates the PAA retrieved is trusted. Generally, the trusted PAA certificates are included in the Distributed Compliance Ledger (DCL).
- The device presents a valid CD, PAI, and DAC. This is determined by verifying the Certificate Chain.
- The VIDs and PIDs on the CD, PAI, and DAC must match. If the VID is included in the PAA, it must also match the other certificates.
- DAC must be valid, and signed by the PAI.
- The PAI must be valid, and signed by a trusted PAA.
- The Discriminator stored on the device must match the one searched by the Commissioner.
- The Passcode provided by the Commissioner must match the one stored on the device.
- The Attestation nonce from the device must match the Commissioners provided nonce.

Matter API Reference

- [DataModel](#)
- [Attributes](#)
- [Clusters](#)
- [Commands](#)
- [Events](#)
- [Cluster Implementation](#)

DataModel

Data Models

This has high level APIs for Data Models.

Header File

- [DataModelTypes](#)

Attributes

Attribute

This has high level APIs for Attributes.

Header File

- [Attribute](#)

Clusters

Cluster

This has high level APIs for Clusters.

Header File

- [Cluster](#)

Commands

Commands

This has high level APIs for Commands.

Header File

- [Commands](#)

Events

Event

This has high level APIs for Event.

Header File

- [Event](#)

Cluster Implementation

Cluster Implementation on Matter Core

This has the high level implementation of all the clusters present in Matter.

References

[Cluster Implementation](#)

Reference Guides

References

The reference pages provide detailed information and instructions on these topics.

- [Matter Commit Hashes](#)
- [How to Flash a Silicon Labs Device](#)
- [How to Find Your Raspberry Pi](#)
- [Using Development Tools in Simplicity Studio](#)
- [Building a Custom Matter Device](#)
- [Building a Multi-Endpoint Device](#)
- [Using ZAP, the ZCL Advanced Platform](#)
- [Using Wireshark with Matter](#)
- [Matter EFR32 Flash Savings Guide](#)

Matter Commit Hashes

Matter Repositories and Commit Hashes

The following repositories, branches and commit hashes are to be used together in this release of the Silicon Labs Matter Out of Box Experience

Open Thread Border Router (OTBR)

Repo	Branch	Commit Hash
https://github.com/SiliconLabs/ot-br-posix	main	1813352247aa60fb8993773918f1e5b4af6f3b79

Radio Co-Processor (RCP)

Repo	Branch	Commit Hash
https://github.com/SiliconLabs/ot-efr32	main	7a567da02a078546eb34136c1c44170c8832dd55

Connectivity Standards Alliance (CSA) connectedhomeip (Matter)

Repo	Branch	Commit Hash
https://github.com/project-chip/connectedhomeip	v1.1-branch	8f66f4215bc0708efc8cc73bda80620e67d8955f

Matter chip-tool

Repo	Branch	Commit Hash
https://github.com/SiliconLabs/matter	<this branch>	<this commit>

Matter Accessory Device (MAD)

Repo	Branch	Commit Hash
https://github.com/SiliconLabs/matter	<this branch>	<this commit>

How to Flash a Silicon Labs Device

How to Flash a Silicon Labs Device

Once you have an image built, you can flash it onto your EFR or SiWx917 device (either a development board or the Thunderboard Sense 2) over USB connected to your development machine. This can be done using either Simplicity Studio or the standalone Simplicity Commander.

Simplicity Studio

Simplicity Studio is a complete development environment and tool suite. It has the ability to discover USB-connected development boards and flash them.

- [Download Simplicity Studio.](#)
- Building application Using Simplicity Studio:
 - [Build Application for EFR32](#)
 - [Build Application for SoC](#)
- Flash application Using Simplicity Studio:
 - [Build Application for EFR32 Step 9](#)
 - [Build Application for SoC Step 9](#)
- [Simplicity Studio Reference Guide](#)

Simplicity Commander

Links to download Simplicity Commander's standalone versions are included below. Full documentation on Simplicity Commander is included in the [Simplicity Commander Reference Guide](#)

- Linux: <https://www.silabs.com/documents/public/software/SimplicityCommander-Linux.zip>
- Mac: <https://www.silabs.com/documents/public/software/SimplicityCommander-Mac.zip>
- Windows: <https://www.silabs.com/documents/public/software/SimplicityCommander-Windows.zip>

How to Find Your Raspberry Pi

How to find your Raspberry Pi on the Network

Finding the IP address of your Raspberry Pi

Sometimes it can be difficult to find your Raspberry Pi on the network. One way of interacting with the Raspberry Pi is to connect a keyboard, mouse, and monitor to it. The preferred method, however, is over SSH. For this, you will need to know the IP address of your Raspberry Pi.

This is a [good tutorial](#) on how to find the IP address.

Mac / Linux

Nmap

The use of nmap on the Mac may require a software download. Use nmap with the following command:

```
$ sudo nmap -sn <subnet>.0/24`
```

Example: sudo nmap -sn 1-.4.148.0/24 , Among other returned values, you will see something:

```
$ Nmap scan report for ubuntu.silabs.com (10.4.148.44)
$ Host is up (0.00025s latency).
$ MAC Address: E4:5F:01:7B:CD:12 (Raspberry Pi Trading)
```

And this is the Raspberry Pi at 10.4.148.44.

Arp

Alternatively, use Arp with the following command:

```
$ arp -a \| grep -i "b8:27:eb\|dc:a6:32"
```

Windows

In the command prompt, use nslookup to fnd your Raspberry Pi.

Example: nslookup ubuntu

Connecting to your Raspberry Pi over SSH

Mac / Linux / Windows

Once you have found your Raspberry Pi's IP address, you can use Secure Shell (SSH) to connect to it over the command line with the following command:

```
$ ssh <raspberry pi's username>@<raspberry pi's IP address>
```

Example:

```
$ ssh ubuntu@10.4.148.44
```

password: raspberrypi or ubuntu When prompted provide the raspberry pi's password, in the case of the Silicon Labs Matter Hub image the username is ubuntu and the password is either ubuntu or <a user set password> . (Note that if you are logging into your Raspberry Pi for the first time you may be asked to change the default password to a password of your choosing.)

Using Development Tools in Simplicity Studio

Development Tools in Simplicity Studio

Simplicity Studio contains a number of integrated tools that you can use with projects created within that environment.

Bluetooth GATT Configurator

The Simplicity Studio Bluetooth LE (BLE) GATT Configurator is an Advanced Configurator within the Simplicity Studio Project Configuration suite. You can add the BLE GATT configuration by adding the `Bluetooth > GATT > Configuration` component to your project. This will enable the BLE GATT Configurator under `Configuration Tools > Bluetooth GATT Configurator`. More information on using the [BLE GATT Configurator is provided here](#).

Energy Profiler

Simplicity Studio's Energy Profiler allows you to see a graphical view of your device's energy usage over time. This can be very useful when developing an energy-friendly device.

Complete documentation on using the Simplicity Studio Energy Profiler is provided in the [Simplicity Studio 5 Energy Profiler User's Guide](#).

Custom Hardware Configuration

At some point during product development you may need to move your project over to your custom hardware. In this case, you will likely need to change the pinout and hardware configuration in the example project to reflect your own custom project. You can do this using the Pin Tool. The Simplicity Studio 5 User's Guide contains full documentation using the [Pin Tool](#).

Building a Custom Matter Device

Custom Matter Device Development

Build a customizable lighting app using the Matter protocol.

Overview

This guide covers the basics of building a customizable lighting application using Matter.

Using Matter with Clusters

In Matter, commands can be issued by using a cluster. A cluster is a set of attributes and commands which are grouped together under a relevant theme.

Attributes store values (think of them as variables). Commands are used to modify the value of attributes.

For example, the "On/Off" cluster has an attribute named "OnOff" of type boolean. The value of this attribute can be set to "1" by sending an "On" command or it can be set to "0" by sending an "Off" command.

The C++ implementation of these clusters is located in the clusters directory. Note that you can also create your own custom cluster.

ZAP Configuration

In Studio, navigate to the ZAP UI in your project by double clicking the zap file located at `config/zap/lighting-app.zap`.

On the left side of the application, there is a tab for Endpoint 0 and Endpoint 1. Endpoint 0 is known as the root node. This endpoint is akin to a "read me first" endpoint that describes itself and the other endpoints that make up the node. Endpoint 1 represents a lighting application device type. There are a number of required ZCL clusters enabled in Endpoint 1. Some clusters are common across most device types, such as identify and group clusters. Others, such as the On/Off, Level Control and Color Control clusters are required for a lighting application device type.

Clicking on the blue settings icon on the right side of the application brings you to the zap configuration settings for that cluster. Each cluster has some required attributes that may cause compile-time errors if they are not selected in the zap configuration. Other attributes are optional and are allowed to be disabled. Clusters also have a list of client-side commands, again some are mandatory and others are optional depending on the cluster. ZCL offers an extensive list of optional attributes and commands that allow you to customize your application to the full power of the Matter SDK.

For example, if a lighting application only includes single color LEDs instead of RGB LEDs, it might make sense to disable the Color Control cluster in the ZAP configuration. Similarly, if a lighting application does not take advantage of the Level Control cluster, which allows you to customize current flow to an LED, it might make sense to disable the Level Control cluster.

Receiving Matter Commands

All Matter commands reach the application through the intermediate function `MatterPostAttributeChangeCallback()`. When a request is made by a Matter client, the information contained in the request is forwarded to a Matter application through this function. The command can then be dissected using conditional logic to call the proper application functions based on the most recent command received.

Adding a Cluster to a ZAP Configuration

In the ZAP UI, navigate to the Level Control cluster. Make sure this cluster is enabled as a server in the drop-down menu in the "Enable" column. Then click on the blue settings wheel in the "Configure" column. This cluster can be used to gather

power source configuration settings from a Matter device. It contains a few required attributes, and a number of optional attributes.

Adding a New Attribute

In the Level Control cluster configurations, ensure the CurrentLevel attribute is set to enabled. Set the default value of this attribute as 1.

Adding a New Command

Navigate to the commands tab in zap and enable the MoveToLevel command. Now save the current zap configuration, and run the generate.py script above.

React to Level Control Cluster Commands in ZclCallbacks

In the MatterPostAttributeCallback function in ZclCallbacks, add the following line of code or a similar line. This will give the application the ability to react to MoveToLevel commands. You can define platform-specific behavior for a MoveToLevel action.

```
else if (clusterId == LevelControl::Id)
{
    ChipLogProgress(Zcl, "Level Control attribute ID: " ChipLogFormatMEI " Type: %u Value: %u, length %u",
                    ChipLogValueMEI(attributeId), type, *value, size);

    if (attributeId == LevelControl::Attributes::CurrentLevel::Id)
    {
        action_type = LightingManager::MOVE_TO_LEVEL;
    }

    LightMgr().InitiateActionLight(AppEvent::kEventType_Light, action_type, endpoint, *value);
}
```

Send a MoveToLevel Command and Read the CurrentLevel Attribute

Rebuild the application and load the new executable on your EFR32 device. Send the following mattertool commands and verify that the current-level default attribute was updated as was configured. Replace {desired_level} with 10, and node_ID with the node ID assigned to the device upon commissioning.

```
$ mattertool levelcontrol read current-level 1 1 // Returns 1

$ mattertool levelcontrol move-to-level {desired_level} 0 1 1 {node_ID} 1

$ mattertool levelcontrol read current-level 1 1 // Returns 10
```

For more information on running a Silicon Labs lighting example on a Thunderboard Sense 2 see you can view documentation in the Silicon Labs Matter GitHub Repo.

Building a Multi-Endpoint Device

Creating a Multiple Endpoint Matter Device in Studio

Matter products can have multiple device types spread out across different endpoints. The following guide walks the user through adding a lighting endpoint to the standard lock example in Simplicity Studio.

[Develop a Lock + Light Example with Matter.](#)

Using ZAP, the ZCL Advanced Platform

ZCL Advanced Platform (ZAP) Tool for Matter

Overview

EFR32 example applications provide a baseline demonstration of a lock device, built using the Matter SDK and the Silicon Labs GeckoSDK. It can be controlled by a CHIP controller over OpenThread network.

The EFR32 device can be commissioned over Bluetooth Low Energy (BLE) where the device and the CHIP controller will exchange security information with the Rendez-vous procedure. Thread Network credentials are provided to the EFR32 device which will then join the network.

The LCD on the Silicon Labs WSTK shows a QR Code containing the needed commissioning information for the BLE connection and starting the Rendez-vous procedure.

The lock example is intended to serve both as a means to explore the workings of CHIP, and a template for creating real products on the Silicon Labs platform.

Each Matter application consists of the following layers:

- Matter SDK: Source code necessary to communicate through the Matter network over Thread or Wi-Fi
- Data model layer in the form of clusters. There are two types of clusters:
 - Utility Clusters:
 - They represent common management and diagnostic features of a Matter endpoint
 - Identify cluster is an example of a Utility Cluster. Given a Node ID, it can be used to Blink LED0 to the corresponding Silicon Labs WSTK
 - Application Clusters:
 - These clusters represent functionalities specific to a given application
 - Door Lock Cluster is an example of an Application specific cluster. This cluster contains commands to lock and unlock a door(door-lock is represented by an LED), with options to set passwords and lock schedules

Clusters

Every Matter Application uses multiple clusters leveraged from the Zigbee Cluster Library (ZCL). A cluster can be seen as a building block for the Data Model of a Matter application. Clusters contains attributes, commands, and events. Attributes are customizable variables specified by the Zigbee Advanced Platform (ZAP) tool. Commands are sent to the application, which may respond with data, LED flickering, lock actuation, etc. Events are notifications sent out by the server.

An application can have multiple Matter endpoints. Application endpoints generally refer to one device, and inherits its information from the "cluster" it belongs to. Utility clusters are required to be on the endpoint with ID 0. Application clusters are assigned to endpoints with IDs 1 and higher.

Some applications have callbacks that are left to be implemented by the device manufacturer. For example, the storage and management of users and credentials in the lock-app is left up to the application developer.

ZAP Tool

The ZAP tool is built and maintained by Silicon Labs and developers in the ZAP open source community. It inherits its name and features from the Zigbee Cluster Library, which was the starting point for the Matter data model. ZAP is used for generating code for Matter applications based on the Zigbee Cluster Library and associated Matter code templates.

The ZAP tool is no longer present as a submodule in the Matter repo. The ZAP tool can be downloaded as a binary from GitHub or optionally you can clone the entire ZAP repo and build the ZAP binary from scratch.

ZAP binaries can be downloaded from the latest ZAP release here:

```
https://github.com/project-chip/zap/releases/latest
```

Optionally, the ZAP tool can be cloned using the following git command. This will create a root level zap folder in your current directory.

```
$ git clone https://github.com/project-chip/zap.git
```

The ZAP tool can be invoked using the `run_zaptool.sh` script located in the Matter repo at `./scripts/tools/zap/run_zaptool.sh`. Before you run this script you have to provide the location of the ZAP instance to be run. This is either the binary that you downloaded or the binary that you built from scratch in the ZAP repo. You can do this by setting the `ZAP_INSTALL_PATH` environment variable like this:

```
$ export ZAP_INSTALL_PATH=(path to your instance of the ZAP binary)
```

The `run_zaptool.sh` script can be invoked without arguments, or, you can provide the path to a ZAP file to be opened upon launch.

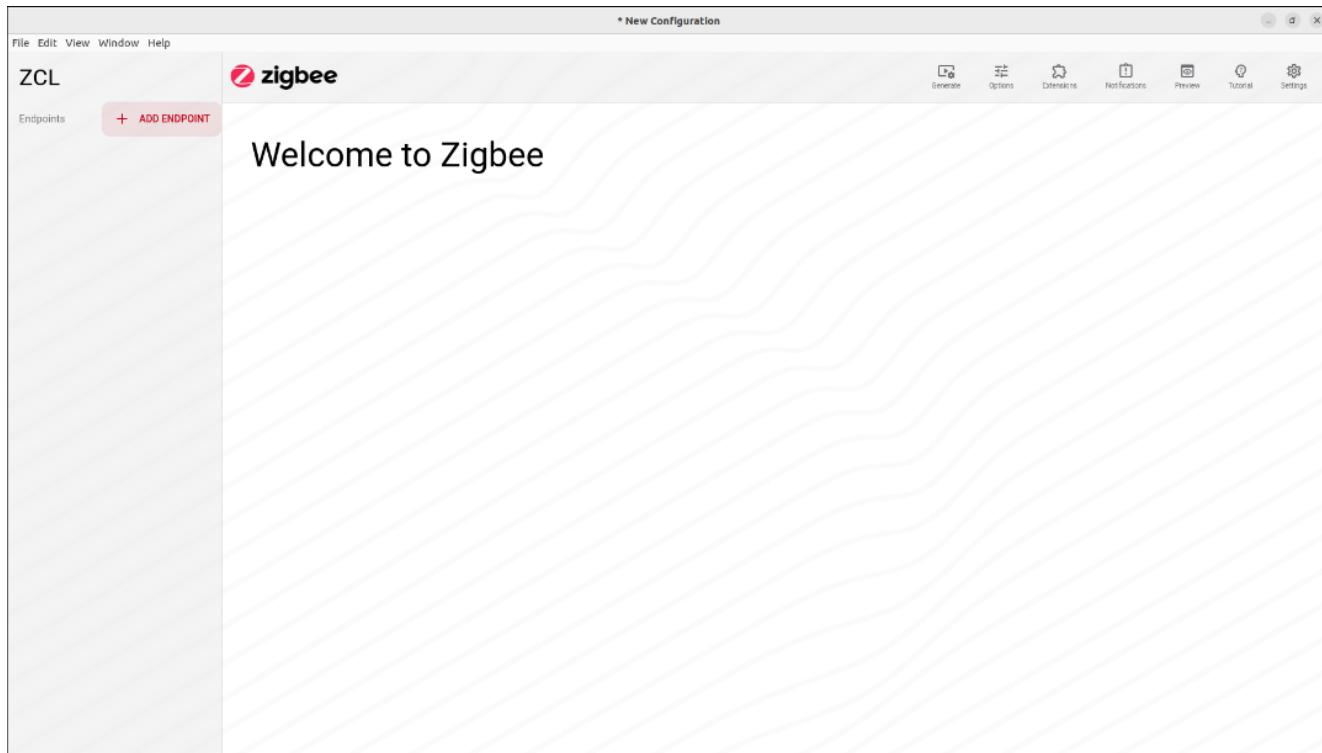
In the following examples, the ZAP file for the lock-app has been chosen.

```
$ ./scripts/tools/zap/run_zaptool.sh ($PATH_TO_ZAP_FILE)
```

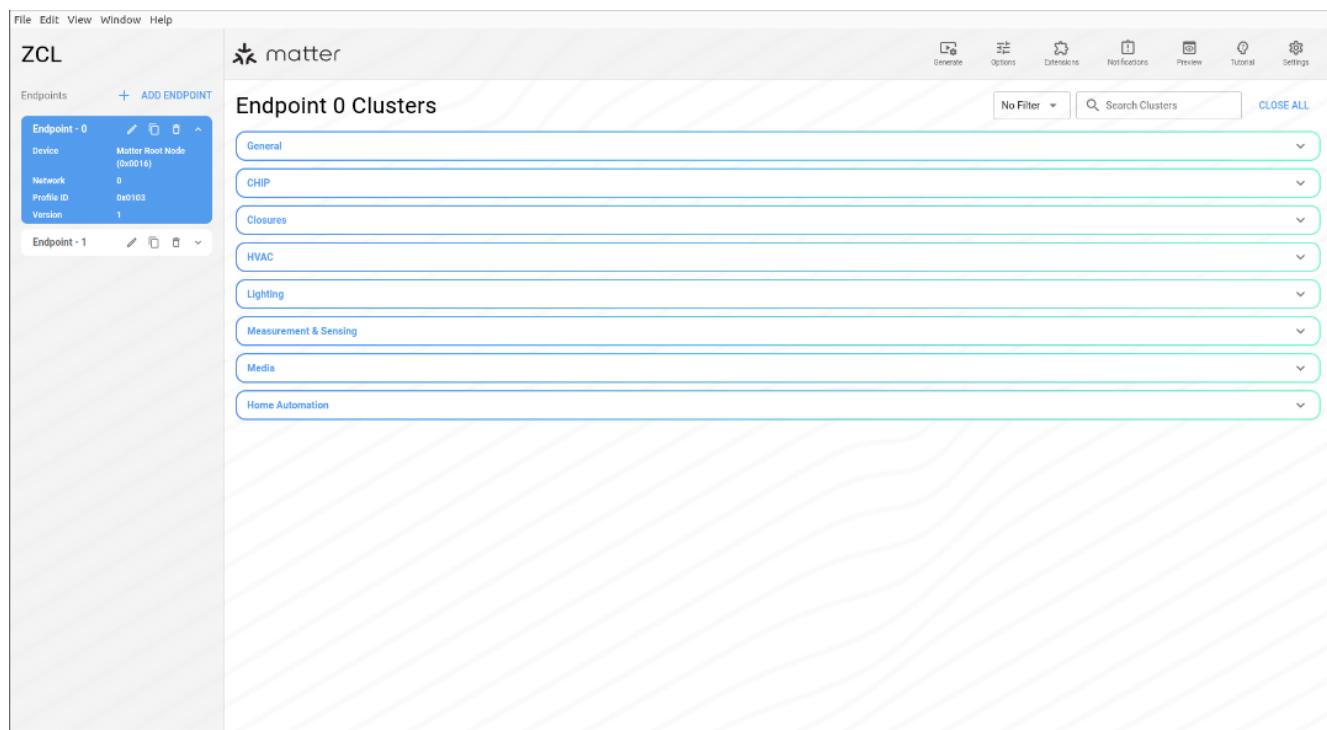
ZAP files for the various sample applications are included in the sample applications `data_model` directory such as

```
./examples/lighting-app/silabs/efr32/data_model/lighting-thread-app.zap .
```

This shows the output of the `run_zaptool` script with no arguments. To load a new zap file, click the application menu for Electron (Upper left corner of the screen for macs), then click "Open File". Then navigate to the desired .zap file.



This shows the output of the run_zaptool script with a zap file given as an argument, or after a .zap file has been opened in the ZAP UI. An Electron application will open, pre-loaded with the information from the .zap file provided as a command line argument.



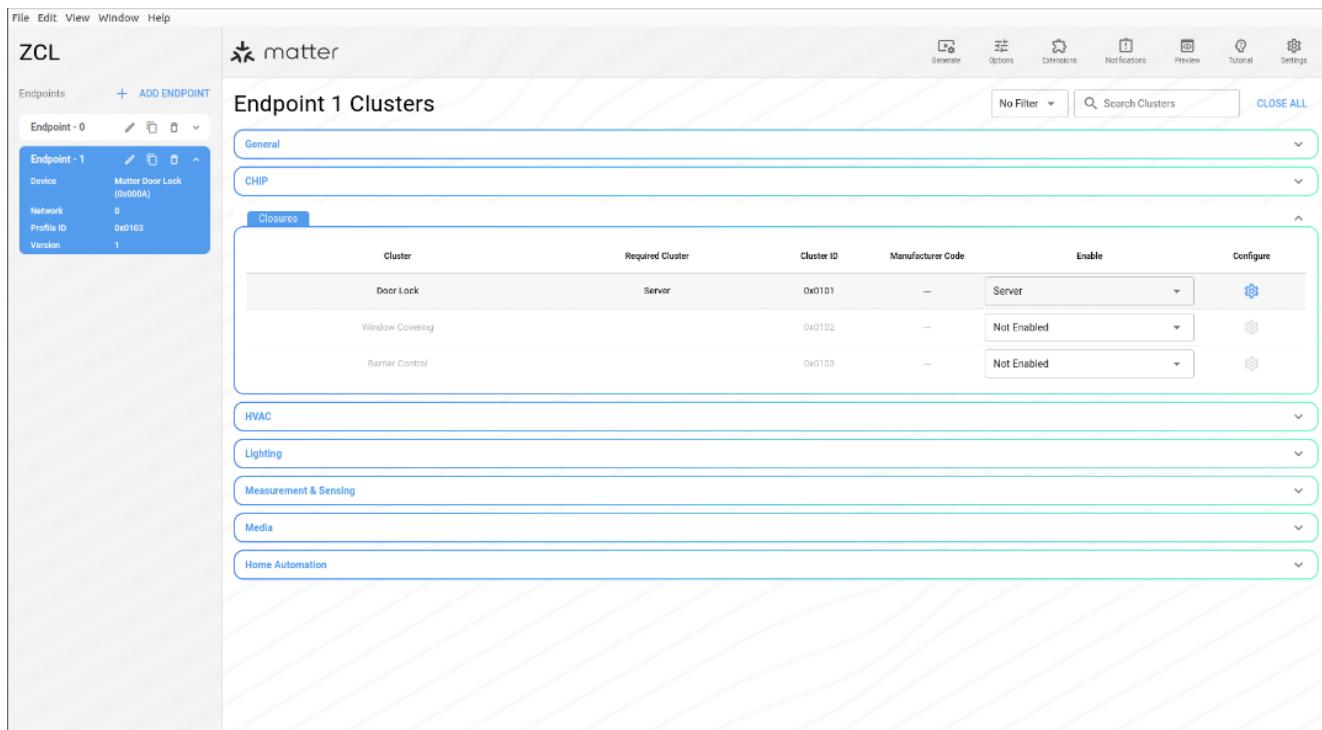
The Out of the box (OOB) example lock application has 2 endpoints. Endpoint 0 is called the root node. It contains all Service and Device management clusters. In general, any cluster or feature that is not specific to a device type belongs in Endpoint 0. Examples of clusters one might find in Endpoint 0: Device Descriptor cluster, Network Diagnostics cluster.

Endpoint 1 contains information specific to the device type. Conveniently, the ZAP tool offers a Door lock cluster, which contains Commands (lock, unlock, set credential, and so on) and Attributes (Lock state, Require PIN) that a standard door lock application might use.

More endpoints can be added. Each endpoint acts like a port on a network interface.

Endpoints contain clusters which are bundles of device functionality. Clusters have both a Client and a Server interface. In general the Client interface sends commands and the Server interface receives them. For instance a Light would implement the Server side of the on/off clusters. A Switch would implement the Client side of the same cluster.

Click on Endpoint 1 on the left hand side of the application. The door lock cluster should already be enabled as "Server".



Attributes

Attributes are analogous to member variables of a class. Each attribute is provided with generated setter/getter code from the ZAP tool. They can be enabled or disabled for each cluster on a Matter endpoint. Some attributes are required to be enabled, else the application will not function properly. There is an option to add attributes to either the server code or client code. The ZAP tool also allows you choose a storage space for attributes. Attributes can be stored in standard RAM, Non-volatile memory or external memory. Each attribute has a type, some are standard C types and some have specially defined enums. Each attribute can be provided with a default starting value value.

Click the settings wheel to enable/disable, choose a storage option, and choose a default value for attributes, commands and events for Endpoint 1.

Door Lock

Endpoint 1 / Closures / Door Lock

An interface to a generic way to secure a door
Cluster ID: 0x0101, Enabled for Server

Search attributes

Enabled	Attribute ID	Attribute	Required	Client/Server	Mfg Code	Storage Option	Singleton	Bounded	Type	Default	
<input checked="" type="checkbox"/>	0x0000	LockState	Yes	Server	NVM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DILLOCKSTATE	1	NULL
<input checked="" type="checkbox"/>	0x0001	LockType	Yes	Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DILLOCKTYPE		
<input checked="" type="checkbox"/>	0x0002	ActuatorEnabled	Yes	Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	BOOLEAN		
<input checked="" type="checkbox"/>	0x0003	DoorState		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DOORSTATENUM		NULL
<input type="checkbox"/>	0x0004	DoorOpenEvents		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT32U		
<input type="checkbox"/>	0x0005	DoorClosedEvents		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT32U		
<input type="checkbox"/>	0x0006	OpenPeriod		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT16U		
<input checked="" type="checkbox"/>	0x0011	NumberOfTotalUsersSupported		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT16U	10	
<input checked="" type="checkbox"/>	0x0012	NumberOfPINUsersSupported		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT16U	10	
<input checked="" type="checkbox"/>	0x0013	NumberOfRFIDUsersSupported		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT16U	10	
<input checked="" type="checkbox"/>	0x0014	NumberOfWeekDaySchedulesSupportedPerUser		Server	RAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	INT8U	10	

Records per page: All ▾ 1-42 of 42

Commands

Commands can be enabled/disabled like attributes. Some commands are required for an application to function properly. Many of the functions run when a command is received are implemented on the server side. But some of these are left up to the application to define. In the EFR32 lock example, the set/get user and credential functions are customizable as each implementation of a lock might store these differently.

Door Lock

Endpoint 1 / Closures / Door Lock

An interface to a generic way to secure a door
Cluster ID: 0x0101, Enabled for Server

Search commands

Attributes	Attribute Reporting	Commands	Events			
Out	In	Direction	ID	Command	Required	Manufacturing Id
<input checked="" type="checkbox"/>		Client → Server	0x00	LockDoor	Yes	-
<input checked="" type="checkbox"/>		Client → Server	0x01	UnlockDoor	Yes	-
<input checked="" type="checkbox"/>		Client → Server	0x03	UnlockWithTimeout	-	-
<input checked="" type="checkbox"/>		Client → Server	0x0B	SetWeekDaySchedule	-	-
<input checked="" type="checkbox"/>		Client → Server	0x0C	GetWeekDaySchedule	-	-
<input checked="" type="checkbox"/>		Server → Client	0x0C	GetWeekDayScheduleResponse	-	-
<input checked="" type="checkbox"/>		Client → Server	0x0D	ClearWeekDaySchedule	-	-
<input checked="" type="checkbox"/>		Client → Server	0x0E	SetYearDaySchedule	-	-
<input checked="" type="checkbox"/>		Client → Server	0x0F	GetYearDaySchedule	-	-
<input checked="" type="checkbox"/>		Server → Client	0x0F	GetYearDayScheduleResponse	-	-
<input checked="" type="checkbox"/>		Client → Server	0x10	ClearYearDaySchedule	-	-

Records per page: All ▾ 1-24 of 24

Generation of Code

Once you have chosen the cluster options, save the current ZAP configuration using the application menu in the upper left corner.

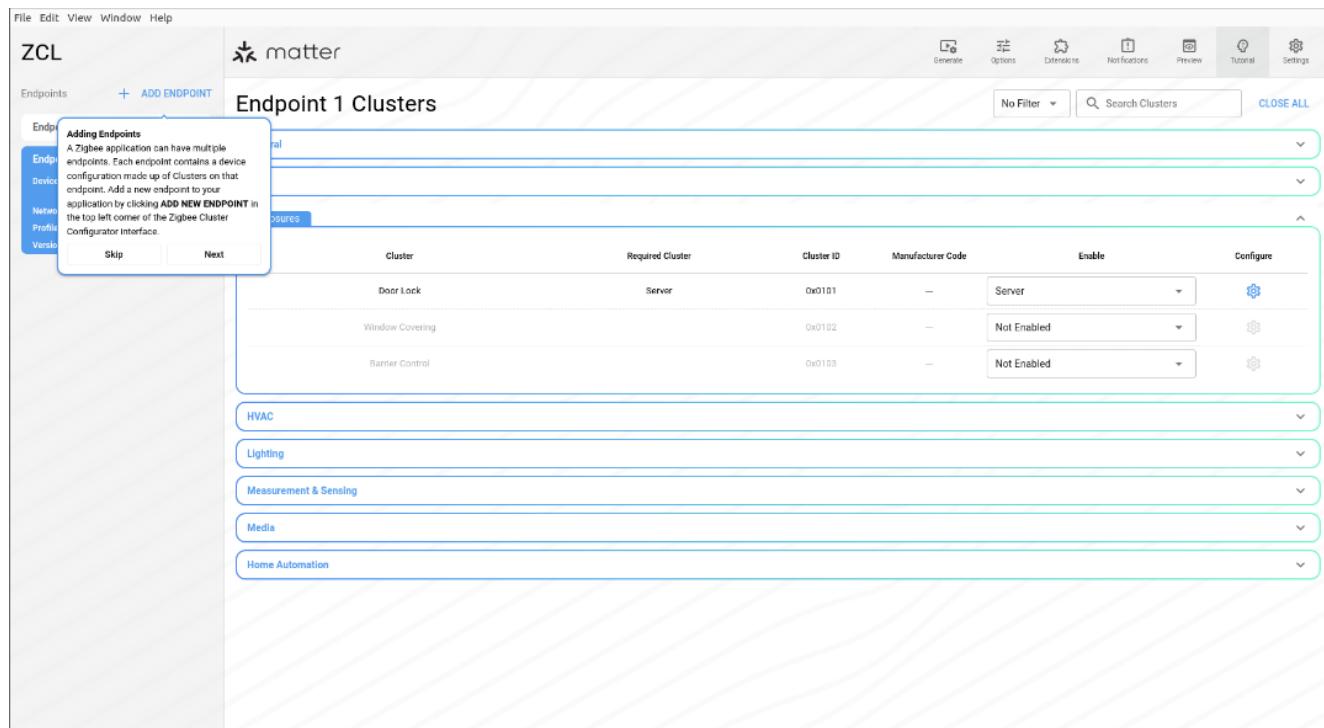
Before v1.1.0-1.1 you needed to click the Generate button to generate code. Now, code is generated automatically in the save function. You will be prompted to choose a save location for the generated ZAP code. In the Silicon Labs Matter repository, the lock-app generated files belong in matter/zzz_generated/lock-app/zap-generated.

New Tutorial Button

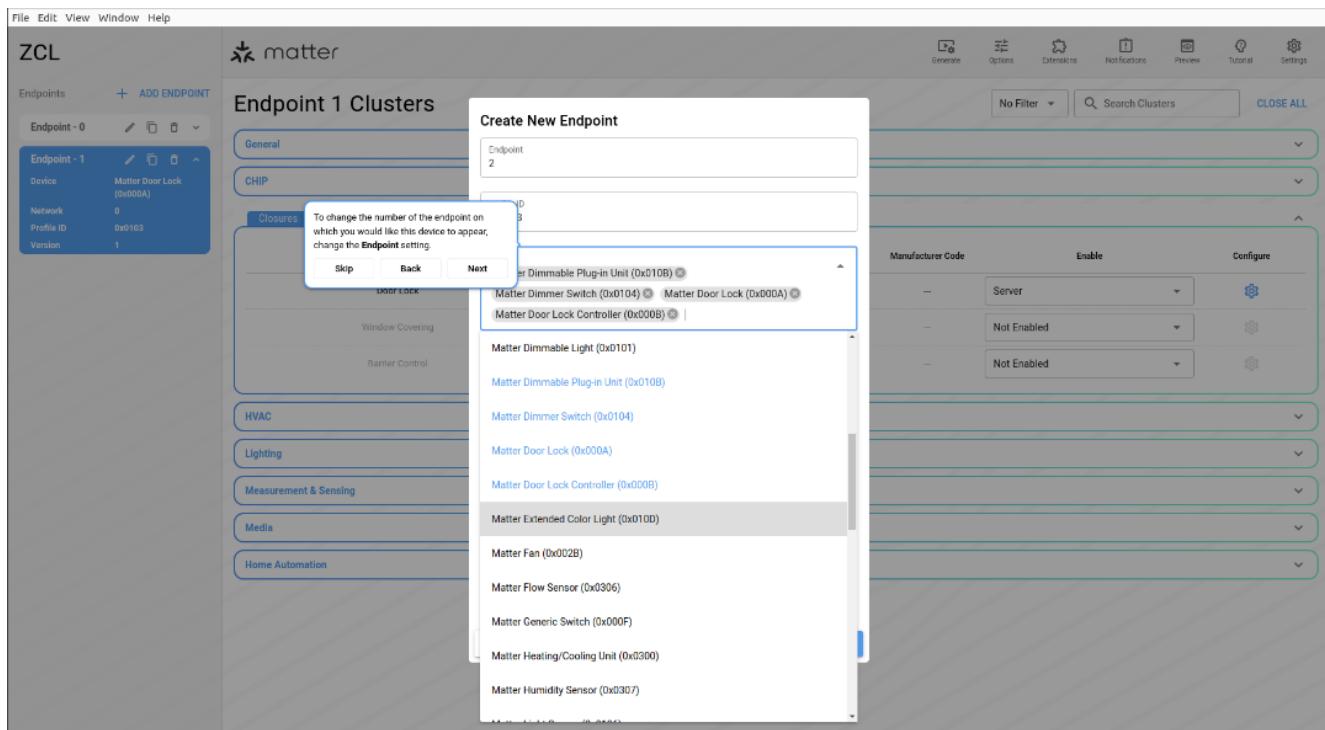
This new feature helps you understand all the steps needed to create a new endpoint.

Click the **Tutorial** button at the top-right side, between the **Preview** and the **Settings** buttons.

A pop-up displays with instructions for next steps.



These steps guide you on the components of this tool, the effects of each component, and the points you need to consider carefully when creating a new endpoint.

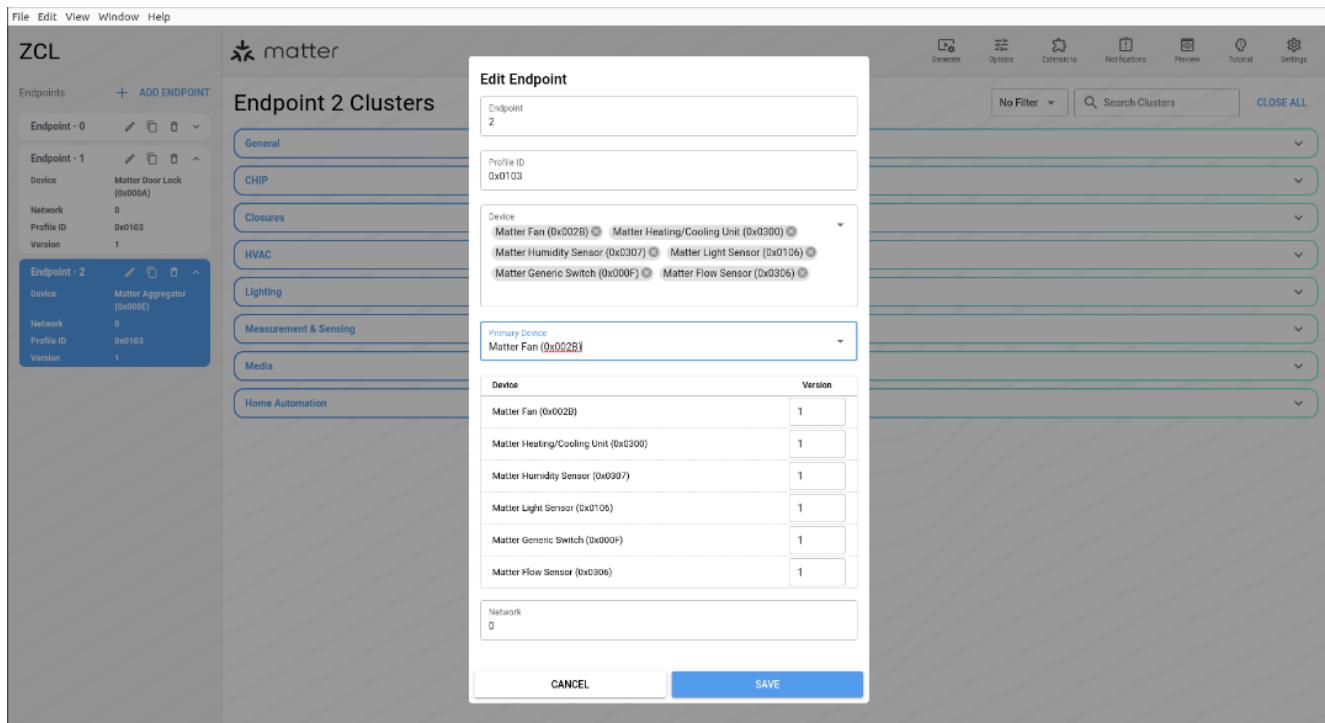


At the final step, a notification asks if you want to keep the endpoint you just created. Select what you want to finish the tutorial.

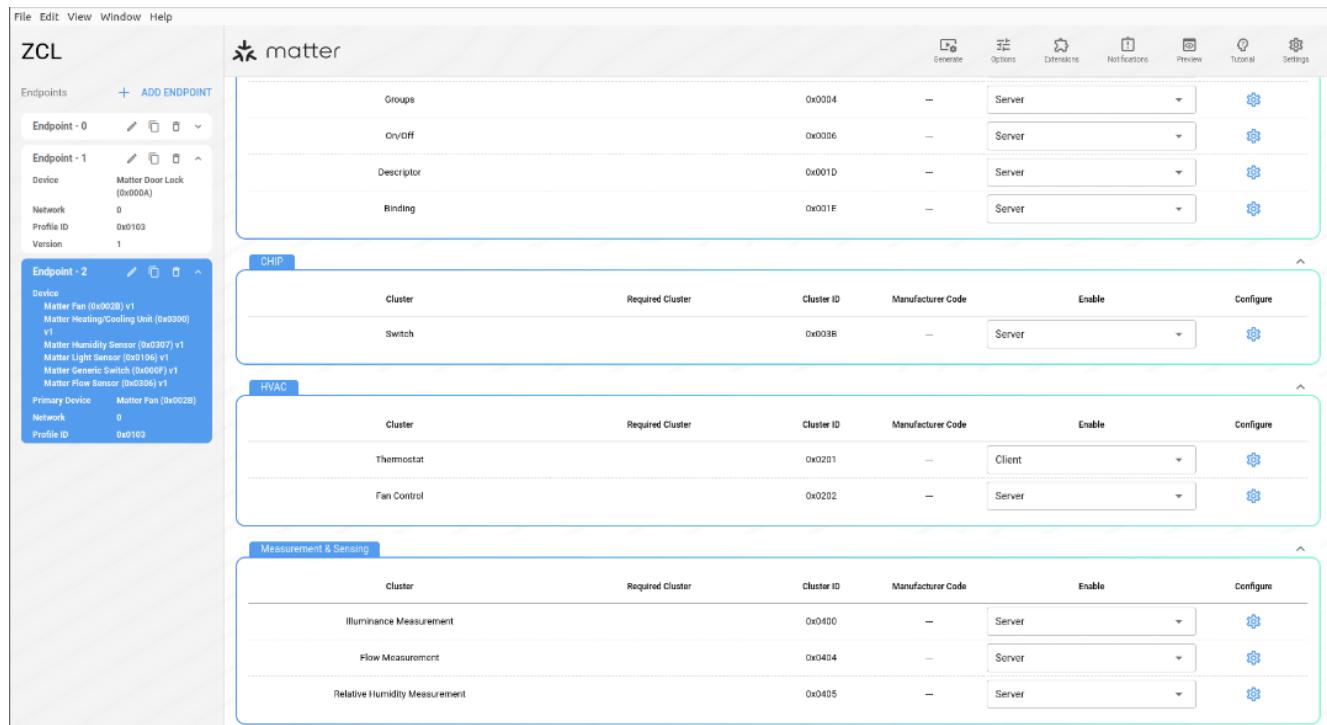
Multiple ZCL Device Types per Endpoint

This is a new Matter-only feature where you can select more than one ZCL device type per endpoint. The addition of multiple device types will add the cluster configurations within the device types to the endpoint configuration.

You can select multiple ZCL device types per endpoint, but there is only one **Primary Device** as in the below image.



The image above shows that endpoint 2 has more than one device type selected. The **Primary Device** denotes the primary device type that the endpoint will be associated with. The primary device type is always present at index 0 of the list of device types selected so selecting a different primary device type will change the ordering of the device types selected. The device type selections also have constraints based on the Data Model specification. ZAP protects you from choosing invalid combinations of device types on an endpoint using these constraints.



The image above is what the multiple device type endpoints looks like after configured. On the left-hand side, it shows the list of all clusters sequentially in order, and on the other side are clusters that are ready to be configured.

Using Wireshark with Matter

Using Wireshark to Capture Network Traffic in Matter

When developing a wireless application it is often useful to be able to visualize the network traffic. [Wireshark](#) is a great tool for this, but you can't use Wireshark alone. You first have to capture the network traffic off a wireless network interface. Fortunately Silicon Labs has provided an open source project for capturing network traffic off its devices called the [Java Packet Trace Library or Silabs PTI jar](#)

Here are the following steps for capturing and visualizing network traffic with Wireshark and the Java Packet Trace Library:

1. Clone and Build Silabs-PTI.jar Out of the Java Packet Trace Library

The [Java Packet Trace Library](#) can be built locally for your development platform. First clone the repository and then build the library according to the [instructions](#)

2. Download Wireshark

If you don't have Wireshark, you can [download Wireshark for your development platform here](#).

3. Follow Instructions for Wireshark Integration

To capture from a Silicon Labs device like a WSTK use the Silabs-PTI.jar utility you built in the previous step. You will further need to integrate the execution of the utility into Wireshark through Wireshark's `excap` interface. A complete guide to [Wireshark Integration is provided here](#). You integrate the Silabs-PTI.jar utility into Wireshark by adding a small script into Wireshark's `excap` directory. Make sure that you make the script executable using something like

```
$ chmod 777 <myexcapscript>
```

This will make it so that Wireshark can execute the script and integrate the WSTK interfaces into its capture functionality.

4. Run Wireshark and Discover and Capture using Silabs-PTI.jar

In order to capture from an adapter such as a WSTK using the utility Silabs-PTI.jar that you built in step 2, your adapter must be connected to the network via Ethernet. If your adapter is not connected via Ethernet and only via USB you will need to use the `silink` utility to make the adapter show up as a localhost.

Once your adapter is connected, you can test out the visibility of your WSTK on the network by running Silabs-PTI.jar from the command line using the following command:

```
$ java -jar silabs-pti-<VERSION>.jar -discover
```

Matter EFR32 Flash Savings Guide

Code Savings Guide for Building Matter Applications

- Remove unnecessary clusters from the zap configuration. Example applications have clusters enabled to support both Thread and WiFi transport layers such as the Diagnostics clusters.
- Remove optional components in from the Matter Project in Studio that may not be needed for a certain application. For example, removing the components Matter Display and Matter QR Code Display will save flash by disabling the LCD screen.

Matter FAQ

Silicon Labs Matter FAQ

This section provides the following FAQs and troubleshooting information.

- [Thread FAQ](#)
- [Wi-Fi FAQ](#)

Thread FAQ

Frequently Asked Questions for Matter over Thread

Demo

- Why are the `mattertool` commands not working after all the steps?
 - Check if the Radio Co-Processor (RCP) image was built and/or flashed correctly to the device.
 - Make sure you see a QR code on the display of the Matter Accessory Device (MAD).
 - Make sure the images being used to flash the Raspberry Pi, RCP and MAD are correct.
- How can I find the IP address of my Raspberry Pi?
 - First, make sure the Raspberry Pi is connected to a network (ethernet or Wi-Fi). This page has more information: [Setting up the Matter Hub \(Raspberry Pi\)](#)
 - Refer to this page for general questions on finding the Raspberry Pi on your network: [Finding your Raspberry Pi](#)
 - For more detailed information, refer to this page: [Raspberry Pi Remote Access](#)
- How can I use a crystal value different than the default (39.0 MHz) for my device?
 - When using an alternative crystal value (i.e.: different than 39.0 MHz), updating the clock speed for both the HFXO and DPLL values is needed and both must match, where the DPLL is set to 2x the HFXO value. These values are used by the RAIL library to determine the radio frequency and the proper timings for the BLE packets.
 - If the DPLL value is left unchanged with a modified HFXO, the radio will be on the right frequency but the BLE packet timings will not be correct, which will cause issues within a few packets due to BLE's strict timing requirements.

Wi-Fi FAQ

Frequently Asked Questions and Troubleshooting for Matter over Wi-Fi

Troubleshooting

1. Bluetooth connection fails when trying to commission the system through the chip-tool

Command leading to error:

```
$ out/standalone/chip-tool pairing ble-wifi 1122 mySSID myPassword 20202021 3840
```

Where `mySSID` is your AP's SSID and `mypassword` is your AP's password.

Error example:

```
[1659464425.856025][34818:34823] CHIP:DL: HandlePlatformSpecificBLEEvent 16386
[1659464425.856035][34818:34823] CHIP:IN: Clearing BLE pending packets.
[1659464425.856055][34818:34823] CHIP:IN: BleConnection Error: ../../examples/chip-
tool/third_party/connectedhomeip/src/platform/Linux/bluez/Helper.cpp:1775: CHIP Error 0x000000AC: Internal error
```

This error indicates that the Bluetooth connection between your system and laptop is failing. Follow the given procedure and then retry the chip-tool commissioning command.

Procedure:

1. Stop Bluetooth service:

```
$ systemctl stop bluetooth.service
```

2. Wait 20 seconds

3. Restart Bluetooth service:

```
$ sudo service bluetooth restart
```

4. Unblock Bluetooth service:

```
$ rfkill unblock all
```

5. Enable Bluetooth service:

```
$ sudo systemctl enable bluetooth
```

6. Issue the pairing command:

```
$ out/standalone/chip-tool pairing ble-wifi 1122 mySSID mypassword 20202021 3840
```

Where `mySSID` is your AP's SSID and `mypassword` is your AP's password.

2. Unsupported certificate format error

When trying to commission the system, if an `Unsupported certificate format` error (example below) is encountered, follow the procedure stated below.

Error example:

```
[1659631352.672826][5076:5076] CHIP:TOO: Run command failure: ../../examples/chip-tool/third_party/connectedhomeip/src/controller/CHIPDeviceController.cpp:1275: CHIP Error 0x00000050: Unsupported certificate format
```

Procedure:

- Delete the existing certificates on your laptop with the following command run from the `/connectedhomeip` directory:

```
$ /bin/rm /tmp/chip_*
```
- Issue the commissioning command

3. WLAN connection fails from RS9116 during commissioning when channel 13 is selected on the AP

The required channel becomes available for connection when the WLAN connection region is configured during compilation to one that supports the channel, such as for Japan for channel 13.

In order to use the desired channel, before building, make sure the WLAN connection region is configured correctly by reviewing/modifying the following lines in `/examples/platform/silabs/efr32/rs911x/rsi_wlan_config.h`:

```
//Make sure this is set to RSI_ENABLE
#define RSI_SET_REGION_SUPPORT RSI_ENABLE

// Note that the channels available for WLAN connection depend on the region selected
// Make sure this is set to 1 to configure from RSI_REGION_CODE value below
// 0: region configurations taken from beacon
// 1: region configurations taken from user
#define RSI_SET_REGION_FROM_USER_OR_BEACON 1

// 0 : Default Region domain
// 1 : US
// 2 : EUROPE
// 3 : JAPAN
#define RSI_REGION_CODE 3
```

4. Incorrect firmware version

```
cd ./third_party/silabs/wiseconnect-wifi-bt-sdk/firmware
```

You will get appropriate firmware in the above mentioned path.

Note:

1. How to check the current firmware version?

You can find the currently used firmware version in the DUT log.

2. How to check whether you are using correct firmware version or not?

Compare last 5 digits of firmware version mentioned in the above path with the currently used firmware version.

5. Apple HomePod associated failures

If there is an Apple HomePod on the network paired with a Thread device, and a commissioning failure is seen with error 3000001 :

Either remove the Apple HomePod from the network, or unpair it from all Thread devices, before re-trying the commissioning.

6. Commissioning failure at step 18

1. Verify router configuration specifically related to IPV6 SLAAC, Internet Group Management Protocol (IGMP) snooping.
2. Delete all the files of chip-tool /tmp folder. (`rm -rf /tmp/chip_*`)
3. After checking the router configuration, factory-reset your access point.

7. Commissioning failure at step 16

Verify the access point settings, SSID, PSK, security type, REGION, CHANNEL.

8. Inconsistent logs

Verify external power is supplied to rs911x

9. To enable different security options on AP/Router

1. Get the router address by entering `route -n` or ifconfig of ipconfig.
2. Enter the router address in the browser and enter the appropriate username and password.
3. Select the appropriate band.
4. In security, select type (WPA / WPA2 / WPA3).

10. CHIP Logs are not available on MG12 + WF200 due to image size constraints

Due to apps taking up more space than available flash on the MG12 + WF200 device combination, `chip_logging=false` needs to be included on the command line while building the app image, to disable CHIP logs and thereby reduce the image size.

This prevents debugging the code on the MG12 + WF200 device combination.

In order to work around this constraint, disable either the LCD or the use of QR codes, depending on your debugging needs. Disabling one of these will sufficiently reduce the image size to allow CHIP Logging to be enabled.

If you disabled QR Codes, you may use the `chip-tool` for commissioning the device.

If you disabled the LCD and need to debug with QR Codes, the URL to display the QR Code will be printed in the device logs.

Disable LCD and enable CHIP Logging: `./scripts/examples/gn_efr32_example.sh examples/lock-app/efr32 out/wf200_lock_app BRD4161A is_debug=false disable_lcd=true --wifi wf200 |& tee out/wf200_lock_app.log`

```
Disable QR Code and enable CHIP Logging: ./scripts/examples/gn_efr32_example.sh examples/lock-app/efr32 out/wf200_lock_app  
BRD4161A is_debug=false show_qr_code=false --wifi wf200 |& tee out/wf200_lock.log
```

11. MG24 device sometimes loses its connection to Ozone during OTA Update with RS9116

While performing an OTA Update with the EFR32MG24 + RS9116 device combination, when the device is reset and bootloading begins with the new image, the Ozone Debugger sometimes loses its connection.

There are two possible workarounds to this:

1. Immediately re-attach the device to the console when the connection is lost.
2. Download the RTT Viewer application instead and use it to view the logs during OTA Update.

12. MG24 device sometimes fails to bootload with the new image during OTA Update with WF200

While performing an OTA Update with the EFR32MG24 + WF200 device combination using the external flash, when the device is reset and bootloading begins with the new image, the device sometimes starts up with the existing image instead of the newly downloaded one.

When this happens, perform the following steps to run the OTA Update successfully:

1. Disconnect the WF200 Expansion Board from the EFR32MG24.
2. Go To the Simplicity Commander's folder path in the command prompt and run this command:

```
commander.exe extflash read --range 0x00:+<total size to read>
```

3. Reconnect the WF200 Expansion Board to the EFR32MG24 and reset the device.
4. Re-run the OTA Update process from the beginning.

Copyright © 2023 Silicon Laboratories. All rights reserved.