



# 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](http://www.realtek.com)

**COPYRIGHT**

©2018 Realtek Semiconductor Corp. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means without the written permission of Realtek Semiconductor Corp.

**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	Update on Device Types support for v1.3 Add Chapter 3. SDK Structure Update build instructions for v1.3 device type and Matter Restructure

# Table of Contents

COPYRIGHT .....	2
DISCLAIMER .....	2
TRADEMARKS.....	3
USING THIS DOCUMENT .....	3
Revision History .....	4
List of Figures .....	7
List of Tables .....	7
1 Summary .....	8
2 Introduction.....	8
2.1 About Matter .....	8
2.2 Realtek Ameba Matter Solution.....	9
2.2.1 Supported Matter Device Types .....	9
2.2.2 Supported Realtek ICs.....	10
2.2.3 Ameba Project Repository .....	10
2.2.4 Ameba Application Notes for Reference .....	10
3 Ameba Matter SDK Structure .....	11
3.1 AmebaZ2 / AmebaD SDK Structure.....	11
4 Build Environment .....	12
4.1 Environment requirement .....	12
4.1.1 Hardware .....	12
4.1.2 Software .....	12
4.2 Build Project .....	13
4.2.1 Building Code.....	13
4.2.2 Matter Sample Application.....	15
4.2.3 Porting Layer Matter Sample Application.....	15
4.3 Troubleshooting.....	16
4.4 Flashing Image .....	17
4.4.1 Image Tool .....	17
5 Matter Tools Generation.....	18
5.1 Chip Tool .....	18
5.2 Chip Cert and spake2p .....	18
5.3 QR Code Generator Tool .....	19
5.4 Zap Tool.....	20
6 Commissioning .....	21
6.1 Commissioning methods.....	21
6.1.1 BLE Commissioning.....	21
6.1.2 Wi-Fi (IP) Commissioning.....	21

6.1.3	Factory Reset Commands .....	21
6.1.4	Troubleshooting.....	22
7	Controlling commands .....	23
7.1	Cluster Control .....	23
7.1.1	OnOff Commands .....	23
7.1.2	ColorControl Commands .....	23
7.1.3	LevelControl Commands.....	23
7.1.4	Thermostat Commands .....	23
7.1.5	Binding and Controlling a Device .....	24
8	Matter OTA.....	25
8.1	Building Matter OTA Firmware .....	25
8.1.1	Building AmebaZ2 Matter OTA Firmware.....	26
8.1.2	Building AmebaD Matter OTA Firmware .....	27
8.1.3	Validate OTA Image .....	27
8.2	Matter Over-the-Air (OTA) Firmware Update.....	28
8.2.1	Prerequisites .....	28
8.2.2	Executing OTA.....	28
8.2.3	Troubleshooting.....	29
9	Matter Multi-Fabric.....	30
10	Matter Device Configuration Table (DCT) .....	32
10.1	DCT Modules Calculation .....	32
10.2	DCT Encryption.....	32
11	Matter Production.....	33
11.1	Matter Certificate and Device Onboarding Payload .....	33
11.1.1	Certificate and Keys .....	33
11.2	Certificate Generation.....	34
11.3	Matter Factory Data Binary.....	36
11.3.1	Prerequisite .....	36
11.3.2	Generate Factory Data Binary .....	36
11.3.3	Flash Factory Data Image.....	37
11.3.4	Factory Data Encryption .....	37
11.3.5	Generation of Firmware .....	37
11.3.6	Commissioning.....	37
11.4	Hardcoding Onboarding Payload into SDK.....	38
12	Matter Certification.....	39
12.1	Introduction to Matter Test Harness (TH).....	39
12.2	Introduction to PICS and PICS Tools .....	39
13	OpenThread Border Router (OTBR) Setup .....	40
13.1	Hardware Requirements .....	40

13.2	Setup OTBR .....	40
13.3	Setup Thread End Device .....	41
13.4	Thread and IP-based Device Communication .....	41

## List of Figures

Figure 1: Matter Data Model Architecture .....	8
Figure 2: Realtek Matter Architecture .....	9
Figure 3: Environment Build Failure.....	16
Figure 4: BLE scan failure .....	22
Figure 5: Unparallel network phenomenon.....	22
Figure 6: Unparallel network failure timeout .....	22
Figure 7: Matter OTA Process .....	29
Figure 8: DCT Module Declaration .....	32
Figure 9: DCT Address Declaration .....	32

## List of Tables

Table 1: Supporting Device Types.....	9
Table 2: AmebaZ2 Specifications .....	10
Table 3: AmebaD Specifications.....	10

# 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.

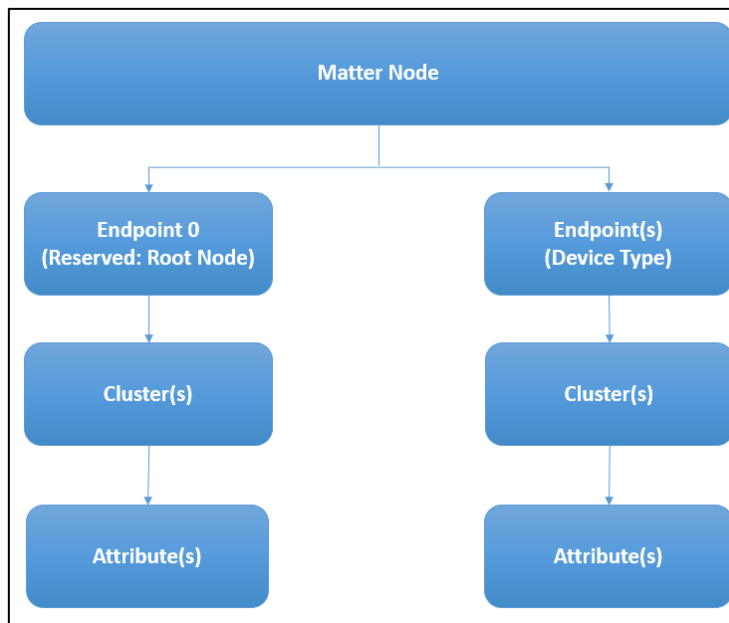


Figure 1: Matter Data Model Architecture

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.

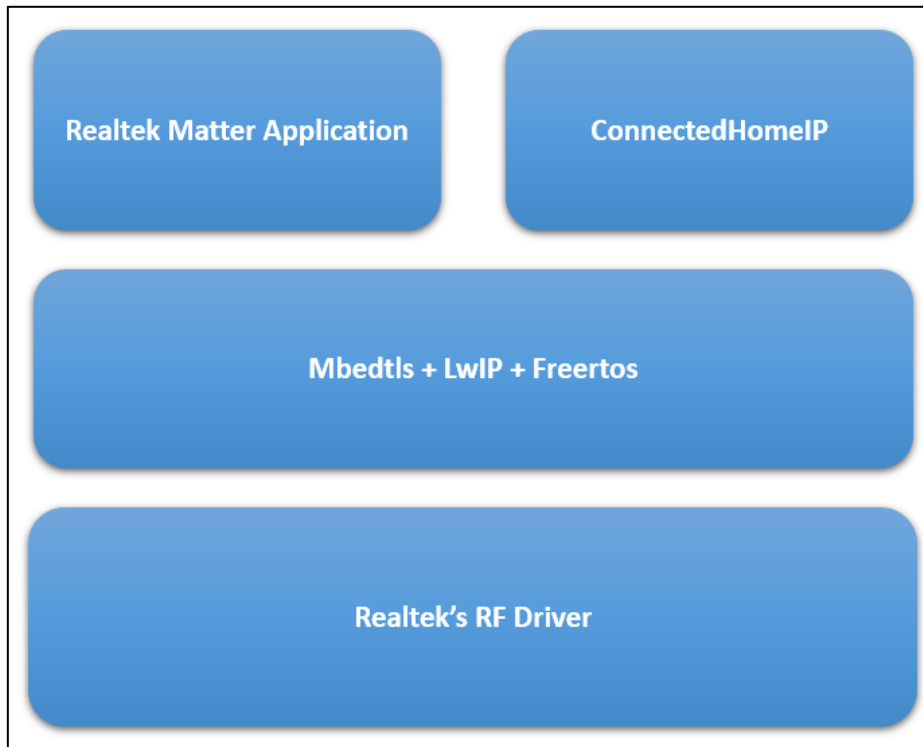


Figure 2: Realtek Matter Architecture

### 2.2.1 Supported Matter Device Types

This is a list of device types supported by Realtek:

1. Air Purifier	14. Flow Sensor	27. Pressure Sensor
2. Air Quality Sensor	15. Heating/Cooling Unit	28. RainSensor
3. Color Dimmer Switch	16. Humidity Sensor	29. Refrigerator
4. Color Temperature Light	17. Laundry Dryer	30. Robotic Vacuum Cleaner
5. Contact Sensor	18. Laundry Washer	31. Room Air Conditioner
6. Cook Surface	19. Light Sensor	32. Smoke/CO Alarm
7. Dimmable Light	20. Microwave Oven	33. Temperature Sensor
8. Dimmable Plug-In Unit	21. Occupancy Sensor	34. TemperatureControlledCabinet
9. Dimmer Switch	22. On/Off Light	35. Thermostat
10. Dishwasher	23. On/Off Light Switch	36. WaterFreezeDetector
11. Extended Color Light	24. On/Off Plug-In Unit	37. WaterHeater
12. ExtractorHood	25. On/Off Sensor	38. WaterLeakDetector
13. Fan	26. Oven	39. Window Covering

Table 1: Supporting Device Types

\* Table 1 is updated after Matter V1.3

\* 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.

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

**Table 2: AmebaZ2 Specifications**

### (2) 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.

<b>MCU</b>	32-bit KM4 (Arm Cortex-M33 compatible) 32-bit KM0 (Arm Cortex-M23 compatible)
<b>Memory</b>	512KB SRAM + 4MB PSRAM + 2MB Flash
<b>Key Features</b>	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 3: AmebaD Specifications**

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

## 2.2.3 Ameba Project Repository

AmebaZ2 SDK: [https://github.com/ambiot/ambz2\\_matter](https://github.com/ambiot/ambz2_matter)

AmebaD SDK: [https://github.com/ambiot/ambd\\_matter](https://github.com/ambiot/ambd_matter)

CHIP SDK: <https://github.com/project-chip/connectedhomeip>

## 2.2.4 Ameba Application Notes for Reference

AmebaZ2 Application Notes : [AN0500 Realtek Ameba-ZII application note.en](#)

AmebaD Application Notes : [AN0400 Ameba-D Application Note](#)

## 3 Ameba Matter SDK Structure

### 3.1 AmebaZ2 / AmebaD 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
      |_ api
      |_ project
        |_ amebaz2 or amebad
          |_ make
            |_ individual subfolders for different device types
      |_ common
        |_ atcmd
        |_ bluetooth
        |_ include
        |_ mbedtls
        |_ port
        |_ protobuf
      |_ core
      |_ driver
      |_ example
        |_ individual subfolders for different device types
  |_ 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.
- **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.
- **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.
- **driver:** Specific drivers enabling different Matter devices to function smoothly with the Ameba SDK, tailored to each device type's unique requirements.
- **example:** Code snippets and projects demonstrating Matter device usage in the Ameba SDK, organized into subfolders for easy navigation and learning across different device types.

## 4 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.

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](#).

### 4.1 Environment requirement

#### 4.1.1 Hardware

- A Realtek Ameba Board
- USB cable – Micro USB
- 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](#).

#### 4.1.2 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
```

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

## 4.2 Build Project

The following demonstration will be on all-clusters app, please find other applications-built information [here](#).

### 4.2.1 Building Code

#### 4.2.1.1 Building with AmebaZ2 (RTL8720C)

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

```
mkdir dev
cd dev
```

Step 2: Git clone repository

```
//Clone AmebaZ2 SDK
git clone https://github.com/ambiot/ambz2_matter.git

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

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

```
cd ambz2_matter/project/realtek_amebaz2_v0_example/GCC_RELEASE

//Make Matter Libraries for all-cluster apps
make all_clusters

//Make Ameba Project
make is_matter

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

The image file, "flash\_is.bin" is located at **project/realtek\_amebaz2\_v0\_example/GCC\_RELEASE/application\_is/Debug/bin**.

#### 4.2.1.2 Building with AmebaD (RTL8722D)

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

```
mkdir dev
cd dev
```

Step 2: Git clone repository

```
//Clone AmebaD SDK
git clone https://github.com/ambiot/ambd\_matter.git

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

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

```
//Make project_lp
cd ambd_matter/project/realtek_amebaD_va0_example/GCC_RELEASE/project_lp/

make all

//Change to project_hp
cd ambd_matter/project/realtek_amebaD_va0_example/GCC_RELEASE/project_hp/

//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**.

## 4.2.2 Matter Sample Application

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

### 4.2.2.1 AmebaZ2 Matter Sample App Build

Under the ***ambz2\_matter/project/realtek\_amebaz2\_v0\_example/GCC-RELEASE*** directory, use the following commands for build:

```
cd ambz2_matter/project/realtek_amebaz2_v0_example/GCC-RELEASE
make all_clusters / air_purifier / light / light_switch
make is_matter
```

### 4.2.2.2 AmebaD Matter Sample App Build

Under the ***amd\_matter/project/realtek\_amebaD\_va0\_example/GCC-RELEASE/project\_hp*** directory, use the following commands for build:

```
cd amd_matter/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp
make -C asdk all_clusters / air_purifier / light / light_switch
make all
```

## 4.2.3 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/example/*** of the SDK.

Within each example, there is a readme, which shows how the example is being implemented and how it can be build.

E.g., lighting-app, the readme can be found in: AmebaZ2 [link](#) and AmebaD [link](#).

Each Sample Application has a macro designated for it. The macro can be found in ***component/common/application/matter/common/config/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.,

- example/light : enable **CONFIG\_EXAMPLE\_MATTER\_LIGHT**
- example/thermostat : enable **CONFIG\_EXAMPLE\_MATTER\_THERMOSTAT**
- example/refrigerator : enable **CONFIG\_EXAMPLE\_MATTER\_REFRIGERATOR**

### 4.2.3.1 AmebaZ2 Porting Layer Matter Sample App Build

Under ***ambz2\_matter/component/common/application/matter/example/*** directory, use the following commands for build:

```
cd ambz2_matter/project/realtek_amebaz2_v0_example/GCC-RELEASE
make aircon_port / bridge_dm / dishwasher_port / fan_port / laundrywasher_port /
light_port / light_dm / microwaveoven_port / refrigerator_port / thermostat_port
make is_matter
```

### 4.2.3.2 AmebaD Porting Layer Matter Sample App Build

Under the `ambd_matter/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp` directory, use the following commands for build:

```
cd ambd_matter/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp

make -C asdk aircon_port / bridge_port / bridge_dm / dishwasher_port /
laundrywasher_port / light_port / light_dm / refrigerator_port / thermostat_port

make all
```

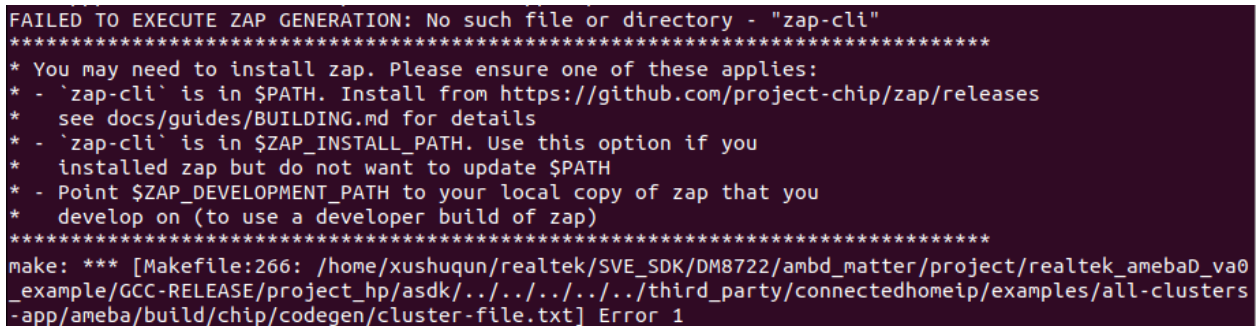
## 4.3 Troubleshooting

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

Here is a list of commonly encountered build errors:

### 4.3.1.1 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 3: Environment Build Failure

Simply activate the environment to fix the issue.

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

### 4.3.1.2 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
```



## 4.4 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.

### 4.4.1 Image Tool

#### 4.4.1.1 Linux Image Tool

The Linux Image Tool comes together in the SDK. Find it under the directory ***tools/AmebaX/Image\_Tool\_Linux***. 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]>
```

#### 4.4.1.2 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].

## 5 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.

### 5.1 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)
```

### 5.2 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
```

## 5.3 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>
productID <product_id>
commissioningFlow <commissioning_flow>
rendezVousInformation <discovery_mode>
discriminator <discriminator>
setUpPINCode <passcode>
```

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);
}
```

## 5.4 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_zapttools.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***.

## 6 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.

### 6.1 Commissioning methods

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

#### 6.1.1 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`

#### 6.1.2 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.

Method 1: Discover devices with setup code.

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

Method 2: Discover devices with discriminator and pair with setup code.

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

Method 3: Discover and pair devices with QR code.

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

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

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

#### 6.1.3 Factory Reset Commands

To reset the device into factory mode, enter the following commands:

For AmebaZ2: ATM\$ (v1.1.0.1 and above version) or ATS\$ (v1.1.0.1 and below version)

For AmebaD: ATM\$ (v1.2.0.0 and above version) or ATS#/ATM# (v1.2.0.0 and below version)

## 6.1.4 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:

### 6.1.4.1 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 4: 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.]

### 6.1.4.2 Commissioning fails due to unparallelled 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 5: Unparallel network phenomenon

And eventually the controller will timeout

```
[1687504182.935703][237792:237792] CHIP:DL: System Layer shutdown
[1687504186.126848][237792:237792] CHIP:T00: Run command failure: ../../examples/chip-tool/third_party/connectedhomeip/src/lib/address_resolve/AddressResolve_DefaultImpl.cpp:114: CHIP Error 0x00000032: Timeout
```

Figure 6: Unparallel network failure timeout

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

## 7 Controlling commands

### 7.1 Cluster Control

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

#### 7.1.1 OnOff Commands

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

#### 7.1.2 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>
```

#### 7.1.3 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>
```

#### 7.1.4 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>
```

### 7.1.5 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>
```



## 8 Matter OTA

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

### 8.1 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"
```

### 8.1.1 Building AmebaZ2 Matter OTA Firmware

Step 1: In *GCC-Release/amebaz2\_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, MO, 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 ambz2_matter/tools/matter/ota

python3 ota_image_tool.py create -v 0x8888 -p 0x9999 -vn 2 -vs 2.0 -da
sha256 ../../../../project/realtek_amebaz2_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.

### 8.1.2 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 ambd_matter/tools/matter/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.

### 8.1.3 Validate OTA Image

To check your OTA image, enter the following command:

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

## 8.2 Matter Over-the-Air (OTA) Firmware Update

### 8.2.1 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>
```

### 8.2.2 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
```

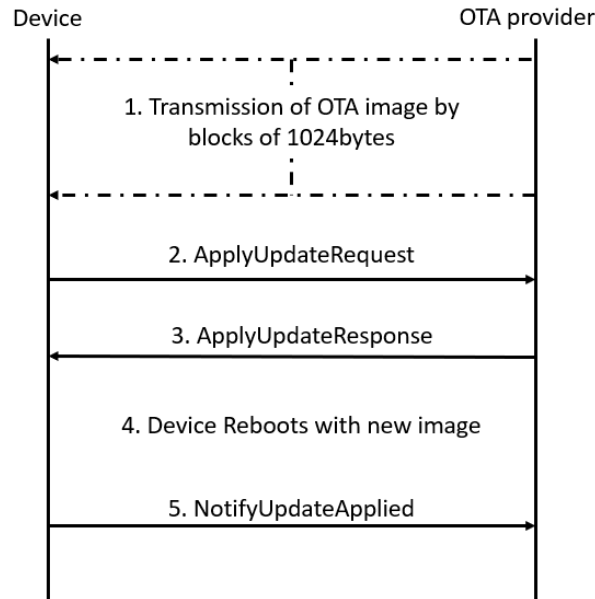
Step 4: Configure the ACL of the ota-provider-app to allow access for device.

```
./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 7: 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.

## 8.2.3 Troubleshooting

### 8.2.3.1 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

```

### 8.2.3.2 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.

## 9 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 pairing 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, ...].

## 10 Matter Device Configuration Table (DCT)

The DCT is a permanent storage that stores data on the flash region.

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)

### 10.1 DCT Modules Calculation

In *component/common/application/matter/common/port/matter\_dcts.c*, the Module Number are defined as follows:

```
#define DCT_BEGIN_ADDR_MATTER DCT_BEGIN_ADDR
#define MODULE_NUM 13
#define VARIABLE_NAME_SIZE 32
#define VARIABLE_VALUE_SIZE 64 + 4

#define DCT_BEGIN_ADDR_MATTER2 DCT_BEGIN_ADDR2
#define MODULE_NUM2 6
#define VARIABLE_NAME_SIZE2 32
#define VARIABLE_VALUE_SIZE2 400 + 4
```

Figure 8: DCT Module Declaration

There are 2 DCT regions, DCT1 and DCT2:

- DCT1 region (MODULE\_NUM) has 13 modules for data below 64 bytes (< 64 bytes).
- DCT2 regions (MODULE\_NUM2) has 6 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)

#### How many variables can fit inside one module?

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

#### How to verify Flash space is sufficient for the configuration?

The declaration of DCT\_BEGIN\_ADDR and DCT\_BEGIN\_ADDR2 is in

*component/common/application/matter/common/config/platform\_opts\_matter.h*.

Here is an example of how DCT is configured for Matter, users can adjust according to their needs.

```
#define DCT_BEGIN_ADDR (0x400000 - 0x13000) // 0x3ED000 ~ 0x3FB000 : 56K
#define DCT_BEGIN_ADDR2 (0x400000 - 0x1A000) // 0x3E6000 ~ 0x3ED000 : 24K
#define MATTER_FACTORY_DATA (0x3FF000) // last 4KB of external flash - write protection is supported in this region
```

Figure 9: DCT Address Declaration

- Total space available for DCT1 = 0x13000 - 0x5000 = 56KB = 57334
  - 13 DCT1 modules take up 13 \* 4096 = 53248 Bytes (Fits)
- Total space available for DCT2 = 0x1A000 - 0x13000 = 24KB = 28672
  - 6 DCT2 modules take up 6 \* 4096 = 24576 Bytes (Fits)

### 10.2 DCT Encryption

To enable DCT Encryption, find **CONFIG\_ENABLE\_DCT\_ENCRYPTION** in application.is.matter.mk (AmebaZ2) or Makefile.include.gen (AmebaD) when building the firmware.



# 11 Matter Production

## 11.1 Matter Certificate and Device Onboarding Payload

### 11.1.1 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.

#### 11.1.1.1 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 a 12-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.

## 11.2 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`.**

## 11.3 Matter Factory Data Binary

### 11.3.1 Prerequisite

Generate chip-cert and spake2p as mentioned [here](#) and install python dependency with this command:

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

### 11.3.2 Generate Factory Data Binary

Run the ameba\_factory.py python script, passing in necessary arguments:

```
cd ambz2_matter/tools/matter

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\_ENCRYPTION** is enabled, learn more about Factory Data Encryption [here](#).

### 11.3.3 Flash Factory Data Image

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

The default address to flash ameba\_factory.bin is **0x083FF000**, customer may configure it using **MATTER\_FACTORY\_DATA** macro in **component/common/application/matter/common/config/platform\_opts\_matter.h**. Make sure to check for partition conflicts. An example to flash using Image\_Tool\_Linux is shown below:

```
cd ambz2_matter/tools/AmebaZ2/Image_Tool_Linux
./flash.sh /dev/ttyUSB0 ./ameba_factory.bin 0x083FF000
```

### 11.3.4 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\_ENCRYPTION** in application.is.matter.mk (AmebaZ2) or Makefile.include.gen (AmebaD) 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).

### 11.3.5 Generation of Firmware

Make sure your firmware is built with **CONFIG\_ENABLE\_AMEBA\_FACTORY\_DATA** enabled in the **matter\_common\_flags.mk** (for AmebaZ2) or **Makefile.include.matter.hp.gen** (AmebaD). Furthermore, you need to check if the VID and PID matches the ones in factory data binary.

### 11.3.6 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>
```

## 11.4 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

## 12 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.

### 12.1 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.

### 12.2 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.

## 13 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](#).

### 13.1 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.

### 13.2 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.



### 13.3 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.

### 13.4 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
```