AN0204
Realtek Matter Application Note

Abstract

This document introduces Matter architecture. It describes the setup process, commonly used tools, commands, and the certification process.

Realtek Semiconductor Corp.

No. 2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan

Tel.: +886-3-578-0211. Fax: +886-3-577-6047

www.realtek.com

DISCLAIMER

Please Read Carefully:

Realtek Semiconductor Corp., (Realtek) reserves the right to make corrections, enhancements, improvements and other changes to its products and services. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

Reproduction of significant portions in Realtek data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Realtek is not responsible or liable for such reproduced documentation. Information of third parties may be subject to additional restrictions.

Buyers and others who are developing systems that incorporate Realtek products (collectively, “Customers”) understand and agree that Customers remain responsible for using their independent analysis, evaluation and judgment in designing their applications and that Customers have full and exclusive responsibility to assure the safety of Customers' applications and compliance of their applications (and of all Realtek products used in or for Customers’ applications) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to their applications, Customer has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Customer agrees that prior to using or distributing any applications that include Realtek products, Customer will thoroughly test such applications and the functionality of such Realtek products as used in such applications.

Realtek’s provision of technical, application or other design advice, quality characterization, reliability data or other services or information, including, but not limited to, reference designs and materials relating to evaluation kits, (collectively, “Resources”) are intended to assist designers who are developing applications that incorporate Realtek products; by downloading, accessing or using Realtek’s Resources in any way, Customer (individually or, if Customer is acting on behalf of a company, Customer’s company) agrees to use any particular Realtek Resources solely for this purpose and subject to the terms of this Notice.

Realtek’s provision of Realtek Resources does not expand or otherwise alter Realtek’s applicable published warranties or warranty disclaimers for Realtek’s products, and no additional obligations or liabilities arise from Realtek providing such Realtek Resources. Realtek reserves the right to make corrections, enhancements, improvements and other changes to its Realtek Resources. Realtek has not conducted any testing other than that specifically described in the published documentation for a particular Realtek Resource.

Customer is authorized to use, copy and modify any individual Realtek Resource only in connection with the development of applications that include the Realtek product(s) identified in such Realtek Resource. No other license, express or implied, by estoppel or otherwise to any other Realtek intellectual property right, and no license to any technology or intellectual property right of Realtek or any third party is granted herein, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Realtek products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of Realtek Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from Realtek under the patents or other Realtek’s intellectual property.

Realtek’s Resources are provided “as is” and with all faults. Realtek disclaims all other warranties or representations, express or implied, regarding resources or use thereof, including but not limited to accuracy or completeness, title, any epidemic failure warranty and any implied warranties of merchantability, fitness for a particular purpose, and non-infringement of any third-party intellectual property rights.
Realtek shall not be liable for and shall not defend or indemnify Customer against any claim, including but not limited to any infringement claim that related to or is based on any combination of products even if described in Realtek Resources or otherwise. In no event shall Realtek be liable for any actual, direct, special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of Realtek’s Resources or use thereof, and regardless of whether Realtek has been advised of the possibility of such damages. Realtek is not responsible for any failure to meet such industry standard requirements.

Where Realtek specifically promotes products as facilitating functional safety or as compliant with industry functional safety standards, such products are intended to help enable customers to design and create their own applications that meet applicable functional safety standards and requirements. Using products in an application does not by itself establish any safety features in the application. Customers must ensure compliance with safety-related requirements and standards applicable to their applications. Designer may not use any Realtek products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death. Such equipment includes, without limitation, all medical devices identified by the U.S.FDA as Class III devices and equivalent classifications outside the U.S.

Customers agree that it has the necessary expertise to select the product with the appropriate qualification designation for their applications and that proper product selection is at Customers' own risk. Customers are solely responsible for compliance with all legal and regulatory requirements in connection with such selection.

Customer will fully indemnify Realtek and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's non-compliance with the terms and provisions of this Notice.

TRADEMARKS

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

USING THIS DOCUMENT

Though every effort has been made to ensure that this document is current and accurate, more information may have become available subsequent to the production of this guide.

Revision History

Revision	Release Date	Summary
0.1	2023/07/14	Initial draft
0.2	2024/02/05	Update on Device Types, Factory Reset command, Matter Sample Application based on v1.2
0.3	2024/05/21	1. Updated on Device Types support for Matter v1.3 2. Added Chapter SDK Structure 3. Updated build instructions for v1.3 device type and Matter Restructure
0.4	2024/11/25	1. Updated on Device Types support for Matter v1.4 2. Added Release History 3. Added Chapter Integrating Ameba Matter Solution into your project 4. Updated on the change of Github Repository 5. Updated on Linux Image Tool for AmebaZ2 and AmebaD 6. Added Chapter Ameba Matter AT Command 7. Added Chapter Matter Device Configuration 8. Modified Build Project Section
0.5	2025/04/22	1. Added AmebaDplus, AmebaLite, and AmebaSmart specifications to Supported Realtek ICs section 2. Added ameba-rtos and ameba-rtos-matter branch to support ameba-rtos details to Ameba Project Repository, and Release History section 3. Added ameba-rtos structure to Ameba Matter SDK Structure section 4. Added ameba-rtos-matter clone method for ameba-rtos to Integrating Ameba Matter Solution section 5. Added AmebaDplus, AmebaLite, and AmebaSmart Matter sample code build guide to Build Project section 6. Added flash image and flash erase guide of AmebaDplus, AmebaLite, and AmebaSmart to Windows Image Tool section 7. Added Matter OTA firmware guide for AmebaDplus, AmebaLite, and AmebaSmart to Matter OTA section 8. Amended Matter DCT Chapter to Matter KVS Chapter 9. Added Ameba Z2 and AmebaD KVS implementation and AmebaDplus, AmebaLite, and AmebaSmart KVS implementation to Matter KVS section 10. Added AmebaZ2plus into AmebaZ2 sections
0.6	2025/08/20	1. Amended Build Project of AmebaDplus, AmebaLite, and AmebaSmart 2. Added Flash Factory Data to AmebaD/Z2/Z2plus and Flash Factory Data to AmebaDplus/Lite/Smart to Flash Factory Data Image section.
0.7	2025/11/12	1. Constructed table for Ameba Repository (Chapter 2.2.3) 2. Modified the porting layer make command for AmebaZ2/AmebaZ2plus/AmebaD (Chapter 5.2.3) 3. Added table for existing Matter AT command and Diagnostic Logs AT Command (Chapter 6) 4. Added DCT backup and wear level details (Chapter 13.1.3)

0.8	2026/01/15	1. Updated Release History (Chapter 2.2.4) 2. Updated hardware requirements for AmebaDplus, AmebaLite, AmebaSmart, and software requirements for ameba-rtos CMake build (Chapter 5.1) 3. Modified building steps of AmebaDplus / AmebaLite / AmebaSmart codes by dividing it into Makefile build and CMake build (Chapter 5.2) 4. Added Python Script Image Tool details (Chapter 5.4.1.3) 5. Modified Matter Device Configuration chapter by dividing it into Makefile build and CMake build (Chapter 7) 6. Modified step one of Building AmebaDplus / AmebaLite / AmebaSmart Matter OTA firmware (Chapter 11.1.3) 7. Modified AmebaDplus / AmebaLite / AmebaSmart KVS Implementation (Chapter 13.2)

Table of Contents

List of Figures

List of Tables

1 Summary

This Realtek Matter User Manual is to provide detailed guidance to all users or developers on how to use Realtek Ameba platform with Matter.
We shall mention the building of project, flashing of images, how to commission and control the Ameba application, as well as how to generate several Matter tools.

2 Introduction

2.1 About Matter

Matter is an open-source connectivity standard that targets to provide a more convenient way to link Smart Home devices. In this new technology, a single protocol seamlessly connects compatible IoT devices or systems with one another in the same network. Matter supports IP-based networking technologies and diverse network transportation via Thread, Wi-Fi, Ethernet as well as Bluetooth LE (BLE). You can visit Matter official website or GitHub to learn more about their solutions.
Let's briefly run through a few items to understand the Matter's data model architecture.

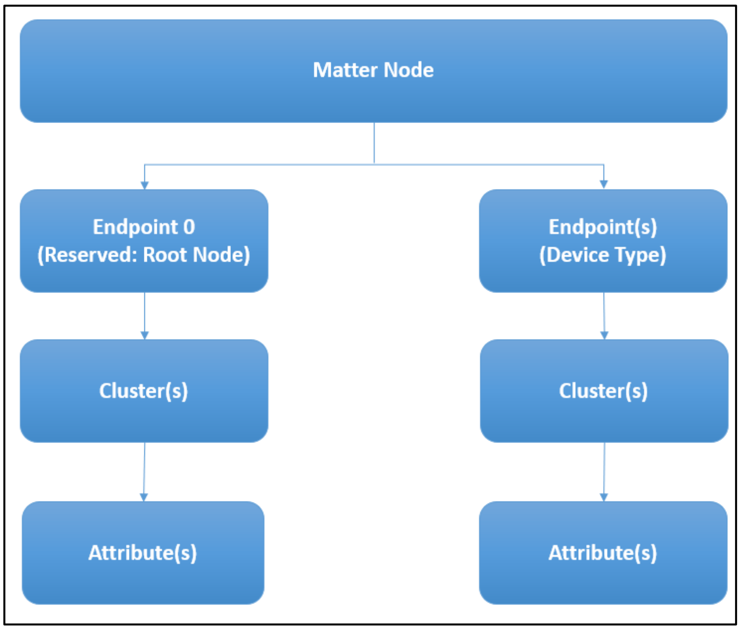

Figure 1: Matter Data Model Architecture

Matter's data model defines the elements that form a Matter Node.

- The Matter **node** is an addressable network resource that display a set of functions.
- The Matter node can have one or more **endpoints**. In Matter, an endpoint is also known as a **device type** which provides services.
- A **Cluster** has its functionality shaped into a building block. For instance, a Level Control cluster could adjust the brightness, and a Color Control cluster could adjust the color of a lighting device.
- A Cluster contains one or more **attributes**. An attribute can be used for reading and writing during network communication.

For more details about the Matter's data model architecture, please refer to this link.

2.2 Realtek Ameba Matter Solution

Realtek's Matter platform solution has both Wi-Fi and BLE working on a single SoC. Initial development of Realtek does not support Thread or Ethernet, and developers can decide whether to support BLE as Matter provides IP (Wi-Fi) commissioning. Theoretically speaking, the support of Wi-Fi is mandatory.
The following diagram is a simplified model of Realtek's Matter application.

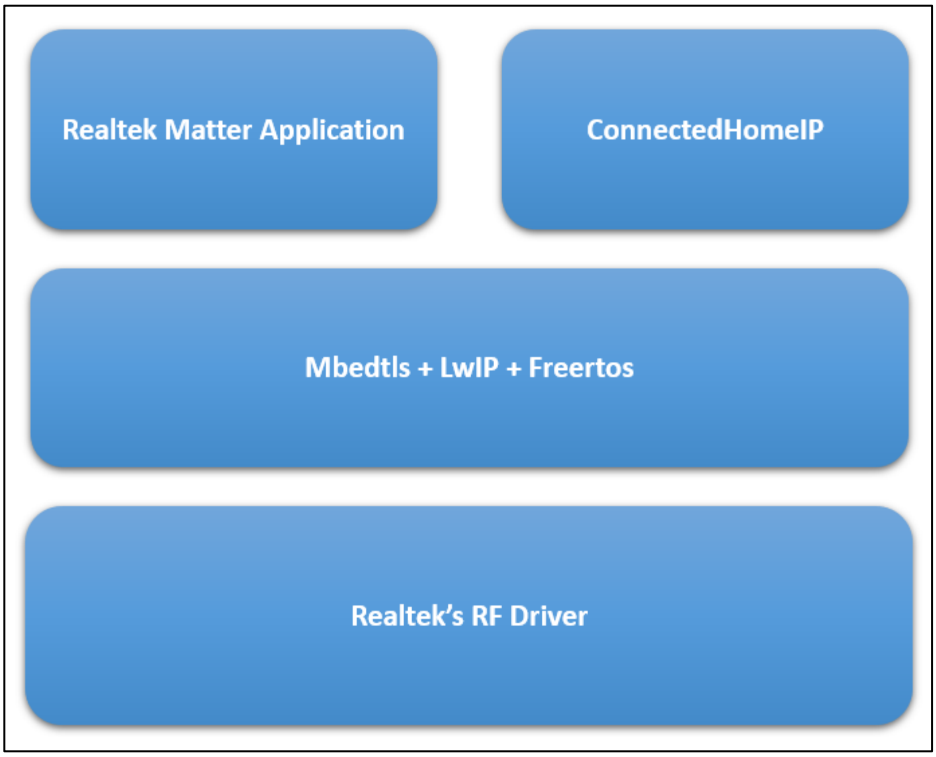

Figure 2: Realtek Matter Architecture

2.2.1 Supported Matter Device Types

This is a list of device types supported by Realtek:

1. Air Purifier
2. Air Quality Sensor
3. Color Dimmer Switch
4. Color Temperature Light
5. Contact Sensor
6. Cook Surface
7. Cooktop
8. Device Energy Management
9. Dimmable Light
10. Dimmable Plug-In Unit
11. Dimmer Switch
12. Dishwasher
13. Electrical Sensor
14. EVSE
15. Extended Color Light
16. Extractor Hood
17. Fan
18. Generic Sensor
19. Heat Pump
20. Heating/Cooling Unit
21. Humidifier/Dehumidifier
22. Humidity Sensor
23. Laundry Dryer
24. Laundry Washer
25. Light Sensor
26. Microwave Oven

1. Occupancy Sensor
2. On/Off Light
3. On/Off Light Switch
4. On/Off Plug-In Unit
5. On/Off Sensor
6. Oven
7. Power Source
8. Power Systems
9. Pressure Sensor
10. Pump
11. Rain Sensor
12. Refrigerator
13. Robotic Vacuum Cleaner
14. Room Air Conditioner
15. Smoke/CO Alarm
16. Solar Power
17. Speaker
18. Temperature Controlled Cabinet
19. Temperature Sensor
20. Thermostat
21. Water Freeze Detector
22. Water Leak Detector
23. Water Valve
24. Water Freeze Detector
25. Water Heater
26. Window Covering

Table 1: Supporting Device Types

* Table 1 is updated after Matter V1.5

* More device types will be added gradually as Matter expands their device supports. If your target device type is not listed, please contact us.

2.2.2 Supported Realtek ICs

The current supported ICs for Matter application and has been certified on:

(1) AmebaZ2 - RTL8720CM

The Realtek AmebaZ2 RTL8720CM is a highly integrated single chip with a low-power-consumption mechanism ideal for IoT (Internet of Things) applications. It combines a Real-M300 CPU (up to 100MHz), Wi-Fi, Bluetooth, Wireless MAC/Baseband/RF, and configurable GPIOs that can function as digital peripherals for various product applications and control usage. The RTL8720CM's embedded memory configuration enables simpler and faster application development.

The only exception is that AmebaZ2plus RAM size is 384KB while AmebaZ2 RAM size is 256KB.

MCU	32-bit Arm®Cortex®-M4, up to 100MHz
Memory	256KB SRAM + 4MB PSRAM
Key Features	Integrated 802.11n Wi-Fi SoC, Hardware SSL Engine, Root Trust Secure Boot, BLE4.2

Table 2: AmebaZ2 Specifications

(2) AmebaZ2plus - RTL8720C

The Realtek AmebaZ2plus has a larger RAM capacity of 384KB compared to the AmebaZ2's 256KB.

Memory	384KB SRAM

Table 3: AmebaZ2plus Specifications

(3) AmebaD - RTL8722DM

The Realtek AmebaD RTL8722DM multi-functional integrated IoT single chip has very high security level architecture and low power consumption. Dual-band Wi-Fi and Bluetooth 5 Mesh transmission achieves long-distance transmission and anti-interference performance across various wireless devices. An integrated global IoT ecosystem allows customers to develop advanced IoT applications with highest security.

MCU	32-bit KM4 (Arm Cortex-M33 compatible) 32-bit KM0 (Arm Cortex-M23 compatible)
Memory	512KB SRAM + 4MB PSRAM + 2MB Flash
Key Features	Integrated 802.11a/n Wi-Fi SoC Trustzone-M Security Hardware SSL Engine Root Trust Secure Boot USB Host/Device SD Host BLE5.0 Codec LCDC Key Matrix

Table 4: AmebaD Specifications

*Note: There is a need to replace external Flash to 4MB or higher to support Matter OTA feature.

(4) AmebaDplus - RTL8721Dx

The Realtek RTL8721Dx series is a Combo SoC that supports dual-band Wi-Fi 4 (2.4GHz + 5GHz) and Bluetooth (BLE 5.0) specifications. With excellent ultra-low power consumption, enhanced encryption strategy (PSA Level 2), and abundant peripheral resources, it is widely used in smart home appliance, line controller, smart door lock, battery camera, smart remote controller, Wi-Fi speaker, Wi-Fi Full MAC NIC, BLE gateway, and smart POS, etc.

MCU	Arm Cortex-M55 @345MHz Arm Cortex-M23@ 115MHz
Memory	512KB SRAM + Up to 8 MB PSRAM + Up to 4 MB Flash
Key Features	802.11 a/b/g/n 1 x 1, 2.4GHz + 5GHz MCS0 - MCS7 40MHz bandwidth BLE 5.0 BLE Mesh 1.1 Long Range Secure Boot & Crypto Engine

Table 5: AmebaDplus Specifications

(5) AmebaLite - RTL8720EA

The Realtek RTL8720EA series is a Combo SoC that supports Wi-Fi 6 (2.4GHz) and Bluetooth (BLE 5.2) specifications. With dual RISC cores running up to 400MHz, enhanced computing capability, stable security performance and abundant peripheral resources, it is widely used in various products such as home appliances, line controller, BLE gateway, micro inverter, portable energy storage, and home energy storage, etc.

MCU	Arm Cortex-M55 @400MHz RISC-V @400MHz
Memory	Up to 768KB SRAM + Up to 16MB PSRAM + Up to 8 MB Flash
Key Features	802.11 b/g/n/ax 1 x 1, 2.4GHz MCS0 - MCS9 20MHz bandwidth BLE 5.2 BLE Mesh 1.1 long Range Secure Boot & Crypto Engine

Table 6: AmebaLite Specifications

(6) AmebaSmart - RTL8730E

The Realtek RTL8730E series is a Combo SoC that supports dual-band Wi-Fi 6 (2.4GHz + 5GHz) and Bluetooth 5.3 specifications. The integrated dual-core Arm© Cortex-A32 CPU and RISC low-power MCU can meet a wide range of customer AIoT application needs. Combined with abundant peripheral interfaces, it has been successfully used in various products such as smart speaker, central control panel, home theater, wireless gateway, smart kitchen appliance, and industrial control panel, etc.

MCU	Dual-core Arm Cortex-A32 @1.32GHz Arm Cortex-M55 @333MHz Arm Cortex-M23 @40MHz
Memory	256KB SRAM + Up to 8MB PSRAM + Up to 32MB NOR Flash + Up to 256 MB DDR
Key Features	802.11 a/b/g/n/ax 1 x 1, 2.4GHz + 5GHz Bluetooth 5.3 Secure Boot & Crypto Engine Low Power design USB 2.0 OTG

Table 7: AmebaSmart Specifications

2.2.3 Ameba Project Repository

Note: This section applies to **the public repositories.**
Customers who have obtained the SDK through private distribution can proceed to the next chapter.

The following table summarizes the SDKs for different Ameba ICs and their compatibility with Matter releases. Matter integration is layered on top of these base SDKs.

IC	Matter Version	Repository	Notes
AmebaZ2	v1.3 and earlier	ambz2_matter	Primary SDK for AmebaZ2
AmebaD	v1.3 and earlier	ambd_matter	Primary SDK for AmebaD
AmebaZ2/Z2plus	v1.4 onwards	ameba-rtos-z2	Primary SDK for AmebaZ2/Z2plus
AmebaD	v1.4 onwards	ameba-rtos-d	Primary SDK for AmebaD
Ameba RTOS	v1.3 onwards	ameba-rtos	Primary SDK for AmebaDplus/Lite/Smart
Ameba RTOS Matter	v1.4 onwards	ameba-rtos-matter	Works alongside Z2, D, Dplus, Lite, Smart SDKs

Table 8: Ameba Repository and Matter Support

Notes

- Matter v1.3 and earlier are integrated into the original AmebaZ2 and AmebaD SDKs.
- Matter v1.4 and later introduced updated SDKs and a separate ameba-rtos-matter repository for flexible integration across multiple ICs.
- Developers should follow the integration guidelines in each repository for setup instructions and build procedures.

2.2.4 Release History

2.2.4.1 AmebaZ2 / AmebaZ2plus / AmebaD Release History

Matter Version	AmebaZ2/Z2plus SDK		Ameba RTOS Matter SDK	
	Repository	Tags/Branch	Repository	Tags/Branch
v1.0	ambz2_matter	v1.0.0.2	-	
v1.1		v1.1.0.1		
v1.2		v1.2.0.1		
v1.3		v1.3-release		
v1.4	ameba-rtos-z2	main	ameba-rtos-matter	release/v1.4
v1.4.2				release/v1.4.2
v1.5				release/v1.5

Table 9: AmebaZ2 / AmebaZ2plus Release History

Matter Version	AmebaD SDK		Ameba RTOS Matter SDK	
	Repository	Tags/Branch	Repository	Tags/Branch
v1.0	ambd_matter	v1.0.0.2	-	
v1.1		v1.1.0.1		
v1.2		v1.2.0.1		
v1.3		v1.3-release		
v1.4	ameba-rtos-d	main	ameba-rtos-matter	release/v1.4
v1.4.2				release/v1.4.2
v1.5				release/v1.5

Table 10: AmebaD Release History

2.2.4.2 AmebaDplus / AmebaLite / AmebaSmart Release History

Matter Version	Ameba RTOS SDK		Ameba RTOS Matter SDK	
	Repository	Tags/Branch	Repository	Tags/Branch
v1.3	ameba-rtos	release/v1.0+matter	ameba-rtos-matter	ameba-rtos/release/v1.3
v1.4				ameba-rtos/release/v1.4
v1.5	-			ameba-rtos/release/v1.5

Table 11: AmebaDplus / AmebaLite / AmebaSmart Release History

2.2.4.3 connectedhomeip Release

Additionally, the connectedhomeip repository is required, as it contains the core Matter source code maintained by the official Matter project.

Matter Version	Tags/Branch
v1.0	v1.0.0.2
v1.1	v1.1.0.1
v1.2	v1.2.0.1
v1.3	SHA: 70d9a614
v1.4	v1.4-branch
v1.4.2	V1.4.2-branch
v1.5	v1.5-branch

Table 12: connectedhomeip Release

2.2.5 Ameba Application Notes for Reference

For access to the Ameba Application Notes, please reach out to members of Realtek to obtain the relevant documentation.

3 Ameba Matter SDK Structure

This chapter serves as an introduction to the SDK structure, aiming to provide clarity on the areas where changes have been made for Matter integration. It outlines the anticipated layout of the Ameba SDK following the integration of Matter's code.

```
Ameba SDK
    |_ component
        |_ common/application/matter (for Ameba Z2/D) or application/matter (Ameba Dplus/Lite/Smart)
            |_ api
            |_ common
                |_ atcmd
                |_ bluetooth
                |_ include
                |_ lwip
                |_ mbedtls
                |_ port
                |_ protobuf
            |_ core
            |_ docs
            |_ drivers
            |_ examples
                |_ individual subfolders for different device types
            |_ project
                |_ amebaz2 or amebad or others
                    |_ make
                        |_ individual subfolders for different device types
            |_ tools
    |_ doc
    |_ projects
    |_ third_party
        |_ connectedhomeip
    |_ tools
```

The "matter" folder encompasses all the necessary code to support Matter within the SDK. Within the "matter" folder, you'll find the following subfolders:

• **api**: Contains essential functions for working with Matter, the API helps to check device status and get device info such as vendor ID, product ID, etc.
• **common**: Contain common code for various Matter functionalities in the Ameba SDK, covering Bluetooth, Wi-Fi, time management, data storage, and more. "**Include**" folders contain configuration headers for easy project integration.
• **core**: Contains fundamental code for powering Matter in the Ameba SDK's Porting Layer, forming the foundation for seamless Matter integration.
• **docs**: Contains documentation about SDK e.g., general guides, building of SDK, and features.
• **drivers**: Specific drivers enabling different Matter devices to function smoothly with the Ameba SDK, tailored to each device type's unique requirements.
• **examples**: Code snippets and projects demonstrating Matter device usage in the Ameba SDK, organized into subfolders for easy navigation and learning across different device types.
• **project**: Contains projects' makefile and scripts for compiling Matter libraries in the Ameba SDK. It is categorized into builds for different chips, with each subfolder corresponding to a different device type.
• **tools**: Contains several tools that could be useful along the way for building a Matter device.

4 Integrating Ameba Matter Solution into your project

This document provides a step-by-step guide to integrating the ameba-rtos-matter repository into your project. Follow the steps below to successfully integrate and configure the necessary components for your Realtek Matter solution.

Your SDK directory structure should look like the following to ensure proper integration.

```
dev/
└── Ameba SDK
└── connectedhomeip
```

Step 1: Getting ameba-rtos-matter

For AmebaZ2, AmebaZ2plus, AmebaD ➔ **clone_directory**: component/common/application/matter

For AmebaDplus, AmebaLite and AmebaSmart ➔ **clone_directory**: component /application/matter

Method 1: Clone the Repository

To integrate the ameba-rtos-matter repository directly into your project:

```
cd sdk
git clone https://github.com/Ameba-AIoT/ameba-rtos-matter.git <clone_directory>
```

Method 2: Add the Repository as a Submodule

Alternatively, you can add the ameba-rtos-matter repository as a submodule within your existing project.

```
cd sdk
git submodule add https://github.com/Ameba-AIoT/ameba-rtos-matter.git <clone_directory>
```

Afterwards, `git checkout` the appropriate branch or tag that matches your project requirements. E.g.,

```
git checkout release/v1.4
```

Step 2: Creating the third_party Folder for Linking to connectedhomeip

To link the connectedhomeip repository with your sdk, follow these steps to create a third_party folder and establish the necessary symbolic link.

```
//Navigate to the sdk Directory
cd sdk

//Create the third_party Directory
mkdir third_party

//Create the Symbolic Link
cd third_party
ln -s ../../connectedhomeip connectedhomeip

//Checkout the appropriate branch or tag that matches your project requirements
cd connectedhomeip
git checkout v1.4-branch
```

5 Build Environment

This chapter illustrates how to build Realtek's SDK under GCC environment. Ameba's Matter integrated SDK is **ONLY** supported in the GCC environment.
Please follow the guidelines below according to the supported ICs.

Matter's build system has been verified on:

- macOS 10.15,
- Debian 11 (64-bit) and
- Ubuntu 22.04 LTS and above.

We recommend using Ubuntu 20.04 or 22.04 LTS as we have been using this build system for development and have ensured that it works well with the Ameba SDK.

Learn more about Matter environment here.

5.1 Environment Setup Requirements

5.1.1 Hardware

- A Realtek Ameba Board
- USB cable – Micro USB (AmebaD, AmebaZ2, and AmebaZ2plus)
- USB cable – Type C (AmebaDplus, AmebaLite, and AmebaSmart)
- Computer running Ubuntu 22.04 LTS or above (preferred) or Virtual Machine with Ubuntu 22.04 LTS or above.

*Note for Virtual Machine: You can build the Matter environment and Projects on VM but BLE adapter is unavailable so it's impossible to *perform ble-wifi commissioning*, alternatively, you could test with *wifi commissioning*.

5.1.2 Software

- Installing prerequisites on Linux

```
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 the SDK supports CMake build (e.g. ameba-rtos_v1.1), please install additional prerequisites

```
sudo apt-get install cmake curl
```

- Realtek Ameba integrated Matter SDK
- Matter (ConnectedHomeIP) SDK

5.2 Build Project

The following demonstration will be on all-clusters app, please find other applications-built information here.

5.2.1 Building with AmebaZ2 (RTL8720C)

Note: Make sure that the Ameba SDK and connectedhomeip SDK are on the same directory level:

```
dev/
└── Ameba SDK
└── connectedhomeip
```

Step 1: Create a common directory for Ameba and Matter SDK

```
mkdir dev
cd dev
```

Step 2: Git clone repository

```
//Clone Ameba SDK (This is for public repository)
git clone https://github.com/ambiot/ambz2_matter.git or https://github.com/Ameba-AIoT/ameba-rtos-z2

//Clone CHIP SDK
git clone https://github.com/project-chip/connectedhomeip.git

//Execute matter_setup.sh to setup the Ameba SDK with Matter support.
cd Ameba SDK
chmod u+x matter_setup.sh ; ./matter_setup.sh <amebaz2/amebaz2plus> <project type: is/tz>
```

Step 3: Build Matter Environment

```
cd connectedhomeip
git submodule sync
git submodule update --init --recursive
source scripts/bootstrap.sh
source scripts/activate.sh
```

Step 4: Build Matter Libraries and Ameba Project

```
//Navigate to GCC-RELEASE
cd Ameba SDK/project/realtek_amebazXX_v0_example/GCC-RELEASE

//Make Matter Libraries for all-cluster apps
make all_clusters
//Make Ameba Project
make is_matter     //For Non-Trustzone project
make tz_matter     //For Trustzone project

//Clean the whole project
make clean_matter_libs
make clean_matter
```

The image file, “flash_is.bin” is located at **project/realtek_amebaXX_v0_example/GCC-RELEASE/application_is/Debug/bin.**

Note: ameba-rtos-matter repository is only required with the latest Realtek SDK or ameba-rtos-z2 GitHub repository.

5.2.2 Building with AmebaD (RTL8722D)

Note: Make sure that the AmebaD SDK and connectedhomeip SDK are on the same directory level:

```
dev/
└── AmebaD SDK
└── connectedhomeip
```

Step 1: Create a common directory for Ameba and Matter SDK

```
mkdir dev
cd dev
```

Step 2: Git clone repository

```
//Clone AmebaD SDK (This is for public repository)
git clone https://github.com/ambiot/ambd_matter.git or https://github.com/Ameba-AIoT/ameba-rtos-d

//Clone CHIP SDK
git clone https://github.com/project-chip/connectedhomeip.git

//Execute matter_setup.sh to setup the Ameba SDK with Matter support.
cd AmebaD SDK
chmod u+x matter_setup.sh ; ./matter_setup.sh amebad
```

Step 3: Build Matter Environment

```
cd connectedhomeip
git submodule sync
git submodule update --init --recursive
source scripts/bootstrap.sh
source scripts/activate.sh
```

Step 4: Build Matter Libraries and Ameba Project

```
//Navigate to project_lp and 'make all' to build it
cd AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/ ; make all

//Navigate to project_hp
cd AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/

//Enable Matter Configuration using menuconfig, on the next page.
make menuconfig

//Make Matter Libraries for all-cluster apps
make -C asdk all_clusters
//Make Ameba Project
make all
//Clean the whole project
make clean
```

The image files, "km0_boot_all.bin" is located at **project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image**, while the "km4_boot_all.bin" and "km0_km4_image2.bin" is located at **project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/asdk/image.**

Note: ameba-rtos-matter repository is only required with the latest Realtek SDK or ameba-rtos-d GitHub repository.

5.2.2.1 Matter Menuconfig

- Navigate to CONFIG BT, select Enable BT and BT_Matter_Adapter

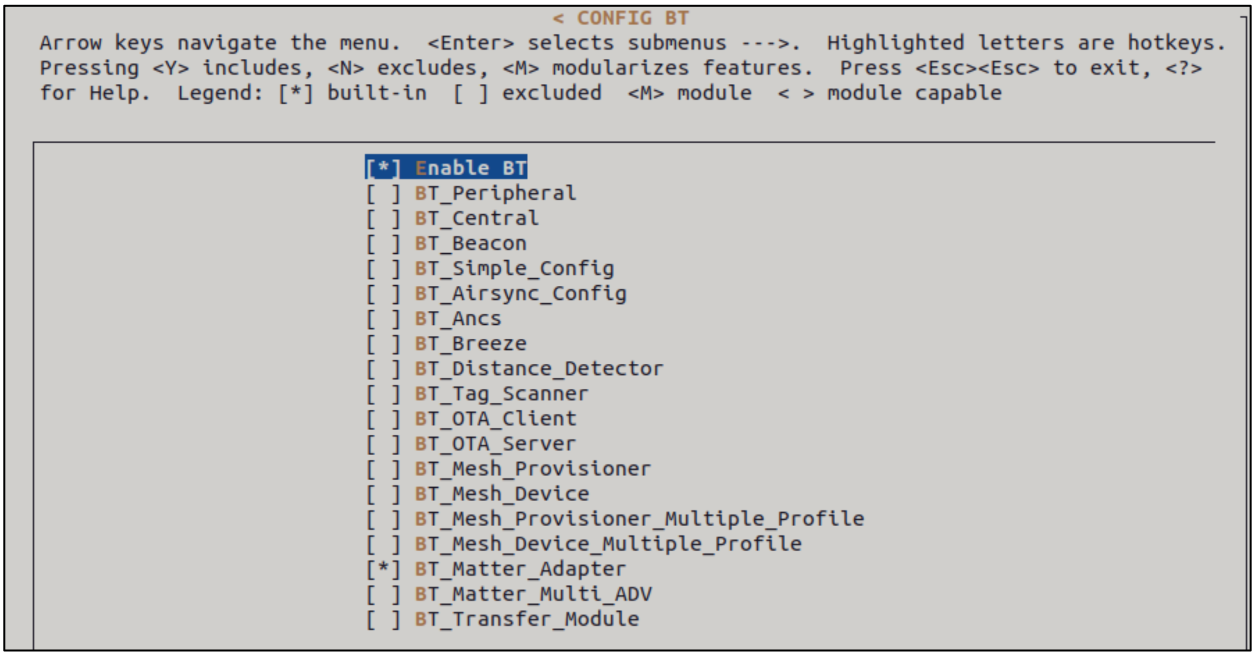

Figure 3: AmebaD Menuconfig enable Matter BLE

- In the SSL Config section, under the MBEDTLS version, select MBEDTLS_MATTER_DEFINED.

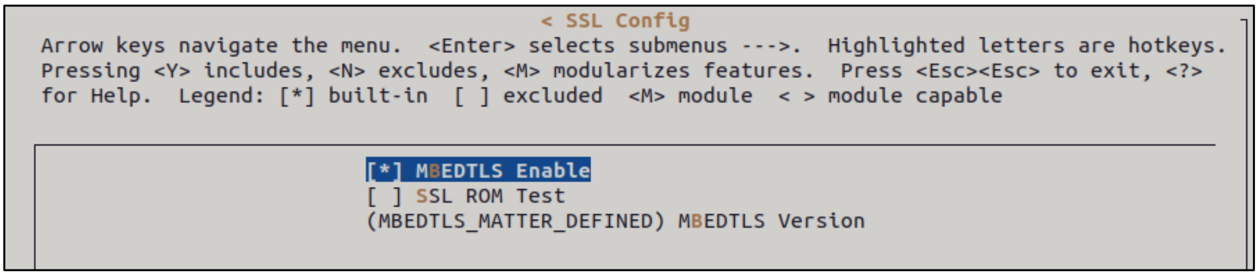

Figure 4: AmebaD Menuconfig Set Matter Mbedtls

- In the Matter Config section, select Enable Matter.

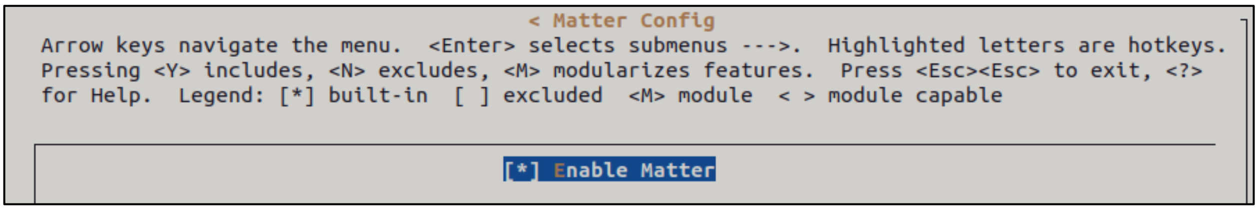

Figure 5: AmebaD Menuconfig Enable Matter

5.2.3 Building with ameba-rtos SDK

Note: Make sure that the ameba-rtos SDK and connectedhomeip SDK are on the same directory level:

```
dev/
└── ameba-rtos SDK
└── connectedhomeip
```

Step 1: Create a common directory for Ameba and Matter SDK

```
mkdir dev
cd dev
```

Step 2: Git clone repository

```
//Clone ameba-rtos (This is for public repository)
git clone https://github.com/Ameba-AIoT/ameba-rtos.git -b release/v1.0+matter

//Clone CHIP SDK
git clone https://github.com/project-chip/connectedhomeip.git

//Execute matter_setup.sh to setup the Ameba SDK with Matter support.
cd ameba-rtos
chmod u+x matter_setup.sh ; ./matter_setup.sh ameba-rtos <matter version: v1.3 / v1.4 / v1.5>
```

Step 3: Setup SDK Environment

ameba-rtos_v1.0 SDK:

```
cd connectedhomeip
git checkout <connectedhomeip commit/branch: 70d9a614 / v1.4-branch / v1.5-branch>
git submodule sync
git submodule update --init --recursive
source scripts/bootstrap.sh
source scripts/activate.sh
```

ameba-rtos_v1.1 SDK:

```
//Setup ameba SDK
cd ameba-rtos
chmod u+x ameba.sh
./ameba.sh

//Setup Matter SDK
cd connectedhomeip
git checkout <connectedhomeip commit/branch: 70d9a614 / v1.4-branch / v1.5-branch>
git submodule sync
git submodule update --init --recursive
source scripts/bootstrap.sh
source scripts/activate.sh
//The next two steps are required to install additional python modules to the connectedhomeip environment
cd ../ameba-rtos
pip install -r tools/requirements.txt
```


5.2.3.1 Makefile build

The Makefile support is available only for ameba-rtos v1.0 SDK.

5.2.3.1.1 Building with AmebaDplus (RTL8721Dx)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebadplus_gcc_project
cd ameba-rtos/amebadplus_gcc_project

//Enable Matter Configuration using menuconfig, on the next page.
make menuconfig

//Make Matter Libraries for all-cluster apps
make -C project_km4/asdk all_clusters

//Make Ameba Project and build the final firmware
make all MATTER_EXAMPLE=chiptest

//Clean the whole project
make clean
```

The image files, “km4_boot_all.bin” and “km0_km4_app.bin”, can be found at **ameba-rtos/amebadplus_gcc_project** or **ameba-rtos/amebadplus_gcc_project/project_km4/asdk/image.**

5.2.3.1.2 Building with AmebaLite (RTL8720EA)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebalite_gcc_project
cd ameba-rtos/amebalite_gcc_project

//Enable Matter Configuration using menuconfig, on the next page.
make menuconfig

//Make Matter Libraries for all-cluster apps
make -C asdk/project_km4 all_clusters

//Make Ameba Project and build the final firmware
make all MATTER_EXAMPLE=chiptest

//Clean the whole project
make clean
```

The image files, “km4_boot_all.bin” and “kr4_km4_app.bin”, can be found at **ameba-rtos/amebalite_gcc_project** or **ameba-rtos/amebalite_gcc_project/project_km4/asdk/image.**

5.2.3.1.3 Building with AmebaSmart (RTL8730E)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebasmart_gcc_project
cd ameba-rtos/amebasmart_gcc_project

//Enable Matter Configuration using menuconfig, on the next page
make menuconfig

//Make Matter Libraries for all-cluster apps
make -C project_ap/asdk all_clusters

//Make Ameba Project and build the final firmware
make all MATTER_EXAMPLE=chiptest

//Clean the whole project
make clean
```

The image files, "km4_boot_all.bin" and "km0_km4_ca32_app.bin", can be found at **ameba-rtos/amebasmart_gcc_project** or **ameba-rtos/amebasmart_gcc_project/project_hp/asdk/image**

5.2.3.1.4 Matter Menuconfig

- In MENUCONFIG FOR KM4 CONFIG (AmebaDplus/AmebaLite) or MENUCONFIG FOR CA32 CONFIG (AmebaSmart) section, under Matter Config, select Enable Matter, choose the Matter Version.
  - Enable Matter Secure may be selected.
  - Enable Matter Terms and Condition (only available for Matter v1.4 onwards) may be selected.

```
                                  < Matter Config
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] built-in  [ ]
excluded  <M> module  < > module capable

                       [*] Enable Matter
                       (MATTER_V_1_4) Matter Version
                       [ ] Enable Matter Terms and Condition
                       [ ] Enable Matter Secure
```

Figure 6: ameba-rtos Makefile Menuconfig enable Matter

- Navigate to CONFIG BT, select Enable BT, enable BT Example Demo, and select BLE_Matter_Adapter.

```
                                   < CONFIG BT
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] built-in  [ ]
excluded  <M> module  < > module capable

                     [*] Enable BT
                     [ ] BT Advanced Feature
                     [*] BT Example Demo
                     [*]    BLE_Matter_Adapter
```

Figure 7: ameba-rtos Makefile Menuconfig enable Matter BLE

- In MENUCONFIG FOR KM4 CONFIG section, under SSL Config, select Matter MBEDTLS Enable.
  - For Matter v1.4, under Matter MBEDTLS Version, select either MBEDTLS_MATTER_V_3_6_1 or MBEDTLS_MATTER_V_2_28_1.

```
                                   < SSL Config
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] built-in  [ ]
excluded  <M> module  < > module capable

                    [*] MBEDTLS Enable
                    [ ] SSL ROM Test
                    [*] Matter MBEDTLS Enable
                    (MBEDTLS_MATTER_V_3_6_1) Matter MBEDTLS Version
```

Figure 8: ameba-rtos Makefile Menuconfig set Matter Mbedtls

5.2.3.2 CMake build

5.2.3.2.1 Building with AmebaDplus (RTL8721Dx)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebadplus_gcc_project
cd ameba-rtos/amebadplus_gcc_project

//Enable Matter Configuration using menuconfig.py, on the next page.
python menuconfig.py

//Build Matter Libraries for all-cluster apps and the final firmware in one command
python build.py -D MATTER_EXAMPLE=all_clusters

//Clean the whole project
cd build && ninja clean_matter_libs clean && cd .. && rm -rf build/
```

The image files, "km4_boot_all.bin" and "km0_km4_app.bin", can be found at **ameba-rtos/amebadplus_gcc_project** or **ameba-rtos/amebadplus_gcc_project/project_km4/asdk/image.**

5.2.3.2.2 Building with AmebaLite (RTL8720EA)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebalite_gcc_project
cd ameba-rtos/amebalite_gcc_project

//Enable Matter Configuration using menuconfig.py, on the next page.
python menuconfig.py

//Build Matter Libraries for all-cluster apps and the final firmware in one command
python build.py -D MATTER_EXAMPLE=all_clusters

//Clean the whole project
cd build && ninja clean_matter_libs clean && cd .. && rm -rf build/
```

The image files, "km4_boot_all.bin" and "kr4_km4_app.bin", can be found at **ameba-rtos/amebalite_gcc_project** or **ameba-rtos/amebalite_gcc_project/project_km4/asdk/image.**

5.2.3.2.3 Building with AmebaSmart (RTL8730E)

Step 1: Prepare ameba-rtos and connectedhomeip SDK
Step 2: Build Matter Libraries and Ameba Project

```
//Navigate to amebasmart_gcc_project
cd ameba-rtos/amebasmart_gcc_project

//Enable Matter Configuration using menuconfig.py, on the next page.
python menuconfig.py

//Build Matter Libraries for all-cluster apps and the final firmware in one command
python build.py -D MATTER_EXAMPLE=all_clusters

//Clean the whole project
cd build && ninja clean_matter_libs clean && cd .. && rm -rf build/
```

The image files, “km4_boot_all.bin” and “km0_km4_ca32_app.bin”, can be found at **ameba-rtos/amebasmart_gcc_project** or **ameba-rtos/amebasmart_gcc_project/project_hp/asdk/image**

5.2.3.2.4 Matter Menuconfig

- Under MENUCONFIG FOR General, enter CONFIG APPLICATION, navigate to Matter Config, select Enable Matter.
    - Enable Matter Secure may be selected.
    - Enable BLE Matter Adapter may be selected
    - Enable Matter Terms and Condition (only available for Matter v1.4 onwards) may be selected.

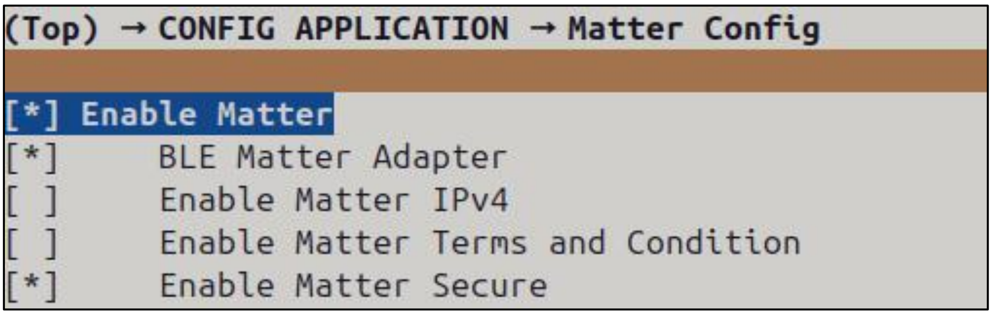

Figure 9: ameba-rtos CMake Menuconfig enable Matter

- To support Matter, LwIP configurations must be modified in the menuconfig. Under Connectivity config, navigate to CONFIG LWIP, Enable LWIP IPv6, and set Len of pbuf pool to 1500.

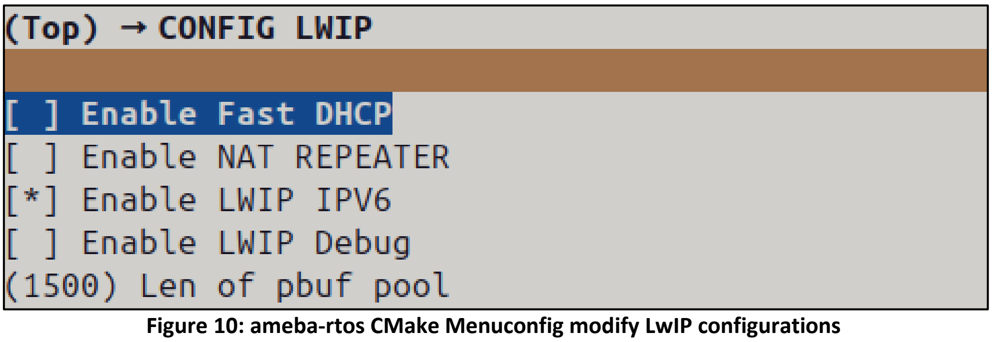

Figure 10: ameba-rtos CMake Menuconfig modify LwIP configurations

- Under Connectivity config, navigate to CONFIG BT, select Enable BT, BLE Matter Adapter shall be enabled under Matter Config (shown in the first point).

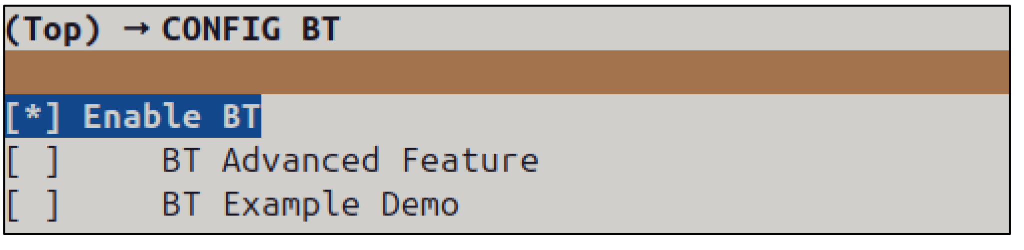

Figure 11: ameba-rtos CMake Menuconfig Enable BT

5.2.4 Matter Sample Application

Each device type has its standalone example, and here is how you can build the application.

5.2.4.1 AmebaZ2/AmebaZ2plus Matter Sample App Build

Under the ***Ameba SDK/project/realtek_amebaXX_v0_example/GCC-RELEASE*** directory, use the following commands for build:

```
cd Ameba SDK/project/realtek_amebaXX_v0_example/GCC-RELEASE

make all_clusters / air_purifier / light / light_switch

make is_matter
```

5.2.4.2 AmebaD Matter Sample App Build

Under the ***AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp*** directory, use the following commands for build:

```
cd AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp

make -C asdk all_clusters / air_purifier / light / light_switch

make all
```

5.2.4.3 Ameba-rtos (AmebaDplus / AmebaLite / AmebaSmart) Matter Sample App Build

5.2.4.3.1 Makefile build

Navigate to the corresponding ameba project, and use the following commands for build

```
cd ameba-rtos/amebaXXX_gcc_project

// For AmebaDplus / AmebaLite to build matter library
make -C project_km4/asdk all_clusters / air_purifier / light

// For AmebaSmart to build matter library
make -C project_ap/asdk all_clusters / air_purifier / light

// For all to build final firmware
make all MATTER_EXAMPLE=chiptest
```

5.2.4.3.2 CMake build

Navigate to the corresponding ameba project, and use the following commands for build

```
cd ameba-rtos/amebaXXX_gcc_project

python build.py -D MATTER_EXAMPLE=example_name

// Available example_name for build
all_clusters / air_purifier / light
```


5.2.5 Porting Layer Matter Sample Application

The porting layer is implemented for more flexible management of code on different Realtek ICs. All the porting layer sample applications can be found under ***component/common/application/matter/examples*** of the AmebaZ2/Z2plus and AmebaD SDK or ***component /application/matter/examples*** of the AmebaDplus, AmebaLite, and AmebaSmart SDK.
Within each example, there is a readme, which shows how the example is being implemented and how it can be build.
Each Sample Application has a macro designated for it.

AmebaZ2/Z2plus and AmebaD SDK
The macro can be found in ***component/common/application/matter/common/include/platform_opts_matter.h***.
Please ensure that **CONFIG_EXAMPLE_MATTER_CHIPTEST** is **disabled**, and the corresponding macro of the sample application has been enabled.
E.g.,

- examples/light : enable **CONFIG_EXAMPLE_MATTER_LIGHT**
- examples/thermostat : enable **CONFIG_EXAMPLE_MATTER_THERMOSTAT**
- examples/refrigerator : enable **CONFIG_EXAMPLE_MATTER_REFRIGERATOR**

AmebaDplus, AmebaLite, and AmebaSmart SDK
The example config macro is automatically defined during Matter compilation and during firmware build if **MATTER_EXAMPLE** is defined **correctly**. Please ensure that **MATTER_EXAMPLE** is defined according to the matter sample code only during firmware compilation. If **MATTER_EXAMPLE** is defined **incorrectly**, the firmware compilation would still be completed successfully because app_example API (***component/application/matter/examples/matter_example_entry.c***) is still compiled and no linking error is thrown. If the image is run, the Matter layer will not be started.

5.2.5.1 AmebaZ2/AmebaZ2plus Porting Layer Matter Sample App Build

Under ***Ameba SDK GCC-RELEASE*** directory, use the following commands for build:

```
cd Ameba SDK/project/realtek_amebaXX_v0_example/GCC-RELEASE

make bridge_dm / dishwasher_port / fan_port / generic_switch_port /laundry_washer_port / light_port / light_dm /
microwave_oven_port / refrigerator_port / room_air_conditioner_port / temperature_sensor_port / thermostat_port

make is_matter
```

5.2.5.2 AmebaD Porting Layer Matter Sample App Build

Under the ***AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp*** directory, use the following commands for build:

```
cd AmebaD SDK/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp

make -C asdk bridge_port / bridge_dm / dishwasher_port / fan_port / generic_switch_port / laundrywasher_port / light_port / light_dm / microwaveoven_port / refrigerator_port / room_air_conditioner_port / temperature_sensor_port / thermostat_port

make all
```

5.2.5.3 Ameba-rtos (AmebaDplus / AmebaLite / AmebaSmart) Porting Layer Matter Sample App Build

5.2.5.3.1 Makefile build

Navigate to the corresponding ameba project, and use the following commands for build

```
cd ameba-rtos/amebaXXX_gcc_project

// For AmebaDplus / AmebaLite to build matter library
make -C project_km4/asdk aircon_port / bridge_port / bridge_dm / dishwasher_port / energy_management_port / fan_port / laundrywasher_port / light_port / light_dm / microwaveoven_port / refrigerator_port / thermostat_port

// For AmebaSmart to build matter library
make -C project_ap/asdk aircon_port / bridge_port / bridge_dm / dishwasher_port / fan_port / laundrywasher_port / light_port / light_dm / microwaveoven_port / refrigerator_port / thermostat_port

// For all to build final firmware
make all MATTER_EXAMPLE=example_name

// Available example_name for build
bridge / bridge_dm / dishwasher / energy_management / fan / laundrywasher / light / light_dm / microwaveoven / refrigerator / thermostat
```

5.2.5.3.2 CMake build

Navigate to the corresponding ameba project, and use the following commands for build

```
cd ameba-rtos/amebaXXX_gcc_project

python build.py -D MATTER_EXAMPLE=example_name

// Available example_name for build
bridge_port / bridge_dm / dishwasher_port / energy_management_port / fan_port / laundrywasher_port / light_port / light_dm / microwaveoven_port / refrigerator_port / thermostat_port
```

5.3 Troubleshooting

When faced with build error, firstly, clean the project and start a fresh build.

Here is a list of commonly encountered build errors:

5.3.1.1 Matter Environment Not Activated

Attempt to build the sdk when the Matter Environment has not been activated, will face the following build error

```
FAILED TO EXECUTE ZAP GENERATION: No such file or directory - "zap-cli"
********************************************************************************
* You may need to install zap. Please ensure one of these applies:
* - `zap-cli` is in $PATH. Install from https://github.com/project-chip/zap/releases
*   see docs/guides/BUILDING.md for details
* - `zap-cli` is in $ZAP_INSTALL_PATH. Use this option if you
*   installed zap but do not want to update $PATH
* - Point $ZAP_DEVELOPMENT_PATH to your local copy of zap that you
*   develop on (to use a developer build of zap)
********************************************************************************
make: *** [Makefile:266: /home/xushuqun/realtek/SVE_SDK/DM8722/ambd_matter/project/realtek_amebaD_va0
_example/GCC-RELEASE/project_hp/asdk/../../../../../third_party/connectedhomeip/examples/all-clusters
-app/ameba/build/chip/codegen/cluster-file.txt] Error 1
```

Figure 12: Environment Build Failure

Simply activate the environment to fix the issue.

```
cd connectedhomeip
source scripts/activate.sh
```

5.3.1.2 Matter Bootstrap or Activate fails

When bootstrap or activate environment fails, remove the environment, and try again.

```
cd connectedhomeip
rm -rf .environment/*
source scripts/activate.sh
```


5.4 Flashing Image

After building the SDK, connect the board via USB and use either the Linux or Windows Image Tool to flash the image file.

5.4.1 Image Tool

5.4.1.1 Linux Image Tool

The Linux Image Tool comes together in the SDK. Find it under the directory ***ameba-rtos-matter/tools/Image_Tool/AmebaX***. Linux Image Tool is only available for AmebaZ2 and AmebaD. Simply enter the following commands on the Linux Terminal.

AmebaZ2 Linux Flash command

```
./flash.sh <Device Port> <Path to flash_is.bin> <address[optional]>
```

AmebaD Linux Flash command

```
./flash.sh <Device Port> <Path to km0_boot_all.bin> <Path to km4_boot_all.bin> <Path to km0_km4_image2.bin>
<address[optional]>
```

5.4.1.2 Windows Image Tool

Alternatively, you can move the image files into a Windows system and use Image Tool to flash the image file. Each IC has their own respective Image tool, for more information on Image Tool please refer to the application notes. [AmebaZ2: AN0500 Chapter 4, AmebaD: AN0400 Chapter 8].

For AmebaDplus, AmebaLite, and AmebaSmart, you may use Windows Image Tool that is provided in the ameba-rtos SDK located in ameba-rtos/tools/ameba/ImageTool. You may also use this Windows Image Tool to flash and erase flash to AmebaD. Please refer to the flashing and erasing flash guide for more details.

5.4.1.3 Python Script Image Tool

This tool is only available on Ameba RTOS v1.1 SDK. Find it under the directory ***ameba-rtos/amebaXXX_gcc_project/flash.py.*** This tool can be run on both Linux and Windows platform. Simply run the following commands on either Linux terminal or Windows command prompt

AmebaDplus/AmebaLite Linux Flash Command

```
python flash.py -p <Device Port> -i <Path to km4_boot_all.bin> 0x08000000 0x08014000 -i <Path to km0_km4_app.bin or
kr4_km4_app.bin> 0x08014000 0x08200000
```

AmebaSmart Linux Flash Command

```
python flash.py -p <Device Port> -i <Path to km4_boot_all.bin> 0x08000000 0x08020000 -i <Pth to km0_km4_ca32_app.bin>
0x08020000 0x08300000
```

To see more details about this tool, please run python flash.py --help.

6 Ameba Matter AT Commands

This page provides detailed instructions for using AT commands on Realtek devices.
To view all available Matter AT commands, enter:

```
ATMH
```

Matter AT Commands List

Command	Description	Usage
ATM$	Factory reset	ATM$
ATM%	Matter OTA query image	ATM%
ATM^	Matter OTA apply update	ATM^
ATMH	Matter help	ATMH
ATMS	Matter client console	ATMS=switch / ATMS=manual

Note: On AmebaD, replace the = sign with a space when using the Matter client console. Example:

```
ATMS switch onoff on
```

Diagnostic Logs

Currently, diagnostic logs are supported only on AmebaZ2 and AmebaZ2plus.
These commands allow testing by inserting test data, which can then be retrieved by the controller.

Command	Usage	Description / Status
$$$$	$$$$=1025	Get user log, status = 0
$$$$	$$$$=1024	Get user log, status = 1
^^^^	^^^^=1025	Get network log, status = 0
^^^^	^^^^=1024	Get network log, status = 1
####	####	Get crash log, status = 0
@@@@	@@@@	Get crash log, status = 1

Notes:

- Status codes indicate log retrieval status:
    - 0: When logs retrieved is more than 1025 bytes
    - 1: When logs retrieved is lesser than or equals to 1024 bytes
    - 2: No logs inserted

7 Matter Device Configuration

7.1 Matter BLE

Matter BLE is a feature that enables Bluetooth Low Energy (BLE) support for Matter devices, allowing communication over BLE for device pairing. To enable or disable Matter BLE functionality:

- In **Makefile** build**,** modify the ***Makefile.include.matter*** by setting the **CHIP_DEVICE_CONFIG_ENABLE_CHIPOBLE** configuration option (E.g., CFLAGS += -DCHIP_DEVICE_CONFIG_ENABLE_CHIPOBLE=**1**).
- In **CMake** build, modify the **menuconfig** by setting the **BLE Matter Adapter** configuration option.

7.2 Matter IPv4

By default, Ameba only enables Matter with IPv6 support, allowing it to accept only IPv6 mDNS packets. To enable or disable Matter IPv4 support:

- In **Makefile** build, modify the ***Makefile.include.matter*** by setting the **INET_CONFIG_ENABLE_IPV4** configuration option (E.g., CFLAGS += -DINET_CONFIG_ENABLE_IPV4=**1**).
- In **CMake** build, modify the **menuconfig** by setting the **Enable Matter IPv4** configuration option.

7.3 Matter Factory Data

For an explanation of Matter Factory Data, refer to this chapter. To enable or disable Matter Factory Data feature:

- I in **Makefile** build, in ***Makefile.include.matter*** set **CONFIG_ENABLE_AMEBA_FACTORY_DATA** configuration option (E.g., CFLAGS += -DCONFIG_ENABLE_AMEBA_FACTORY_DATA=**1**).
- In **CMake** build, in ***cmake/flags/public_definitions_matter.cmake*** set **CONFIG_ENABLE_AMEBA_FACTORY_DATA** configuration option (E.g., CONFIG_ENABLE_AMEBA_FACTORY_DATA=**1**).

7.4 Matter OTA Requestor

Matter OTA Requestor initiates Over-the-Air (OTA) firmware update requests.
To enable or disable Matter OTA Requestor feature:

- In **Makefile** build, in ***Makefile.include.matter*** set **CONFIG_ENABLE_OTA_REQUESTOR** configuration option (E.g., CFLAGS += -DCONFIG_ENABLE_OTA_REQUESTOR=**1**).
- In **CMake** build, in ***cmake/matter.cmake*** set **CHIP_ENABLE_OTA_REQUESTOR** configuration option (E.g., set(CHIP_ENABLE_OTA_REQUESTOR **ON**)).

8 Matter Tools Generation

In this section, let's talk about how to generate Matter tools which are used for commissioning, modifying of zap, generation of certificates chain and QR codes.

8.1 Chip Tool

Chip-Tool is a simulator of Matter Commissioner, which is to send messages to a Matter Commissionee. The purpose of the Matter commissioner is to commission the Matter Commissionee into the Matter fabric, as well as controlling the Matter Commissionee to read or write commands.

Generate the Chip-Tool using the following command:

```
cd connectedhomeip
source scripts/activate.sh
scripts/examples/gn_build_example.sh examples/chip-tool SOME-PATH/

e.g.
scripts/examples/gn_build_example.sh examples/chip-tool out/chip-tool

(Afterwards you should be able to find chip-tool in connectedhomeip/out/chip-tool directory)
```

8.2 Chip Cert and spake2p

CHIP Certificate Tool (chip-cert) is used to generate, convert, validate, resign of Matter certificate. Meanwhile, spake2p tool is used for generating spake parameters for device manufacturing provisioning.

Build chip-cert tool:

```
cd connectedhomeip
source scripts/activate.sh
cd src/tools/chip-cert
gn gen out
ninja -C out
```

Build spake2p tool:

```
cd connectedhomeip
source scripts/activate.sh
cd src/tools/spake2p
gn gen out
ninja -C out
```

8.3 QR Code Generator Tool

Matter has also provided a QR Code generation Tool for mass production in batch.

Step 1: Generate the QR Code tool using the following command:

```
cd connectedhomeip
source scripts/activate.sh
gn gen out/qrcode
ninja -C out/qrcode qrcodetool
```

Step 2: Afterwards, create an input file:

```
cd out/qrcode
touch input.txt
```

Step 3: Enter the following information into the input file:

```
version 0
vendorID <vendor_id> <e.g., 0xFFF1>
productID <product_id> <e.g., 0x8001>
commissioningFlow <commissioning_flow> <e.g., Standard = 0, UserIntent = 1, Custom = 2>
rendezVousInformation <discovery_mode> <e.g., SoftAP = 1, BLE = 2, OnNetwork = 3>
discriminator <discriminator> <e.g., 3840>
setUpPINCode <passcode><e.g., 20202021>
```

Step 4: Run the QR Code Tool:

```
./obj/src/qrcodetool/bin/qrcodetool generate-qr-code input.txt
```

Step 5: Modify the source code in ***connectedhomeip src/qrcodetool/setup_payload_commands.cpp***.

```
@@ -37,22 +37,7 @@ static std::string _extractFilePath(int argc, char * const * argv)
    {
       return path;
    }
-   int ch;
-   const char * filePath = nullptr;
-
-   while ((ch = getopt(argc, argv, "f:")) != -1)
-   {
-    ... //remove all the code in while loop
-   }
+    const char * filePath = argv[1];
    return std::string(filePath);
 }
```

8.4 Zap Tool

ZAP contains the Zigbee Cluster Library (ZCL) developed by Connectivity Standards Alliance (CSA). The Zap Tool generates a user interface to modify these libraries to support different types of Devices (e.g., clusters and attributes).

Generate the Zap-Tool using the following command:

```
cd connectedhomeip
source scripts/activate.sh
scripts/tools/zap/run_zaptools.sh <PATH_TO_ZAP_FILE>
```

Where:

- You can build a fresh zap file by running the zap tool without including <PATH_TO_ZAP_FILE>.
- Alternatively, you can use any existing zap files and modify it for your product.
- For instance, if you are supporting lighting application, the <PATH_TO_ZAP_FILE> you can use is ***examples/lighting-app/lighting-common/lighting-app.zap***.

9 Commissioning

Matter Commissioner will issue command to add Matter Commissionee into the Matter fabric, Matter Commissioner can be Chip-Tool, Google Nest Hub, Apple Homekit or Amazon Echo Dot.

9.1 Commissioning methods

Depending on the supported transportation, Ameba only supports the following commissioning methods.

9.1.1 BLE Commissioning

Using chip-tool to commission the device:

```
./chip-tool pairing ble-wifi <NODE_ID> <SSID> <PASSWORD> <SETUP_CODE> <DISCRIMINATOR>
```

Where:

- The <NODE_ID> is an id assigned to the node being commissioned.
- <SSID> and <PASSWORD> is the Wi-Fi SSID and Password.
- The default <SETUP_CODE> is 20202021.
- The <DISCRIMINATOR> is used to distinguish between advertising devices, default is 3840.
- E.g., **./chip-tool pairing ble-wifi 0x1234 MySSID MyPassword 20202021 3840**

9.1.2 Wi-Fi (IP) Commissioning

There are several ways to pair a device over IP. First, the device must be connected to the network, and use one of the following commands to add device into the Matter fabric.

<u>Method 1</u>: Discover devices with setup code.

```
chip-tool pairing onnetwork <NODE_ID> 20202021
```

<u>Method 2</u>: Discover devices with discriminator and pair with setup code.

```
chip-tool pairing onnetwork-long <NODE_ID> 20202021 3840
```

<u>Method 3</u>: Discover and pair devices with QR code.

```
chip-tool pairing code <NODE_ID> MT:xxxxxx
```

Where QR code can be found in the Ameba console log:

```
chip[SVR] SetupQRCode: [MT:-24J042C00KA0648G00]
```


9.1.3 Troubleshooting

When commissioning fails, please try to do a Factory Reset on the board and try again. Here is a list of commonly encountered issues for commissioning:

9.1.3.1 Controller unable to scan BLE connection

There might be chances that the controller is unable to scan any BLE connection and display the log as follow:

```
[1687504368.666587][239442:239443] CHIP:BLE: New device scanned: F5:54:69:F6:F5:34
[1687504368.666635][239442:239443] CHIP:BLE: Device discriminator match. Attempting to connect.
[1687504368.669792][239442:239443] CHIP:BLE: Scan complete notification without an active scan.
[1687504378.674316][239442:239444] CHIP:DL: HandlePlatformSpecificBLEEvent 16386
[1687504378.674339][239442:239444] CHIP:DIS: Closing all BLE connections
[1687504378.674406][239442:239443] CHIP:DL: FAIL: ConnectDevice : Operation was cancelled (19)
[1687504378.674501][239442:239444] CHIP:DL: HandlePlatformSpecificBLEEvent 16386
```

Figure 13: BLE scan failure

This could be due to controller's Bluetooth adapter or Ameba's Bluetooth did not start.
Try to turn off and on the controller's Bluetooth, and likewise try to do a factory reset on Ameba.

[Note: Ameba's Bluetooth will ONLY start when it is un-provisioned.]

9.1.3.2 Commissioning fails due to unparalleled network connection

If the commissioner is on Network A and is provisioning Ameba onto Network B, the commissioning will fail after Ameba connects to Network B. This is because Ameba is advertising mDNS on a different network than the commissioner.

```
[1687504137.930285][237792:237794] CHIP:DMG: ICR moving to [AwaitingDe]
[1687504138.130568][237792:237794] CHIP:DIS: Checking node lookup status after 200 ms
[1687504140.494051][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504142.970266][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504142.970359][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504145.534068][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504148.010485][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504148.010591][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504150.574004][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504153.049398][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504153.049513][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504155.614539][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504158.090465][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504158.090570][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504160.654391][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504163.130460][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504163.130560][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504165.694452][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504168.170384][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504168.170486][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504168.935540][237792:237794] CHIP:DIS: Timeout waiting for mDNS resolution.
[1687504170.734187][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504173.210445][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504173.210550][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504175.774378][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504178.250423][237792:237793] CHIP:DL: Indication received, conn = 0x7fd74c05b2c0
[1687504178.250562][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16389
[1687504180.814024][237792:237794] CHIP:DL: HandlePlatformSpecificBLEEvent 16387
[1687504182.930512][237792:237794] CHIP:DIS: Checking node lookup status after 45000 ms
```

Figure 14: Unparallel network phenomenon

And eventually the controller will timeout

```
[1687504182.935703][237792:237792] CHIP:DL: System Layer shutdown
[1687504186.126848][237792:237792] CHIP:TOO: Run command failure: ../../examples/chip-tool/third_part
y/connectedhomeip/src/lib/address_resolve/AddressResolve_DefaultImpl.cpp:114: CHIP Error 0x00000032:
Timeout
```

Figure 15: Unparallel network failure timeout

In this case, change the controller's network, or change the commission command's SSID and password.

10 Controlling commands

10.1 Cluster Control

Below are a few control commands that can be used for controlling the device.

10.1.1 OnOff Commands

```
./chip-tool onoff on <node_id> <endpoint>
./chip-tool onoff off <node_id> <endpoint>
./chip-tool onoff toggle <node_id> <endpoint>
```

10.1.2 ColorControl Commands

```
./chip-tool colorcontrol read current-hue <node_id> <endpoint>
./chip-tool colorcontrol read current-saturation <node_id> <endpoint>
./chip-tool colorcontrol read color-temperature-mireds <node_id> <endpoint>
```

10.1.3 LevelControl Commands

```
./chip-tool levelcontrol read on-off-transition-time <node_id> <endpoint>
./chip-tool levelcontrol read on-off-transition-time <node_id> <endpoint>
./chip-tool levelcontrol read on-level <node_id> <endpoint>
```

10.1.4 Thermostat Commands

```
./chip-tool thermostat read local-temperature <node_id> <endpoint>
./chip-tool thermostat read outdoor-temperature <node_id> <endpoint>
./chip-tool thermostat read occupied-cooling-setpoint <node_id> <endpoint>
./chip-tool thermostat read occupied-heating-setpoint <node_id> <endpoint>
```


10.1.5 Binding and Controlling a Device

Here is how to bind a Switch Device to a Controllee Device and control it through the Matter Shell. One binding client (Switch Device) and one binding server (Controllee) is required.

Commission the switch (nodeID 1) and controllee device (nodeID 2) using chip-tool.

```
./chip-tool pairing ble-wifi 1 <SSID> <PASSWORD> 20202021 3840
./chip-tool pairing ble-wifi 2 <SSID> <PASSWORD> 20202021 3840
```

After successful commissioning, configure the ACL in the controllee device to allow access from switch device and chip-tool.

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

Bind the endpoint 1 OnOff cluster of the controllee device to the switch device.

```
./chip-tool binding write binding '[{"fabricIndex": 1, "node":2, "endpoint":1, "cluster":6}]' 1 1
```

Send OnOff command to the device through the switch device's Matter Shell

```
ATMS=switch onoff on
ATMS=switch onoff off
```

Bind more than one cluster to the switch device. Below command binds the Identify, OnOff, LevelControl, ColorControl and Thermostat clusters to the switch device.

```
./chip-tool binding write binding '[{"fabricIndex": 1, "node":2, "endpoint":1, "cluster":3}, {"fabricIndex": 1, "node":2, "endpoint":1, "cluster":6}, {"fabricIndex": 1, "node":2, "endpoint":1, "cluster":8}, {"fabricIndex": 1, "node":2, "endpoint":1, "cluster":768}, {"fabricIndex": 1, "node":2, "endpoint":1, "cluster":513}]' 1 1
```

After binding the clusters, you may send these cluster commands to the controllee device through the switch device's Matter Shell. Follow the format shown in the description of the commands.

```
ATMS=switch onoff on
ATMS=switch levelcontrol movetolevel 100 0 0 0
ATMS=switch colorcontrol movetohue 100 0 0 0 0
ATMS=switch thermostat SPRL 0 0
```

You may also request to read cluster attributes from Matter Shell

```
ATMS=switch <cluster> read <attribute>
```

Note: please make sure to replace the '=' sign with space when using matter client console on AmebaD e.g., ATMS switch onoff on.

11 Matter OTA

Over-the-air programming (OTA) provides a methodology to update device firmware remotely via IP-layer network transmission.

11.1 Building Matter OTA Firmware

When building the OTA firmware, ensure that the software version and software version string is higher than that of the current image. First, check the current image's software version using `chip-tool` commands below.

```
./chip-tool basicinformation read software-version 1 0
./chip-tool basicinformation read software-version-string 1 0
```

Afterwards, define CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION and CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING in ***connectedhomeip/src/platform/Ameba/CHIPDevicePlatformConfig.h***

For instance, if the current software version is 1, for the new firmware set it as 2:

```
#define CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION 2
#define CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING “2.0”
```


11.1.1 Building AmebaZ2 / AmebaZ2plus Matter OTA Firmware

Step 1: In ***GCC-Release/amebaXX_firmware_is.json***, update the serial value to a higher number than the current image.

```
"FWHS": {
        "source":"application_is/Debug/bin/application_is.axf",
        "header":{
                "next":null,
                "__comment_type":"Support
Type:PARTAB,BOOT,FWHS_S,FWHS_NS,FWLS,ISP,VOE,WLN,DTCM,ITCM,SRAM,ERAM,XIP,M0,CPFW",
                "type":"FWHS_S",
                "enc":false,
      "user_key2":"BB0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F",
                        "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 100
        },
```

Step 2: After building the firmware, use the \`ota_image_tool.py\` in ***tools\matter\ota*** to generate the OTA image. This tool will add Matter OTA header to the firmware image.

```
python3 ota_image_tool.py create -v <VENDORID> -p <PRODUCTID> -vn <VERSION> -vs <VERSIONSTRING> -da <DIGESTALGO>
<path to firmware> <output ota image>
```

This is an example:

```
cd ameba-rtos-matter/tools /ota

python3 ota_image_tool.py create -v 0x8888 -p 0x9999 -vn 2 -vs 2.0 -da
sha256 ../../../project/realtek_amebaXX_v0_example/GCC-RELEASE/application_is/Debug/bin/firmware_is.bin ota_image.bin
```

Ensure that the \`VERSION\` and \`VERSIONSTRING\` matches your \`CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION\` and \`CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING\` respectively.

11.1.2 Building AmebaD Matter OTA Firmware

Step 1: After building a firmware use `python_custom_ecdsa_D_gcc.py` in ***tools\matter\ota*** to add AmebaD OTA header.

```
python3 python_custom_ecdsa_D_gcc.py <path to km0_km4_image2.bin> <output image with AmebaD header>
```

Step 2: After adding Ameba OTA header to the image, use the `ota_image_tool.py` in ***tools\matter\ota*** to generate the OTA image. This tool will add Matter OTA header to the image.

```
python3 ota_image_tool.py create -v <VENDORID> -p <PRODUCTID> -vn <VERSION> -vs <VERSIONSTRING> -da <DIGESTALGO>
<path to firmware> <output ota image>
```

This is an example:

```
cd ameba-rtos-matter/tools /ota

python3 python_custom_ecdsa_D_gcc.py ../../../project/realtek_amebaD_va0_example/GCC-
RELEASE/project_hp/asdk/image/km0_km4_image2.bin ota_firmware.bin

python3 ota_image_tool.py create -v 0x8888 -p 0x9999 -vn 2 -vs 2.0 -da sha256 ota_firmware.bin ota_image.bin
```

Ensure that the `VERSION` and `VERSIONSTRING` matches your `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` and `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING` respectively.

11.1.3 Building AmebaDplus / AmebaLite / AmebaSmart Matter OTA Firmware

Step1: Update image version in the manifest file

For ameba-rtos_v1.0: In ***ameba-rtos/amebaXXX_gcc_project/manifest.json***, update the IMG_VER_MAJOR or/and IMG_VER_MINOR of “app” value to a higher number than the current image.

```
"//": "cert/app share IMG_ID/IMG_VER, rdp img is in app",
"app":
{
	"IMG_ID": "1",
	"IMG_VER_MAJOR": 1,
	"IMG_VER_MINOR": 1,
	"SEC_EPOCH": 1,

	"HASH_ALG": "sha256",

	"RSIP_IV": "213253647586a7b8",

	"algorithm":"ed25519",
	"private key":"9FC60C4CB6162E49C54FB94511497E16F5EB605167836F15DECBB8363B18E243",
	"public key":"45BABD665908AD5576D9DA4A80287262A4CB1475ACAD543E27A9C2598579FC72",
	"public key hash":"1DFC2FE01CA8274F06E2E112D027C3C6FF9CED59EE79944BED46ADE35C44B422"
},
```

For ameba-rtos_v1.1: In ***ameba-rtos/amebaXXX_gcc_project/manifest.json5***, update the img_ver_major or/and img_ver_minor of image2 value to a higher number than the current image.

```
image2: {
  img_id: 1,
  img_ver_major: 1,
  img_ver_minor: 1,
  huk_epoch: 1,
  rsip_iv: "213253647586a7b8",
  sboot_private_key: "9FC60C4CB6162E49C54FB94511497E16F5EB605167836F15DECBB8363B18E243",
  sboot_public_key: "45BABD665908AD5576D9DA4A80287262A4CB1475ACAD543E27A9C2598579FC72",
  sboot_public_key_hash: "1DFC2FE01CA8274F06E2E112D027C3C6FF9CED59EE79944BED46ADE35C44B422",
},
```


Step 2: After building the firmware, use the `ota_image_tool.py` in ***tools\matter\ota*** to generate the OTA image. This tool will add Matter OTA header to the firmware image.

```
python3 ota_image_tool.py create -v <VENDORID> -p <PRODUCTID> -vn <VERSION> -vs <VERSIONSTRING> -da <DIGESTALGO>
<path to firmware> <output ota image>
```

This is an example:

```
cd ameba-rtos-matter/tools/ota

python3 ota_image_tool.py create -v 0x8888 -p 0x9999 -vn 2 -vs 2.0 -da sha256 ../../../../../amebaXXX_gcc_project/project_xx/
/image/xxx_app.bin ota_image.bin
```

Ensure that the `VERSION` and `VERSIONSTRING` matches your `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION` and `CHIP_DEVICE_CONFIG_DEVICE_SOFTWARE_VERSION_STRING` respectively.

11.1.4 Validate OTA Image

To check your OTA image, enter the following command:

```
python3 ota_image_tool.py show <ota image>
```


11.2 Matter Over-the-Air (OTA) Firmware Update

11.2.1 Prerequisites

Build Linux ota-provider-app:

```
cd connectedhomeip
source scripts/activate.sh
./scripts/examples/gn_build_example.sh examples/ota-provider-app/linux/ ota-provider
```

Copy the OTA image in this chapter to the directory of the ota-provider built from the previous step.

```
cp ota_image.bin <path to the ota-provider directory>
```

11.2.2 Executing OTA

Open two terminal: one for Linux ota-provider-app and another for chip-tool

Step 1: In Terminal 1 (Linux ota-provider-app), start the OTA provider.

```
cd connectedhomeip/ota-provider
./chip-ota-provider-app -f ota_image.bin
```

Step 2: In Terminal 2 (chip-tool), commission the device and OTA provider into the same Fabric.

Pair the device on NodeID=1 and pair the ota-provider-app on NodeID=2.

```
./chip-tool ble-wifi 1 <SSID> <PASSWORD> 20202021 3840
./chip-tool pairing onnetwork 2 20202021
```

Step 3: Set the ota-provider to be the default OTA provider of the device.

```
./chip-tool otasoftwareupdaterequestor write default-otaproviders '[{"fabricIndex": 1, "providerNodeID": 2, "endpoint": 0}]' 1 0
```

Step 4: Configure the ACL of the ota-provider-app to allow access for device.

```
./chip-tool accesscontrol write acl '[{"fabricIndex": 1, "privilege": 3, "authMode": 2, "subjects": null, "targets": [{"cluster": 41, "endpoint": null, "deviceType": null}]}]' 2 0
```

Step 5: Announce the ota-provider-app to the device to start the OTA process.

```
./chip-tool otasoftwareupdaterequestor announce-otaprovider 2 0 0 0 1 0
```

The OTA process should have started, and you should observe:

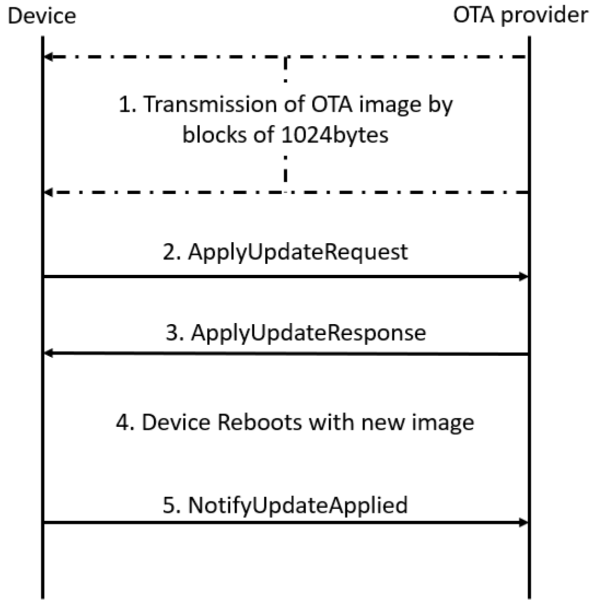

Figure 16: Matter OTA Process

1. The ota-provider-app will transfer the OTA image by blocks of 1024 bytes.
2. After the full transmission of image is completed, the device will send an `ApplyUpdateRequest` message to the ota-provider-app.
3. In return, the ota-provider-app will respond with an `ApplyUpdateResponse`.
4. Upon receiving the `ApplyUpdateResponse`, the device will countdown 10 seconds before rebooting.
5. If the OTA process is successful, the device will reboot into the new image and will send a `NotifyUpdateApplied` to the ota-provider-app.

11.2.3 Troubleshooting

11.2.3.1 Mismatch VendorID and ProductID

If the VendorID and ProductID of the new OTA image does not match the ones in the Basic Information cluster, OTA will fail. Check your VendorID and ProductID using the `chip-tool` commands.

```
./chip-tool basicinformation read vendor-id 1 0
./chip-tool basicinformation read product-id 1 0
```

11.2.3.2 Software Version and serial value lower than current image

The software version of the new OTA image is not higher than the current image's version will affect the OTA process. OTA will only be conducted when the new image is newer than the current image, this will be determined by the software version.

12 Matter Multi-Fabric

A Node can be commissioned into multiple separately managed ecosystem, and this is known as Multi-Fabric. A tutorial on Multi-Fabric features is provided below.

Step 1: Pair the device with the first commissioner/admin using ble-wifi or onnetwork.

```
./chip-tool pairing ble-wifi 1234 SSID PASSWORD 20202021 3840
```

Step 2: The first admin will need to send a command to open the commissioning window of the device and the second admin can commission the device into its fabric.
There are two types of methods to open the commissioning window:

- Basic Commissioning Method (BCM) - Use existing manual pairing code (20202021).
- Enhanced Commissioning Method (ECM) - Use new randomly generated manual pairing code.

BCM Command:

```
// Admin 1 (Fabric 1) opens a commissioning window
./chip-tool administratorcommissioning open-basic-commissioning-window 500 1234 0 --timedInteractionTimeoutMs 1000

// Admin 2 (Fabric 2) pair with the device
./chip-tool pairing onnetwork 1234 20202021 --commissioner-name beta
```

ECM Command:
After Admin 1 has opened a commissioning window, you can find the newly generated manual pairing code in the chip-tool logs which Admin 2 uses it to commission the device: e.g., **CHIP:CTL: Manual pairing code: [35407541839]**

```
// Admin 1 (Fabric 1) opens a commissioning window
./chip-tool pairing open-commissioning-window 1234 1 400 2000 3840

// Admin 2 (Fabric 2) pair with the device using the Manual Pairing Code generated by // Admin 1
./chip-tool pairing code 1234 35407541839 --commissioner-name beta
```

Other useful commands:

```
// To check the fabric list of the device
./chip-tool operationalcredentials read fabrics 1234 0 --fabric-filtered 0

// To remove device from fabric 2
//./chip-tool operationalcredentials remove-fabric 2 1234 0
```

The `--commissioner-name` argument is only needed if you are using the same Chip-Tool instance to pair the device. For instance, opening multiple terminals in a single Ubuntu machine. In this case, use the `--commissioner-name` argument when paring of device and in all control commands

It is unnecessary to add the `--commissioner-name` argument if you are using different chip-tool instance such as different Ubuntu machine for pairing.

This is an example when using one Chip-Tool to execute Multi-Fabric:

```
// Admin1: Pairing and control with NodeID 1234
./chip-tool pairing ble-wifi 1234 SSID PASSWORD 20202021 3840
./chip-tool onoff on 1234 1

// Admin2: Pairing and control with NodeID 1235
./chip-tool pairing code 1235 35407541839 --commissioner-name beta
./chip-tool onoff on 1235 1 --commissioner-name beta

// Admin3: Pairing and control with NodeID 1236
./chip-tool pairing code 1236 35407541839 --commissioner-name gamma
./chip-tool onoff on 1236 1 --commissioner-name gamma
```

The names used for `--commissioner-name` argument are: [alpha, beta, gamma, 4, 5, ...].

13 Matter Key Value Storage (KVS)

Matter KVS is a persistent storage implementation that uses key-value structure.
The types of data that are stored for Matter are:

- Wi-Fi Credential (SSID and Password)
- Fabric information (FabricIndex, FabricNOC, FabricICAC, FabricRCAC, FabricMeta, FabricOpKey)
- Onboarding payload (If factory data is enabled)
- Cluster attribute (In Zap, you can choose to store the attribute in RAM or NVM)

13.1 AmebaZ2 / AmebaZ2plus / AmebaD KVS Implementation

AmebaZ2, AmebaZ2plus and AmebaD utilize Device Configuration Table (DCT) to store data permanently on the flash region.

13.1.1 DCT Modules Calculation

The Module Number are defined in ***platform_opts_matter.h*** as follows:

```
#define MATTER_KVS_MODULE_NUM               13
#define MATTER_KVS_VARIABLE_NAME_SIZE       32
#define MATTER_KVS_VARIABLE_VALUE_SIZE      64+4


#define MATTER_KVS_MODULE_NUM2              10
#define MATTER_KVS_VARIABLE_NAME_SIZE2      32
#define MATTER_KVS_VARIABLE_VALUE_SIZE2     400+4
```

Figure 17: DCT Module Declaration

There are 2 DCT regions, DCT1 and DCT2:

- DCT1 region (MATTER_KVS_MODULE_NUM) has 13 modules for data below 64 bytes (< 64 bytes).
- DCT2 regions (MATTER_KVS_MODULE_NUM2) has 10 modules for 64 bytes and above (>=64 bytes).
- 1 DCT module = 72 Bytes for DCT header + 4024 Bytes for data = 4096 Bytes (1 flash block is 4KB)

<u>How many variables can fit inside one module?</u>

- 1 DCT1 module can fit 4024 bytes / (32+64+4) bytes = 40 variables
- 1 DCT2 module can fit 4024 bytes / (32+400+4) = 9 variables

<u>How to verify Flash space is sufficient for the configuration?</u>

The declaration of MATTER_KVS_BEGIN_ADDR and MATTER_KVS_BEGIN_ADDR2 is in ***platform_opts_matter.h***.
Here is an example of how DCT is configured for Matter, users can adjust according to their needs.

```
#define MATTER_KVS_BEGIN_ADDR                (0x400000 - 0x13000)
#define MATTER_KVS_BEGIN_ADDR2               (0x400000 - 0x1E000)
```

Figure 18: DCT Address Declaration

- Total space available for DCT1 = 0x13000 - 0x5000 = 56KB = 57334 bytes
    - 13 DCT1 modules take up 13 * 4096 = 53248 bytes (Fits)
- Total space available for DCT2 = 0x1E000 - 0x13000 = 44KB = 45056 bytes
    - 10 DCT2 modules take up 10 * 4096 = 24576 bytes (Fits)

13.1.2 DCT Encryption

To enable DCT Encryption, find **CONFIG_ENABLE_AMEBA_DCT_ENC** in ***Makefile.include.matter*** when building the firmware.

13.1.3 DCT Backup and Wear Leveling

DCT stores critical system data in flash memory, but writing data during a system reboot can corrupt the DCT. If corruption occurs, data retrieval may fail on reboot, potentially requiring a full factory reset.

To mitigate this risk, the following options are recommended:

- DCT Backup
  - Creates a redundant copy of the DCT to improve reliability.
  - Note: Enabling DCT Backup requires twice the flash memory since a backup copy is stored alongside the original.
- DCT Wear Leveling
  - Distributes flash writes evenly across the memory to extend flash lifespan.
  - Note: Enabling wear leveling requires six times the module size due to internal data management overhead.

By enabling DCT Backup and/or Wear Leveling, system stability and flash durability can be significantly improved, though at the cost of additional flash usage.

Enabling the Features

To enable these features, edit ***platform_opts_matter.h*** and set the corresponding macros:

```
#define MATTER_KVS_ENABLE_BACKUP            1
#define MATTER_KVS_ENABLE_WEAR_LEVELING     1
```

Setting these macros to 1 enables DCT Backup and Wear Leveling, improving system reliability and flash durability.

13.2 AmebaDplus / AmebaLite / AmebaSmart KVS Implementation

AmebaDplus, AmebaLite, and AmebaSmart utilize Virtual File System (VFS) to store data permanently on the flash region. The ameba-rtos SDK uses the Key Value (KV) API to get and set the data on the VFS.

13.2.1 Ameba VFS and KV

By default, VFS storage capacity at ameba-rtos SDK is set to 128KB (131072 Bytes). It can be adjusted at ***component/soc/ usrcfg /amebaXXX /ameba_flashcfg.c*** (ameba-rtosv 1.1) or ***component/soc/amebaXXX/usrcfg/ameba_flashcfg.c*** (ameba-rtos v1.0) by adjusting the start address and end address of VFS1 region.

```
const FlashLayoutInfo_TypeDef Flash_Layout[] = {
    /*Region_Type,  [StartAddr, EndAddr]        */
    {IMG_BOOT,      0x08000000, 0x08013FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1, 0x08014000, 0x081F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1

    {IMG_BOOT_OTA2, 0x08200000, 0x08213FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K) OTA
    {IMG_APP_OTA2, 0x08214000, 0x083DCFFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2

    {FTL,           0x083DD000, 0x083DFFFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1,          0x083E0000, 0x083FFFFF}, //VFS region 1 (128K)
    {VFS2,          0xFFFFFFFF, 0xFFFFFFFF}, //VFS region 2
    {USER,          0xFFFFFFFF, 0xFFFFFFFF}, //reserve for user

    /* End */
    {0xFF,          0xFFFFFFFF, 0xFFFFFFFF},
};
```

Figure 19: AmebaDplus Flash Layout highlighting VFS1 region

```
FlashLayoutInfo_TypeDef Flash_Layout[] = {
    /*Region_Type,  [StartAddr, EndAddr]        */
    {IMG_BOOT,      0x08000000, 0x08013FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1, 0x08014000, 0x081F3FFF}, //Certificate(4K) + Manifest(4K) + KR4 & KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1

    {IMG_BOOT_OTA2, 0x08200000, 0x08213FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K) OTA
    {IMG_APP_OTA2, 0x08214000, 0x083DCFFF}, //Certificate(4K) + Manifest(4K) + KR4 & KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2
    {FTL,           0x083DD000, 0x083DFFFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1,          0x083E0000, 0x083FFFFF}, //VFS region 1 (128K)
    {IMG_DSP,       0x08400000, 0x086FFFFF}, //Manifest(4K) + DSP IMG, only one DSP region in layout
    {VFS2,          0xFFFFFFFF, 0xFFFFFFFF}, //VFS region 2
    {USER,          0xFFFFFFFF, 0xFFFFFFFF}, //reserve for user

    /* End */
    {0xFF,          0xFFFFFFFF, 0xFFFFFFFF},
};
```

Figure 20: AmebaLite Flash Layout highlighting VFS1 region

```
FlashLayoutInfo_TypeDef Flash_Layout_Nor[] = {
    /*Region_Type,      [StartAddr, EndAddr]        */
    {IMG_BOOT,          0x08000000, 0x0801FFFF}, //Boot Manifest(4K) + KM4 Bootloader(124K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1,      0x08020000, 0x082FFFFF}, //Certificate(4K) + Manifest(4K) + KR4 & KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1
    // + AP IMG OTA1
    {IMG_BOOT_OTA2,     0x08300000, 0x0833FFFF}, //Boot Manifest(4K) + KM4 Bootloader(252K) OTA
    {IMG_APP_OTA2,      0x08340000, 0x0861FFFF}, //Certificate(4K) + Manifest(4K) + KR4 & KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2
    // + AP IMG OTA2
    {FTL,               0x08620000, 0x08622FFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1,              0x08623000, 0x08642FFF}, //VFS region 1 (128K)
    {VFS2,              0xFFFFFFFF, 0xFFFFFFFF}, //VFS region 2
    {USER,              0xFFFFFFFF, 0xFFFFFFFF}, //reserve for user
    /* End */
    {0xFF,              0xFFFFFFFF, 0xFFFFFFFF},
};
```

Figure 21: AmebaSmart NOR Flash Layout highlighting VFS1 region

KV API is used at ***component/application/matter/common/port/matter_kvs.c*** to set and get the Matter persistent data. KV API is declared at ***component/file_system/kv/kv.c*** and additionally at ***component/application/matter/common/port/matter_kv.c***. For more information, please visit the Virtual File System chapter of each SoC Application Note [AmebaDplus: AN1000 Chapter 11, AmebaLite: AN0800 Chapter 11, AmebaSmart: AN0600 Chapter 11].

13.2.2 KV Encryption

To enable KV Encryption in **Makefile** build, find **CONFIG_ENABLE_KV_ENCRYPTION** in **Makefile.include.matter** when building the firmware. To enable KV Encryption in **CMake** build, find **CONFIG_ENABLE_KV_ENCRYPTION** in **cmake/flags/public_definitions_matter.cmake** when building the firmware.

14 Matter Production

14.1 Matter Certificate and Device Onboarding Payload

14.1.1 Certificate and Keys

The Matter certificate chain consists of:

- **Product Attestation Authority (PAA)** which is the root CAs certificate.
- **Product Attestation Intermediate (PAI)** signed by PAA.
- **Device Attestation Credentials (DAC)** signed by PAI.

The **Certificate Declaration (CD)** is a cryptographic document. Once your device has passed the Matter Certification Test, it shall be Matter certified and you will be given a Certification Declaration by CSA.

The DAC and CD must be inserted into the device firmware. During commissioning, these certificates will be used for commissioner to determine if the device is a Matter certified product.

14.1.1.1 Onboarding Payload

Here is a list of mandatory onboarding payload for Matter product:

1. The **Vendor ID (VID)** is a 16-bit number that uniquely identifies a particular product manufacturer or a vendor. It will be allocated to you once you have become a member of CSA.
2. Human-readable **Vendor Name** that provides a simple string containing the identification of the device's vendor for the application and Matter stack purposes.
3. The **Product ID (PID)** is a 16-bit number that uniquely identifies a product of a vendor.
4. Human-readable **Product Name** that provides a simple string containing identification of the product for the application and the Matter stack purposes.
5. The **Setup Passcode** is a 27-bit unsigned integer, which serves as proof of possession during commissioning. The passcode will be restricted to the values 00000001 to 99999998. A corresponding Spake2p verifier will be generated and stored in the device.
6. The **Discriminator** is a12-bit unsigned integer, used to distinguish between advertising devices, and should be different for each individual device.
7. The **Spake2p Parameters** consist of the iteration count, salt, and the verifier. This will be automatically generated and stored in the factory binary data during the generating factory data binary process.
8. The **Hardware Version** number that specifies the version number of the hardware of the device. The value meaning and the versioning scheme is defined by the vendor.
9. The **Hardware Version String** is a more user-friendly representation of the hardware version. The value meaning and the versioning scheme is defined by the vendor.
10. The **Manufacturing Date** that the device was manufactured. Format used is ISO 8601 – YYYY-MM-DD.
11. The **Serial Number** defines a unique number of the manufactured device.
12. The **Rotating Device ID** The unique ID for rotating device ID.
13. The **Unique ID** for rotating device ID.

14.2 Certificate Generation

In this section, the generation of test PAA, PAI, DAC and CD will be shown.
Please take note that this is only for testing and development. During actual production, the certificates and keys must be delivered by a Matter-certified PKI provider for security measure purposes.

Step 1: Generate a Chip-Cert. Please read here about the Chip-Cert generation.

Step 2: Modify the `gen-cert.sh` script located in ***tools/matter/factorydata***

You can decide to generate a new set of PAA certificates and keys or use the existing ones located in *connectedhomeip\credentials\test\attestation*.

To use new generated PAA:

- In Matter's test PAA section, **comment** the variables `paa_key_file` and `paa_cert_file`
- In Self-generated PAA section, **uncomment** out the variables `paa_key_file` and `paa_cert_file`.

```
# Matter's test PAA (uncomment if you want to use Matter's test PAA)
#paa_key_file="$chip_dir/credentials/test/attestation/Chip-Test-PAA-NoVID-Key"
#paa_cert_file="$chip_dir/credentials/test/attestation/Chip-Test-PAA-NoVID-Cert"

# Self-generated PAA
paa_key_file="$dest_dir/Chip-Test-PAA-$vid-Key"
paa_cert_file="$dest_dir/Chip-Test-PAA-$vid-Cert"
```

To use existing PAA:

- In Matter's test PAA section, **uncomment** the variables `paa_key_file` and `paa_cert_file`
- In Self-generated PAA section, **comment** out the variables `paa_key_file` and `paa_cert_file`.

```
# Matter's test PAA (uncomment if you want to use Matter's test PAA)
paa_key_file="$chip_dir/credentials/test/attestation/Chip-Test-PAA-NoVID-Key"
paa_cert_file="$chip_dir/credentials/test/attestation/Chip-Test-PAA-NoVID-Cert"

# Self-generated PAA
#paa_key_file="$dest_dir/Chip-Test-PAA-$vid-Key"
#paa_cert_file="$dest_dir/Chip-Test-PAA-$vid-Cert"
```

You must generate new CD if you are trying to remap manufacturer's VID/PID to end-product's VID/PID. Please configure as follow:

- Ensure that `vid` and `pid` variables are manufacturer's VID and PID respectively.
- Ensure that `endproduct_vid` and `endproduct_pid` are end-product's VID and PID respectively
- Under Generate Credential Declaration section,
    - **Comment** out CD without dac_origin_vid, dac_origin_pid section
    - **Uncomment** CD with dac_origin_vid, dac_origin_pid section


```
# Generate Credential Declaration
cd_signing_key="$chip_dir/credentials/test/certification-declaration/Chip-Test-CD-Signing-Key.pem"
cd_signing_cert="$chip_dir/credentials/test/certification-declaration/Chip-Test-CD-Signing-Cert.pem"

# CD without dac_origin_vid, dac_origin_pid
#"$chip_cert_tool" gen-cd --key "$cd_signing_key" --cert "$cd_signing_cert" --out "$dest_dir/Chip-Test-CD-$vid-$pid.der" --format-version "$format_version" --vendor-id "0x$vid" --product-id "0x$pid" --device-type-id "$device_type_id" --certificate-id "$certificate_id" --security-level "$security_level" --security-info "$security_info" --version-number "$version_num" --certification-type "$certification_type"

# CD with dac_origin_vid, dac_origin_pid
"$chip_cert_tool" gen-cd --key "$cd_signing_key" --cert "$cd_signing_cert" --out "$dest_dir/Chip-Test-CD-$endproduct_vid-$endproduct_pid-WithDACOrigin.der" --format-version "$format_version" --vendor-id "$endproduct_vid" --product-id "$endproduct_pid" --device-type-id "$device_type_id" --certificate-id "$certificate_id" --security-level "$security_level" --security-info "$security_info" --version-number "$version_num" --certification-type "$certification_type" --dac-origin-vendor-id "$vid" --dac-origin-product-id "$pid"
```

Step 3: Run the `gen-cert.sh` script located in ***tools/matter/factorydata***

```
./gen-certs.sh <path to connectedhomeip> <path to chip-cert binary> <c-style filename>
```

Example of command:

```
./gen-certs.sh ../../../third_party/connectedhomeip ../../../third_party/connectedhomeip /src/tools/chip-cert/out/chip-cert output
```

The certificate and keys will be outputted in `***connectedhomeip/myattestation***`.

14.3 Matter Factory Data Binary

14.3.1 Prerequisite

Generate chip-cert and spake2p as mentioned here and install python dependency with this command:

```
pip3 install protobuf==4.21.9
```

14.3.2 Generate Factory Data Binary

Run the ameba_factory.py python script, passing in necessary arguments:

```
cd ameba-rtos-matter/tools/factorydata

python3 ameba_factory.py \
--spake2p_path <path to spake2p binary> \
-d <discriminator> \
-p <passcode> \
--dac_cert <path to DAC cert> \
--dac_key <path to DAC key> \
--pai_cert <path to PAI cert> \
--cd <path to CD> \
--vendor-id <vendor id> \
--vendor-name <vendor name> \
--product-id <product id> \
--product-name <product-name> \
--hw-ver <hardware version> \
--hw-ver-str <hardware version string> \
--mfg-date <manufacturing date> \
--serial-num <serial number> \
--rd-id-uid <rotating id unique id> \
--factorydata-key <32-bytes key to encrypt factorydata, hexstring, without "0x"> \
--factorydata-iv <16-bytes iv to encrypt factorydata, hexstring, without "0x">
```

Example command on running from ***tools/matter/factorydata***:

```
python3 ameba_factory.py \
--spake2p_path ../../../third_party/connectedhomeip/src/tools/spake2p/out/spake2p \
-d 3840 \
-p 20202021 \
--dac_cert ../../../third_party/connectedhomeip/myattestation/Chip-Test-DAC-8888-9999-Cert.der \
--dac_key ../../../third_party/connectedhomeip/myattestation/Chip-Test-DAC-8888-9999-Key.der \
--pai_cert ../../../third_party/connectedhomeip/myattestation/Chip-Test-PAI-8888-NoPID-Cert.der \
--cd ../../../third_party/connectedhomeip/myattestation/Chip-Test-CD-8888-9999.der \
--vendor-id 0x8888 \
--vendor-name ameba \
--product-id 0x9999 \
--product-name amebaz2 \
--hw-ver 1 \
--hw-ver-str "1.0" \
--mfg-date 2022-12-01 \
--serial-num 123456 \
--rd-id-uid 00112233445566778899aabbccddeeff \
--factorydata-key ff0102030405060708090a0b0c0d0e0fff0102030405060708090a0b0c0d0e0f \
--factorydata-iv ff0102030405060708090a0b0c0d0e0f
```

After running the script successfully, ameba_factory.bin should be generated in the same directory. Do take note that the "factorydata-key" and "factorydata-iv" argument is ONLY needed when **CONFIG_ENABLE_FACTORY_DATA_ENC** is enabled, learn more about Factory Data Encryption here.

14.4 Flash Matter Factory Data Binary

After generating the factory data binary, flash the image (ameba_factory.bin) using Image Tool.

14.4.1 Flash Factory Data to AmebaZ2 / AmebaZ2plus / AmebaD

The default address to flash ameba_factory.bin is **0x083FF000**, customer may configure it using **MATTER_FACTORY_DATA** ***platform_opts_matter.h***. Make sure to check for partition conflicts. An example to flash using Image_Tool_Linux is shown below:

```
cd ameba-rtos-matter/tools/Image_Tool/AmebaX
./flash.sh /dev/ttyUSB0 ./ameba_factory.bin 0x083FF000
```

14.4.1.1 Factory Data Encryption

Encryption of Factory Data is also available. If encryption is preferred, follow the steps below.

Step 1: Pass in factorydata-key, if you want to use an IV for encryption, pass in factorydata-iv as well.

Step 2: Make sure that you have enabled **CONFIG_ENABLE_FACTORY_DATA_ENC** in ***Makefile.include.matter*** when building the firmware.

Step 3: Make sure that in DecodeFactory in ***matter_utils.c***, you have implemented a way to retrieve the key and iv for runtime factorydata decryption (By default, it is using a hardcoded key and iv).

14.4.2 Flash Factory Data to AmebaDplus / AmebaLite / AmebaSmart

The default address to flash ameba_factory.bin is **0x08400000** for AmebaDplus/AmebaLite, and **0x08644000** for AmebaSmart. This default address is located right after VFS1 region. Customer may configure it using **MATTER_FACTORY_DATA** macro in ***component /application/matter/common/port/matter_utils.c***. Make sure to check for partition conflicts. An example to flash using Windows Image Tool is shown below:

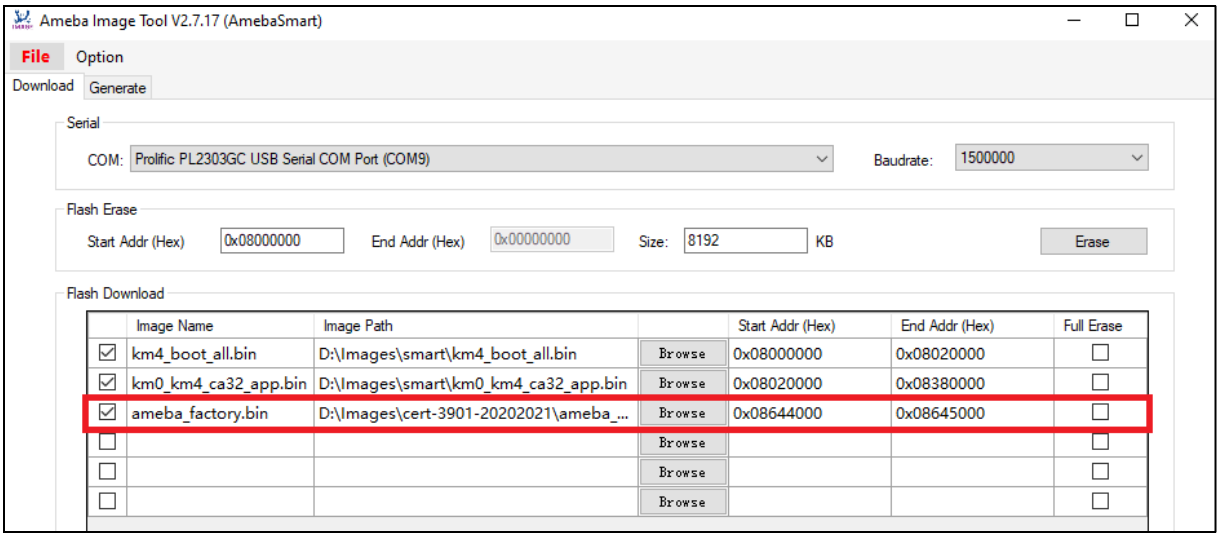

Figure 22 Windows Ameba Image Tool with ameba_factory.bin added for AmebaSmart.

14.5 Generation of Firmware

Make sure your firmware is built with **CONFIG_ENABLE_AMEBA_FACTORY_DATA** enabled in the ***Makefile.include.matter***. Furthermore, you need to check if the VID and PID match the ones in factory data binary.

14.6 Commissioning

If this is for testing and development, during commissioning, pass the path to the PAA as an argument to chiptool.

```
./chip-tool pairing ble-wifi 1 <SSID> <PASSWORD> <passcode> <discriminator> --paa-trust-store-path <path to myattestation>
```

14.7 Hardcoding Onboarding Payload into SDK

An alternative to change the onboarding payload is by hardcoding into the sdk.

In ***connectedhomeip/src/platform/Ameba/CHIPDevicePlatformConfig.h***, add the macros below:

```
#define CHIP_DEVICE_CONFIG_DEVICE_VENDOR_ID 0x8888

#define CHIP_DEVICE_CONFIG_DEVICE_VENDOR_NAME “Realtek”

#define CHIP_DEVICE_CONFIG_DEVICE_PRODUCT_ID 0x9999

#define CHIP_DEVICE_CONFIG_DEVICE_PRODUCT_NAME “AmebaZ2”

#define CHIP_DEVICE_CONFIG_DEFAULT_DEVICE_HARDWARE_VERSION 1

#define CHIP_DEVICE_CONFIG_DEFAULT_DEVICE_HARDWARE_VERSION_STRING "1.0 "

#define CHIP_DEVICE_CONFIG_TEST_SERIAL_NUMBER 123456

#define CHIP_DEVICE_CONFIG_USE_TEST_SETUP_DISCRIMINATOR 3841

#define CHIP_DEVICE_CONFIG_ROTATING_DEVICE_ID_UNIQUE_ID                      \
{                                                \
   0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff \
}
```

In ***connectedhomeip/src/platform/Ameba/FactoryDataProvider.cpp***, hardcode the certificates and key in the buffer.

- kCdForAllExamples[]: Certification Declaration
- kDacCert[]: DAC Certificate
- kPaiCert[]: PAI Certificate
- kDacPublicKey[]: DAC Public Key
- kDacPrivateKey[]: DAC Private Key

15 Matter Certification

For a device to become a Matter certified product, it is necessary to go through the process of Matter Certification. CSA organises Test Event to prepare end-product for certification. The Specification Validation Event (SVE) requires executing and passing all the test cases that the product supports. The actual certification is ONLY granted after Test Houses and CSA has fully reviewed all the test cases result.

15.1 Introduction to Matter Test Harness (TH)

Test Harness (TH) which contains the Test Cases is used on Raspberry Pi for Matter Certification Test. The TH will be more user-friendly as it constructs a GUI for users to pick the Test Cases that they wish to run.
There are several ways to execute the Test Cases:

1. **UI-Automated**: Test Cases are fully operated by TH.
2. **UI-Semi-Automated**: Partially automated Test Cases where several steps require the user to execute commands on Chip-Tool and submit the result on TH.
3. **UI-Manual**: Completely manual operation where all the steps in the Test Case require user to execute commands on Chip-Tool and submit the result on TH.
4. **Verification Steps Document**: Test Cases that are not included in TH should follow the Verification Steps Document.

When using a newly generated PAA, TH must include `--paa-trust-store-path` argument to pair the devices.

Step 1: Update PAA certificates on TH and set **CHIP_TOOL_USE_PAA_CERTS** to true.

```
cd ~/chip-certification-tool
./scripts/stop.sh
./scripts/pi-setup/update-paa-certs.h
rm .env
./scripts/install-default-env.sh
echo "CHIP_TOOL_USE_PAA_CERTS=true" >> .env
./scripts/start.sh
```

Step 2: Copy new PAA certificate to /var/paa-root-certs/

```
sudo cp /path/to/new_PAA_cert /var/paa-root-certs/
```

Step 3: Run any automated chip-tool tests and it will use the `--paa-trust-store-path` argument.

15.2 Introduction to PICS and PICS Tools

The PICS files indicate the Matter features that your device supports and determine the test cases to be tested in Matter Certification Test. The PICS Tool is also available to assist in the selection and validation of XML PICS files.

The reference XML PICS files include all the PICS files of one or several cluster. Upload the XML PICS files to PICS Tool and start selecting the features supported by your device. Click the button `Validate All` on the PICS Tool and a list of Test Cases that must be tested will be generated.

16 OpenThread Border Router (OTBR) Setup

An OTBR interlinks other IP-based networks to a Thread Network, such as Wi-Fi or Ethernet for communication. Read more about OTBR here.

16.1 Hardware Requirements

To test the communication between an IP-based device and thread device, the below equipment is needed.

- 2 Raspberry Pi
- 2 nRF dongle for OTBR

Where one set will be setup as an OTBR and another setup as a Thread device.

16.2 Setup OTBR

Step 1: Program the nRF device with the RCP application, follow the steps here.
Step 2: Connect the nRF dongle to Raspberry Pi and on the Raspberry Pi, build the otbr-posix.

```
git clone https://github.com/openthread/ot-br-posix
cd ot-br-posix
./script/bootstrap
./script/setup
```

Step 3: Start the otbr-posix.

```
sudo ./build/otbr/src/agent/otbr-agent -I wpan0 -B eth0 -v spinel+hdlc+uart:///dev/ttyACM0
```

Where

- '/dev/ttyACM0' is the port of RCP connected to the Raspberry Pi
- 'wpan0' is the thread network interface that will be created.
- 'eth0' is the backbone network interface.

Step 5: Start a new terminal for otbr-posix console to form the Thread network and get dataset.

```
sudo ot-ctl
> ifconfig up
> thread start
> dataset active -x
```

Record down the dataset set as it will be used to join the Border Router's network on otcli-posix.

16.3 Setup Thread End Device

With the other set of nRF dongle with RCP application and Raspberry Pi, let's build a Thread end device.
On the Raspberry Pi, build the ot-cli.

```
git clone --recursive https://github.com/openthread/openthread.git
cd openthread/
./script/bootstrap
./bootstrap
./script/cmake-build posix
./build/posix/src/posix/ot-cli 'spinel+hdlc+uart:///dev/ttyACM0?uart-baudrate=115200' -v
```

Where '/dev/ttyACM0' is the port of RCP connected to the Raspberry Pi.

The ot-cli app will be launched with RCP, now let's connect the RCP to the thread network hosted by Raspberry Pi (Border Router).

```
> dataset set active <PROVIDE THE DATASET OF THE BR THAT YOU NEED TO JOIN i.e BR hosted in TH>
> dataset commit active
> ifconfig up
> thread start
> srp client autostart enable
```

Where <PROVIDE THE DATASET OF THE BR THAT YOU NEED TO JOIN i.e BR hosted in TH> is the dataset recorded in Setup Router Step 5.

16.4 Thread and IP-based Device Communication

Let's test out the communication between Thread device and IP-Based Device.
First, commission the IP-Based device to the fabric. Afterwards, on the chip-tool enter the following command to discover the IP-Based device over DNS-SD.

```
avahi-browse -rt _matter._tcp

//output
+ wlp3s0 IPv6 077D799425BEE2B7-0000000000000001        _matter._tcp      local
= wlp3s0 IPv6 077D799425BEE2B7-0000000000000001        _matter._tcp      local
   hostname = [00E04C09E10F.local]
   address = [2401:7400:6008:f59a:2e0:4cff:fe09:e10f]
   port = [5540]
   txt = ["T=1" "SAI=300" "SII=5000"]
```

To ensure that the thread end device is communicating to the IP-Based device through the OTBR, disable the WiFi and Ethernet interfaces. Replace the interface names with your own.

```
sudo ifconfig wlan0 down
sudo ifconfig eth0 down
```

In the RCP Shell console, discover the device IP address.

```
> dns service 077D799425BEE2B7-0000000000000001 _matter._tcp.default.service.arpa

//output
DNS service resolution response for 077D799425BEE2B7-0000000000000001 for service _matter._tcp.default.service.arpa.
Port:5540, Priority:0, Weight:0, TTL:10
Host:00E04C09E10F.default.service.arpa.
HostAddress:2401:7400:6008:f59a:2e0:4cff:fe09:e10f TTL:10
TXT:[SII=35303030, SAI=333030, T=31] TTL:10
```

Ping the IP address of the IP-Based device.

```
> ping 2401:7400:6008:f59a:2e0:4cff:fe09:e10f0f

//output
16 bytes from 2401:7400:6008:f59a:2e0:4cff:fe09:e10f: icmp_seq=1 hlim=254 time=66ms
1 packets transmitted, 1 packets received. Packet loss = 0.0%. Round-trip min/avg/max = 66/66.0/66 ms.
Done
```

 All information provided in this document is subject to legal disclaimers.