RA6W2 Matter Certificate

RA6W2 is a highly integrated ultra-low-power Wi-Fi + Bluetooth Low Energy (Bluetooth® LE) combo system in a package that allows you to develop Wi-Fi and Bluetooth LE solutions using a single chip. The purpose of this document is to provide detailed guidelines on how to do Matter 1.4 certification.

Contents

Figures

Tables

1. Terms and Definitions

AP	Access Point
ATL	Authorized Test Laboratory
CSA	Connectivity Standard Alliances
DAC	Device Attestation Certificate
DCL	Distributed Compliance Ledger
DUT	Device under Test
PAA	Product Attestation Authority
PAI	Product Attestation Intermediate
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation Extra Information for Testing
SKID	Subject Key Identifier
STA	Station
TH	Test Harness
VID	Vendor ID
ZAP	ZCL Advanced Platform

2. References

[1] RA6W2 Datasheet, Renesas Electronics.
[2] RA6W2 Getting Started Guide, Manual, Renesas Electronics.
[3] RA6W2 Getting Started with Matter, Manual, Renesas Electronics.
[4] RA6W1 OTA Update, Application Note, Renesas Electronics.
[5] Matter TH User Guide.
[6] Matter V1_4_Test Plan Step Verification.

Note 1 References are for the latest published version, unless otherwise indicated.

3. Introduction

Matter emerged as the universal application layer for connected devices, enabling seamless interoperability across smart home platforms through standardized communication protocols, robust security, and a unified certification framework. Version 1.4 of the specification builds on this foundation with expanded device type support, advanced security enhancements, and improved multi-fabric synchronization features.
Achieving Matter certification is a critical step in ensuring that a product not only meets CSA technical requirements but also delivers a consistent and reliable user experience across all Matter-enabled ecosystems. The certification process validates functional behaviour, security compliance, and interoperability through a series of rigorous, standardized test cases executed with CSA-approved tools and procedures.
This document serves as a comprehensive guide to achieving Matter 1.4 certification for products based on the RA6W2. This is intended to guide engineers and test teams through that process in a structured, practical manner — bridging the gap between the official Matter 1.4 specification and real-world implementation. It emphasizes both technical accuracy and process efficiency, ensuring a smooth path from product development to certification approval.

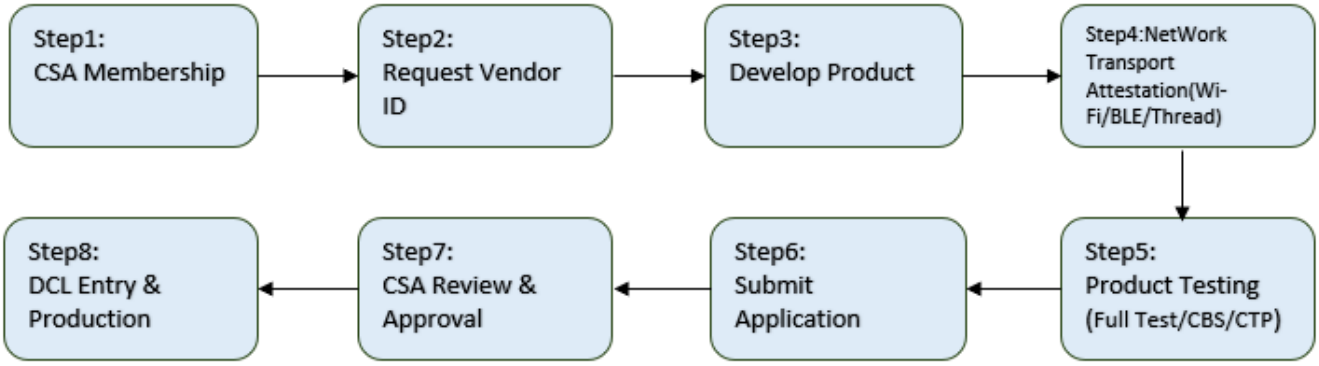

Figure 1. Matter certification plan

Figure 1 shows the various processes for complete matter certification:

- Become a member of the CSA – join the Connectivity Standards Alliance to gain access to specifications, certification tools, and resources.
- Request a Vendor ID (VID) – obtain a unique identifier for your company from CSA Certification.
- Select applicable network transports – decide which connectivity options (Wi-Fi, Thread, Ethernet) your product uses.
- Certification from relevant standards organization – Complete any non-Matter certifications required for your chosen network transports (Wi-Fi Alliance, Thread Group).
- Matter product testing – test your product against the Matter specification using CSA-approved tools (Test Harness, PICS, PIXIT, and so on).
- Submit certification application – use the CSA Certification Web Tool to submit your application with all required documentation.
- Application review by CSA Certification – CSA reviews your test results, documentation, and compliance declarations.
- Approval – after passing the review, CSA issues a Certificate of Compliance, lists your product on the CSA website, grants use of certified logos, and finalizes the Certification Declaration (CD).
- Distributed Compliance Ledger (DCL) update – your product's certification record is added to the CSA's blockchain-based compliance ledger for transparency and verification.

4. Build PICS Test Plan

PICS is a formal document (in CSV or XML format) that defines exactly which Matter features, clusters, attributes, and commands your device implements. Building a PICS test plan for Matter 1.4 certification involves several key steps. The test plan acts as a blueprint to ensure your product meets all the technical and functional requirements defined by the Connectivity Standards Alliance (CSA). This process is crucial for successful certification.
To build the PICS test plan:

1. Download the PICS template: download the official PICS template from the CSA website. This is typically a zipped folder containing all the cluster XMLs. This is your primary tool for creating a test plan.
2. Identify your product's capabilities: review your product's implementation and identify every endpoint, cluster, attribute, and command it supports. You need to understand what is mandatory and what is optional. For instance, if you are certifying a door lock, you need to know:
   - How many endpoints does it have (one for the door lock, one for the root node).
   - Which clusters are implemented at each endpoint (door lock cluster, General Commissioning cluster).
   - Which specific attributes and commands within those clusters are supported.
3. Fill out the PICS document: with your product's capabilities identified, you can now populate the PICS template. This involves:
   - Marking clusters: for each endpoint, you mark the clusters that are implemented as either a client or a server. A server is the device that provides the functionality (the light bulb itself), while a client is the device that controls it (for example, a switch).
   - Specifying attributes: within each cluster, you can indicate which attributes are supported. You also need to specify if they are readable, writable, or both.
   - Indicating commands: you do the same for commands, specifying which ones your device can send or receive.
4. Validate the PICS Document: before submitting, it is crucial to validate your PICS document for accuracy and completeness. The CSA provides tools to help with this. The PICS file is used to generate the test cases, so any errors here could lead to failures during the certification testing. Ensure that the PICS file accurately reflects the device's behavior. For instance, if your PICS says your lock supports the Wi-Fi commissioning cluster, the test automation expects to be able to check related Wi-Fi attributes of the device. For example, if you are devising the PICS for a Door Lock device type, to get a finalized PICS plan:
   a. Download the official PICS/XML file from the CSA site.
   b. Identify the device’s endpoints and clusters; a door lock has at least two logical endpoints, and your PICS has separate sections for each of these endpoints:
      - Endpoint 0: this is a special, mandatory endpoint for all Matter devices. This is the mandatory "root node" for all Matter devices. It houses the foundational clusters that every device must support.
      - Endpoint 1 (or higher): this is where your device's core functionality resides. For a Door Lock, this endpoint contains the Door Lock Cluster.
        Populate the PICS Plan for Foundational Clusters (Endpoint 0).
   c. Populate the PICS Plan for Foundational Clusters (Endpoint 0).
   d. Populate the PICS Plan for Your Door Lock Cluster (Endpoint 1).

To understand or devise the clusters, attributes, features for each of the endpoints you need see the two following documents which are provided by the CSA):

- The Matter Device Library Specification: this document defines all the standard device types, such as "On/Off Light","Door Lock,", "Thermostat," and so on.
  To find out which clusters are required for your door lock, see the Door Lock device type in the Matter Device Library Specification. Within this document, see the Section 8.1 for "Door Lock." This section explicitly lists the clusters that are mandatory for the device type. It tells you that the Door Lock device must implement the Door Lock Cluster (as a server). It also specifies that other clusters, like the Basic Cluster and Identify Cluster, are mandatory and must be on Endpoint 0. This is where you confirm which clusters need to be included in your PICS plan.
- The Matter Application Cluster Specification: this document provides the detailed breakdown of every cluster, including its attributes, commands, and events. To navigate these documents to get the information you need for your PICS plan for a door lock device type.
  When you know which clusters your device must implement, you need to find the specific requirements for each of

those clusters. This is where the Matter Application Cluster Specification comes in. In this document, navigate to the section for each cluster that you identified in the Device Library Specification.

- For your door lock, see Section 5.2 Add Door Lock Cluster in ZAP Tool. Within this section, you can find a detailed breakdown of the cluster's attributes, commands, and events. Each of these is tagged as either Mandatory (M), Optional (O), or Conditional (C).
  - Mandatory (M): you must implement this feature. If your device does not support it, it fails certification.
  - Optional (O): you may choose to implement this feature. If you do, you must declare it in your PICS plan, and it is tested. If you do not, it is not tested.
  - Conditional (C): this feature is mandatory only if a specific condition is met. For example, a "user management" command might be mandatory if your lock supports multiple user codes.

Some specific examples to show the Door Lock cluster attributes are (Figure 2):

- LockState (ID: 0x0000): mandatory (M). this attribute reports the current state of the lock (locked, unlocked). Your door lock must implement this.
- LockType (ID: 0x0001): mandatory (M). this attribute defines the physical type of lock (deadbolt, handle lock). Your door lock must implement this.
- DoorState (ID: 0x0003): optional (O). this attribute reports whether the door is open or closed, typically requiring a physical sensor. If your device has this sensor and functionality, you mark it as supported in your PICS. If not, you do not.
- NumberOfTotalUsers (ID: 0x0011): conditional (C). this attribute is mandatory if your door lock supports user management features (allowing multiple PIN codes to be set remotely). If your lock does not support user management, this attribute is not required.

130.2.2. Attributes

Variable	Description	Mandatory/Optional
DRLK.S.A0000(LockState)	Does the device implement LockState attribute ?	DRLK.S:M
DRLK.S.A0001(LockType)	Does the device implement LockType attribute ?	DRLK.S:M
DRLK.S.A0002(ActuatorEnabled)	Does the device implement ActuatorEnabled attribute ?	DRLK.S:M
DRLK.S.A0003(DoorState)	Does the device implement DoorState attribute ?	DPS
DRLK.S.A0004(DoorOpenEvents)	Does the device implement DoorOpenEvents attribute ?	[DPS]
DRLK.S.A0005(DoorClosedEvents)	Does the device implement DoorClosedEvents attribute ?	[DPS]
DRLK.S.A0006(OpenPeriod)	Does the device implement OpenPeriod attribute ?	[DPS]
DRLK.S.A0011(NumberOfTotalUsersSupported)	Does the device implement NumberOfTotalUsersSupported attribute ?	USR
DRLK.S.A0012(NumberOfPINUsersSupported)	Does the device implement	PIN

Figure 2. Application cluster specification – Attributes

DRLK.S.A0000	Does the device implement LockState attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S : M	T/F ✓
DRLK.S.A0001	Does the device implement LockType attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S : M	T/F ✓
DRLK.S.A0002	Does the device implement ActuatorEnabled attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S : M	T/F ✓
DRLK.S.A0003	Does the device implement DoorState attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F05 : M	T/F ✓
DRLK.S.A0004	Does the device implement DoorOpenEvents attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F05 : O	T/F
DRLK.S.A0005	Does the device implement DoorClosedEvents attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F05 : O	T/F
DRLK.S.A0006	Does the device implement OpenPeriod attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F05 : O	T/F
DRLK.S.A0011	Does the device implement NumberOfTotalUsersSupported attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F08 : M	T/F ✓
DRLK.S.A0012	Does the device implement NumberOfPINUsersSupported attribute ? §130.2.2. Attributes - allclusters.html[pdf]	DRLK.S AND DRLK.S.F00 : M	T/F ✓
DRLK.S.A0013	Does the device implement NumberOfRFIDUsersSupported attribute ?	DRLK.S AND DRLK.S.F01 :	T/F

Figure 3. Attributes enabled for Door Lock cluster in PICS

Figure 4 and Figure 5 show the specific commands:

- LockDoor : mandatory (M). This command allows a commissioner to remotely lock the door. Your door lock must respond to this command.
- UnlockDoor : mandatory (M). This command allows a commissioner to remotely unlock the door. Your door lock must respond to this command.
- SetUser : conditional (C). This command is mandatory if your door lock supports advanced user management, allowing new users or PINs to be configured. If your lock only supports basic lock/unlock, this command is not required.

130.2.5. Commands received

Variable	Description	Mandatory/Optional	Notes/Additional Constraints
DRLK.S.C00.Rsp(LockDoor)	Does the device implement receiving of LockDoor command?	DRLK.S:M	
DRLK.S.C01.Rsp(UnlockDoor)	Does the device implement receiving of UnlockDoor command?	DRLK.S:M	

Figure 4. Application Cluster Specification – Commands

PICS ITEMS – SERVER COMMANDS RECEIVED			
DRLK.S.C00.Rsp	Does the device implement receiving of LockDoor command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S : M	T/F ☑
DRLK.S.C01.Rsp	Does the device implement receiving of UnlockDoor command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S : M	T/F ☑
DRLK.S.C03.Rsp	Does the device implement receiving of UnlockWithTimeout command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S : O	T/F ☐
DRLK.S.C0b.Rsp	Does the device implement receiving of SetWeekDaySchedule command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S AND DRLK.S.F04 : M	T/F ☑
DRLK.S.C0c.Rsp	Does the device implement receiving of GetWeekDaySchedule command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S AND DRLK.S.F04 : M	T/F ☑
DRLK.S.C0d.Rsp	Does the device implement receiving of ClearWeekDaySchedule command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S AND DRLK.S.F04 : M	T/F ☑
DRLK.S.C0e.Rsp	Does the device implement receiving of SetYearDaySchedule command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S AND DRLK.S.F0a : M	T/F ☑
DRLK.S.C0f.Rsp	Does the device implement receiving of GetYearDaySchedule command? §130.2.5. Commands received - allclusters.html[pdf]	DRLK.S AND DRLK.S.F0a : M	T/F ☑

Figure 5. Commands enabled for Door Lock cluster in PICS

Figure 6 and Figure 7 show the following events:

- Lock Operation (ID: 0x0002): mandatory (M). This event is a mandatory event that indicates when a lock/unlock operation is performed. So, if your DUT claims Door Lock server support, it must implement this event.
- DoorLockAlarm (ID: 0x0000): mandatory (M). This event signals security alarms (for example, tamper detection, forced door). It means every device that implements Door Lock server must implement the DoorLockAlarm event.

130.2.4. Events

Variable	Description	Mandatory/Optional	Notes/Additional Constraints
DRLK.S.E00(DoorLockAlarm)	Does the device implement DoorLockAlarm event?	DRLK.S:M	
DRLK.S.E01(DoorStateChange)	Does the device implement DoorStateChange event?	DPS	
DRLK.S.E02(LockOperation)	Does the device implement LockOperation event?	DRLK.S:M	
DRLK.S.E03(LockOperationError)	Does the device implement LockOperationError event?	DRLK.S:M	
DRLK.S.E04(LockUserChange)	Does the device implement LockUserChange event?	USR	

Figure 6. Application Cluster Specification – Events

PICS ITEMS – SERVER EVENTS			
DRLK.S.E00	Does the device implement DoorLockAlarm event? §130.2.4. Events - allclusters.html[pdf]	DRLK.S : **M**	T/F ✓
DRLK.S.E01	Does the device implement DoorStateChange event? §130.2.4. Events - allclusters.html[pdf]	DRLK.S AND DRLK.S.F05 : **M**	T/F ✓
DRLK.S.E02	Does the device implement LockOperation event? §130.2.4. Events - allclusters.html[pdf]	DRLK.S : **M**	T/F ✓
DRLK.S.E03	Does the device implement LockOperationError event? §130.2.4. Events - allclusters.html[pdf]	DRLK.S : **M**	T/F ✓
DRLK.S.E04	Does the device implement LockUserChange event? §130.2.4. Events - allclusters.html[pdf]	DRLK.S AND DRLK.S.F08 : **M**	T/F ✓

Figure 7. Events enabled for Door Lock cluster in PICS

When you identify and populate all the supported endpoints, their clusters, attributes, features and events, mark them in the relevant pics template, and follow the same for all supported clusters.

Item Number	Feature Description	Status	Support
PICS ITEMS – USAGE			
DRLK.S	Does the device implement the Door Lock Cluster as a server? §130.1. Role - allclusters.html[pdf]		T/F ✓
DRLK.C	Does the device implement the Door Lock Cluster as a client? §130.1. Role - allclusters.html[pdf]		T/F ☐
PIXIT ITEMS – PIXIT			
PIXIT.DRLK.WrongCodeEntryLimit	The number of incorrect Pin codes or RFID presentment attempts a user is allowed to enter before the lock will enter a lockout state. §131. PIXIT Definition - allclusters.html[pdf]	DRLK.S AND (DRLK.S.F00 OR DRLK.S.F01) : **M**	CLICK TO AUTOFILL N/A True False Yes No 0x00 0x01 0xff
PIXIT.DRLK.UserCodeTemporaryDisableTime	The number of seconds that the lock shuts down following wrong code entry. §131. PIXIT Definition - allclusters.html[pdf]	DRLK.S AND (DRLK.S.F00 OR DRLK.S.F01) : **M**	CLICK TO AUTOFILL N/A True False Yes No 0x00 0x01 0xff

Figure 8. PIXIT items of Door Lock cluster in PICS

In the Figure 9, the items labeled PIXIT stand for Protocol Implementation Extra Information for Testing. PIXIT items are used to capture extra device-specific information that is not in the PICS but is needed to run certain test cases in the Test Harness.

For the Door Lock cluster, provide test-specific configuration values, so that the Test Harness can correctly interact with DUT. In the example:

- PIXIT.DRLK.WrongCodeEntryLimit (0XFF): the number of incorrect PIN/RFID attempts before lockout.
- PIXIT.DRLK.UserCodeTemporaryDisableTime (0XFF): the duration (in seconds) the lock remains disabled after wrong code attempts.

For each PIXIT item, determine the value from your firmware/device behaviour and give the value according to the device firmware. Also, the PIXIT must have values in the right format expected by the TH.

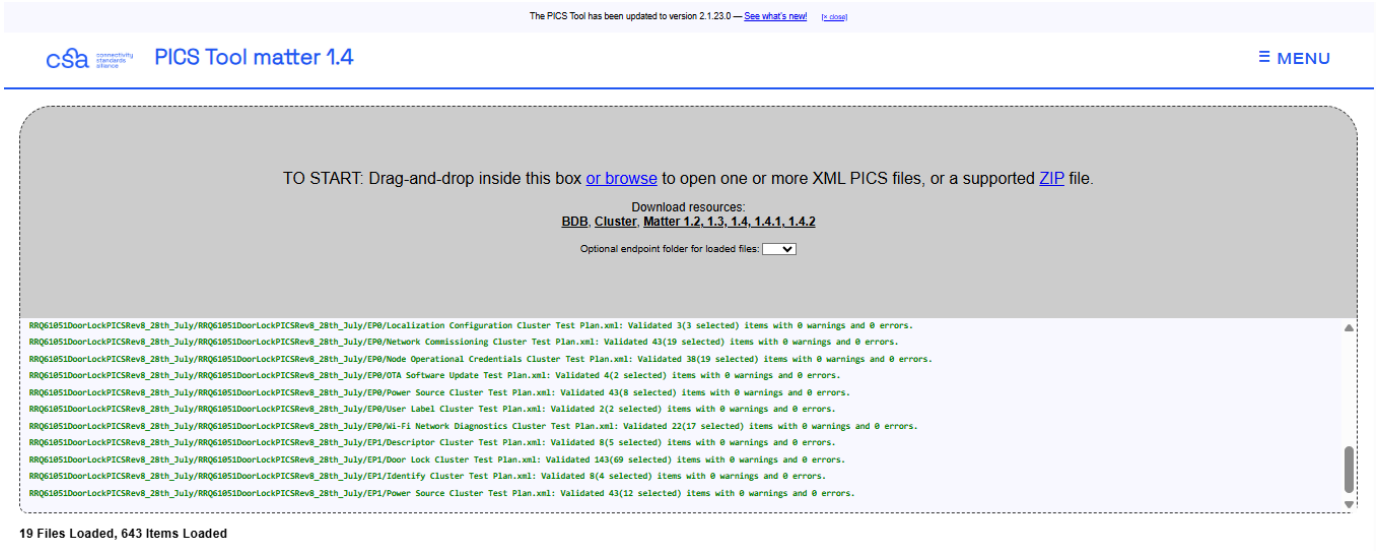

Figure 9. PICS tool

When all the relevant attributes, features, events are marked on the PICS tool, click Validate Test Cases as shown in Figure 10 to generate the list of test cases based on the clusters marked. All the testcases marked as Required are mandatory and are tested in the Certification lab. When you check the testcases and clusters, save the PICS file as a zip file and you need to send this along to the test lab and you need to input this PICS file while running test internally in the Test Harness tool.

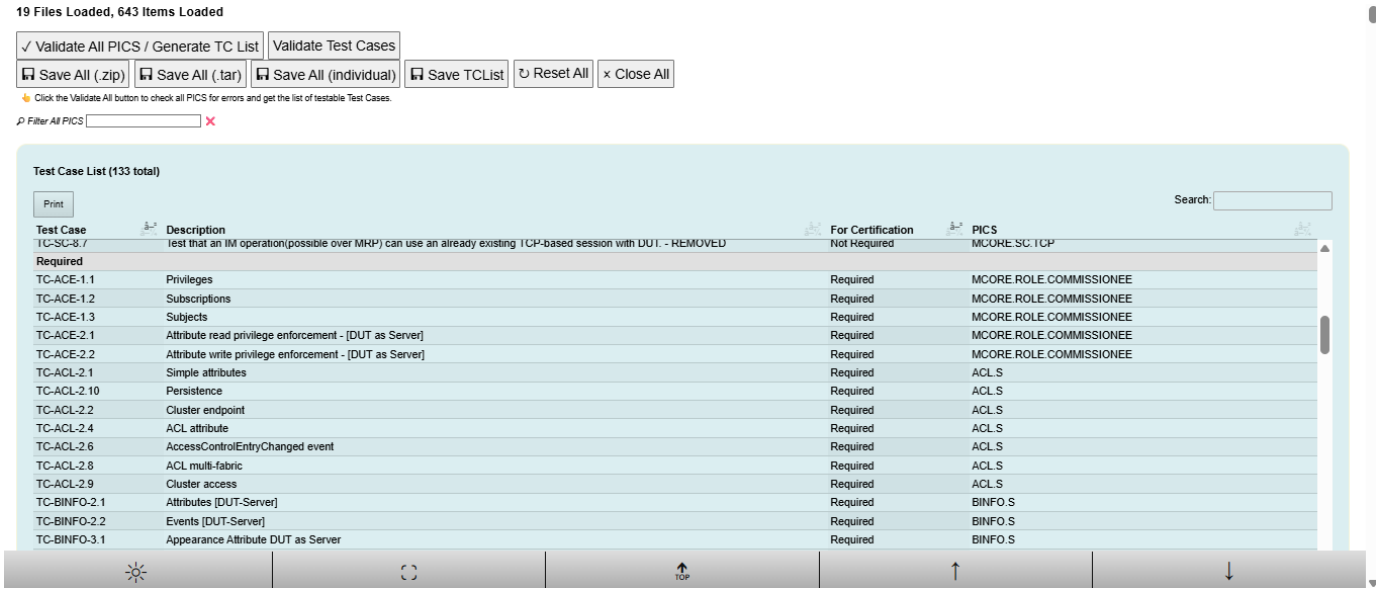

Figure 10. PICS tool test case generation

The PICS document is a foundational element of the Matter certification process. When you submit your product for certification, you provide this document to the Test lab. The lab's automation software then parses the PICS file to generate a customized test suite. This suite only runs the tests that are relevant to the features you've declared your product supports. This ensures that the testing is efficient and targeted, focusing on the specific functionality your product offers. By meticulously preparing your PICS test plan, you significantly increase the chances of a smooth and successful certification journey.

5. ZAP Tool

The ZCL Advanced Platform (ZAP) tool is an indispensable resource for Matter developers generally used on a Linux host. While the PICS plan declares what your device supports, ZAP is where you configure the device's data model, including its clusters, attributes, commands, and events, and then generate the corresponding source code for your application. Properly implementing attributes in ZAP is crucial for ensuring your device behaves as expected and passes certification tests.
ZAP is a graphical user interface tool that provides a visual way to define the capabilities of your Matter device. It abstracts away the complexities of the Matter data model, allows you to easily select and configure the various elements your device exposes. The output of the ZAP tool is a set of generated source code files (typically C++), which are then integrated into your device's firmware.

5.1 Access and Navigate ZAP Tool

To install ZAP tool, (connectedhomeip GitHub repository):

1. Clone the connectedhomeip repository: git clone https://github.com/projectchip/connectedhomeip.git.
2. Perform submodule update: git submodule update --init --recursive
3. Checkout matter 1.4 branch : git checkout tags/v1.4.0.0
4. Perform submodule update: git submodule update --init –recursive
5. Run bootstrap.h: source ./scripts/bootstrap.sh
6. Run activate.sh: source ./scripts/activate.sh

For more information on the ZAP tool, see Figure 11 and code generation, see the connectedhomeip zap tool document.
You can also use the following options:

- To launch the ZAP tool, find the example app for your device type, and edit it according to your device and save the zap.
- To edit the door lock cluster and add attributes, you can use the lock-app example zap from examples/lock-app/lock-common/lock-app.zap.
- To launch the ZAP tool, use the following command : ./scripts/tools/zap/run_zaptool.sh <.zap>
- To generate the final zap fil, specify the following configurations:
  - Endpoints: a list of endpoints (Endpoint 0, Endpoint 1) to select for your application.
  - Clusters: below each endpoint, you find a list of clusters that can be enabled for that endpoint.
  - Attributes, Commands, Events: within each cluster's configuration, you find tabs or sections for its attributes, commands, and events.

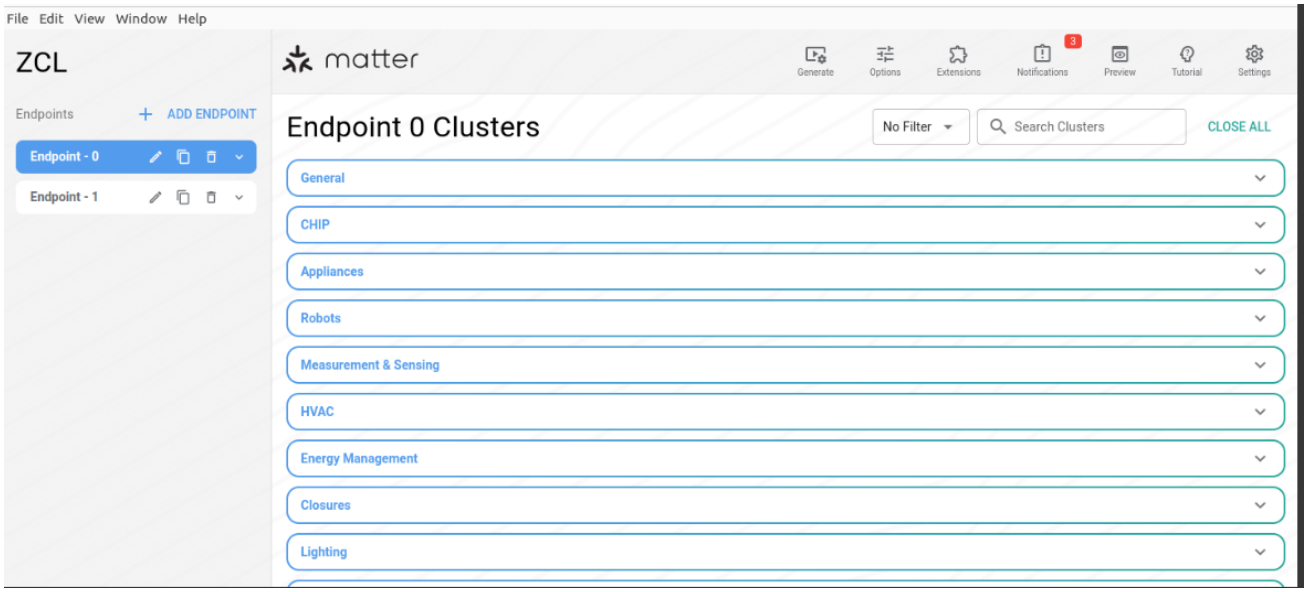

Figure 11. ZAP tool

5.1.1 Select Endpoints and Clusters

Matter devices have at least one endpoint, Endpoint 0, for foundational clusters. Your door lock, acting as a server, has at least one additional functional endpoint (Endpoint 1) dedicated to its door lock capabilities. You need to select both endpoints for the Door Lock cluster:

- Endpoint 0: in the ZAP tool, go to Endpoint 0. Check that the mandatory clusters Basic, Identify, Access Control, and so on, are enabled as Server clusters.
- Functional Endpoint (Endpoint 1):
  - Select your functional endpoint.
  - Enable the Door Lock Cluster as a Server cluster.
    If your device has other functionalities (a contact sensor), you can also enable those corresponding clusters on this or another appropriate endpoint, see Figure 12, and Figure 13, Figure 14.

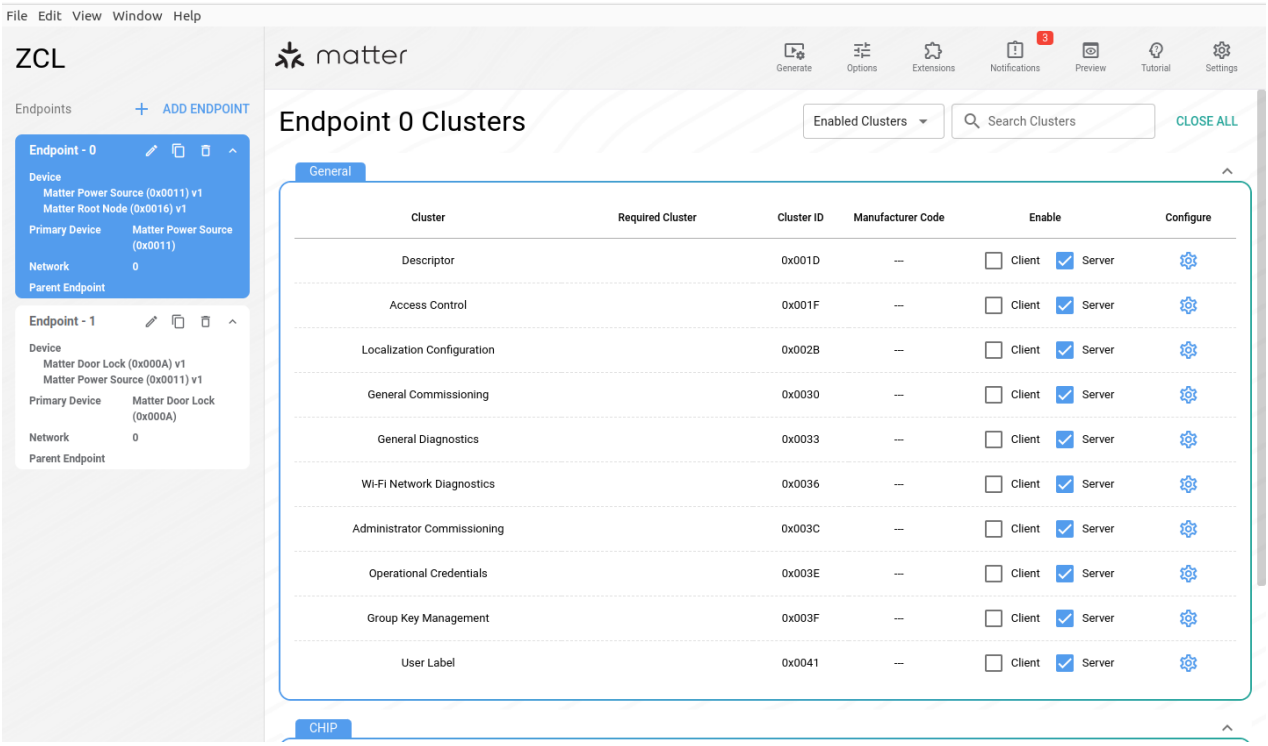

Figure 12. Endpoint 0 and Clusters

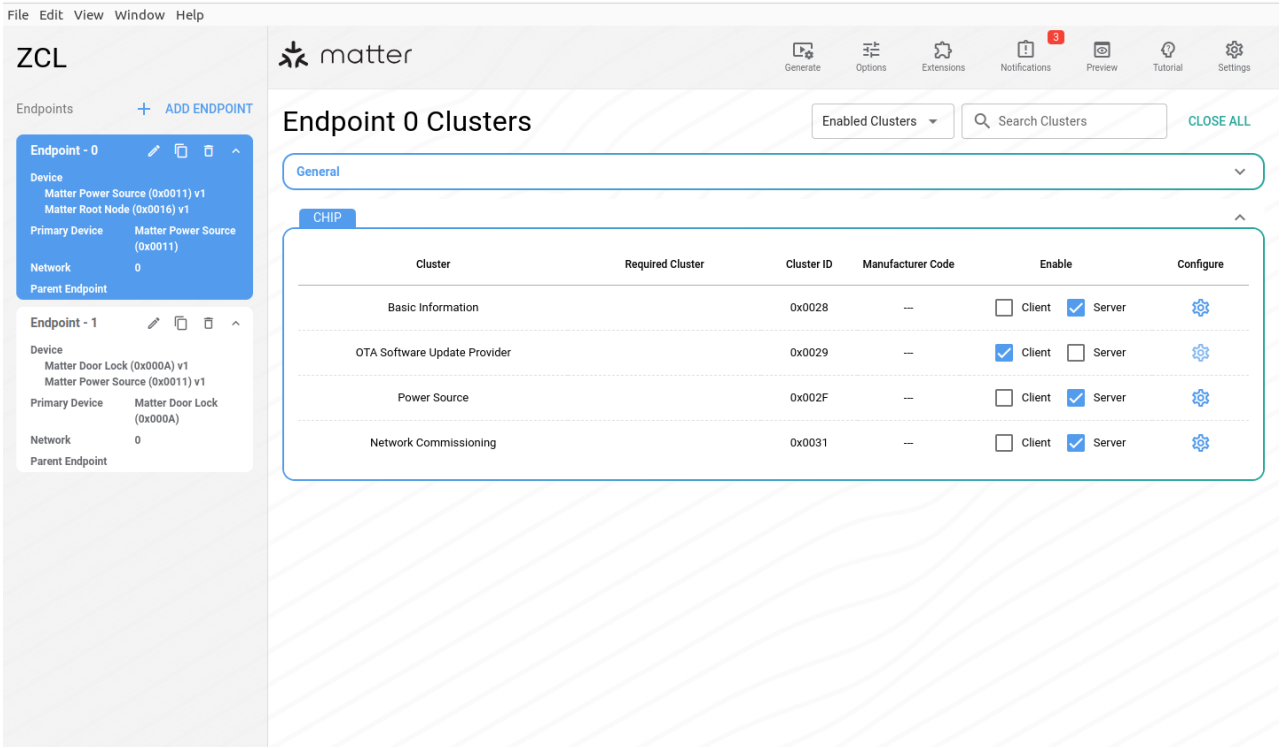

Figure 13. Endpoint 0 Clusters

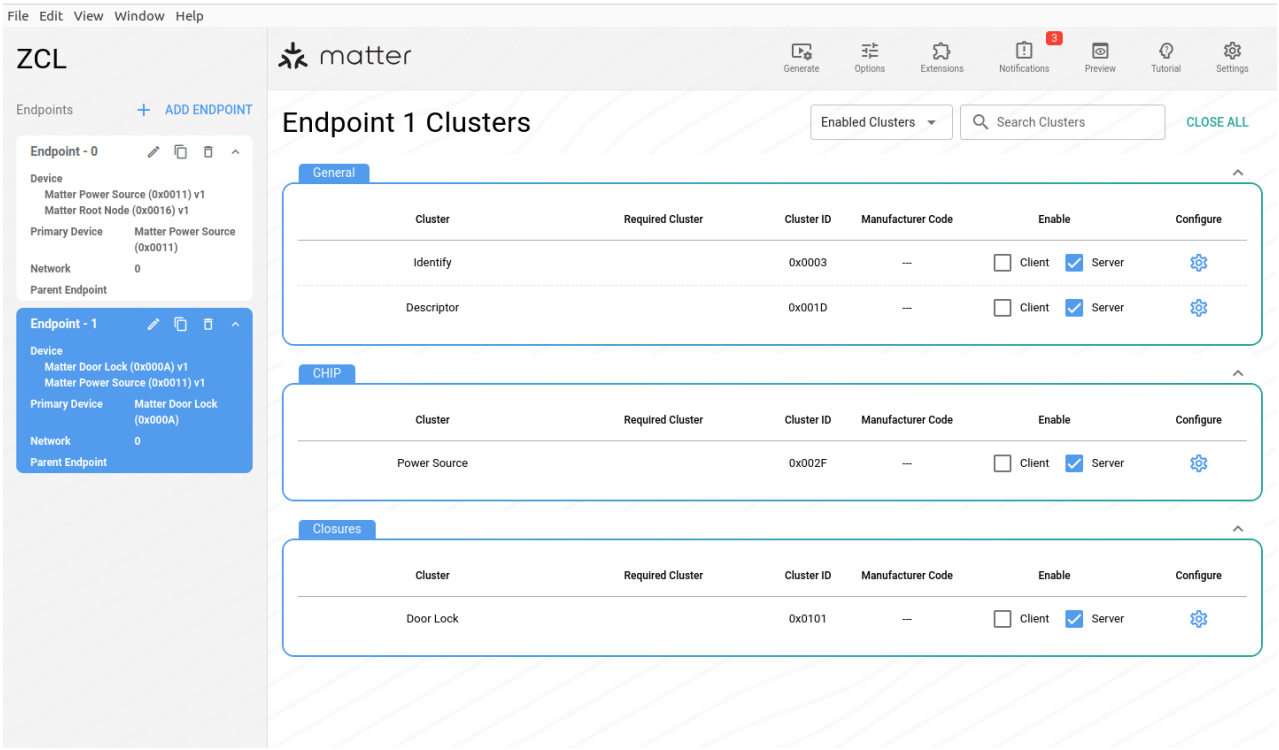

Figure 14. Endpoint 1 Clusters

When you select an endpoint and a cluster (the Door Lock cluster on Endpoint 1), configure its attributes by the following steps:

1. Go to the **Attributes** tab, which lists all the attributes defined for that cluster in the Matter specification, see Figure 15.
2. Enable Mandatory Attributes: for the Door Lock cluster, you must enable attributes **LockState** and **LockType**. ZAP often visually indicates mandatory attributes, sometimes preventing you from disabling them. Ensure these are marked as enabled.
3. Enable Optional Attributes (If implemented), to review the list of optional attributes. If your door lock device implements a feature corresponding to an optional attribute (**DoorState**). If it has a physical door sensor, or NumberOfTotalUsers or If it supports user management), enable it.
4. For each enabled attribute, configure the following properties:
   - RAM: Data is stored in volatile memory and is lost on power cycle.
   - NVM (Non-Volatile Memory): data persists across power cycles (flash memory). This is essential for attributes that must retain their state, as LockState.
   - External/Application-Managed: the attribute's value is managed directly by your application code, and ZAP generates accessors for you to read/write it.
   - Default Value: for some attributes, you can set a default value that the attribute has when the device first starts or after a factory reset.
   - Access Control: defines read/write permissions for the attribute.
   - Review Read/Write Status: shows whether an attribute is declared as readable, writable, or both, as in the Matter specification and your product's design. The ZAP tool reflects these properties.

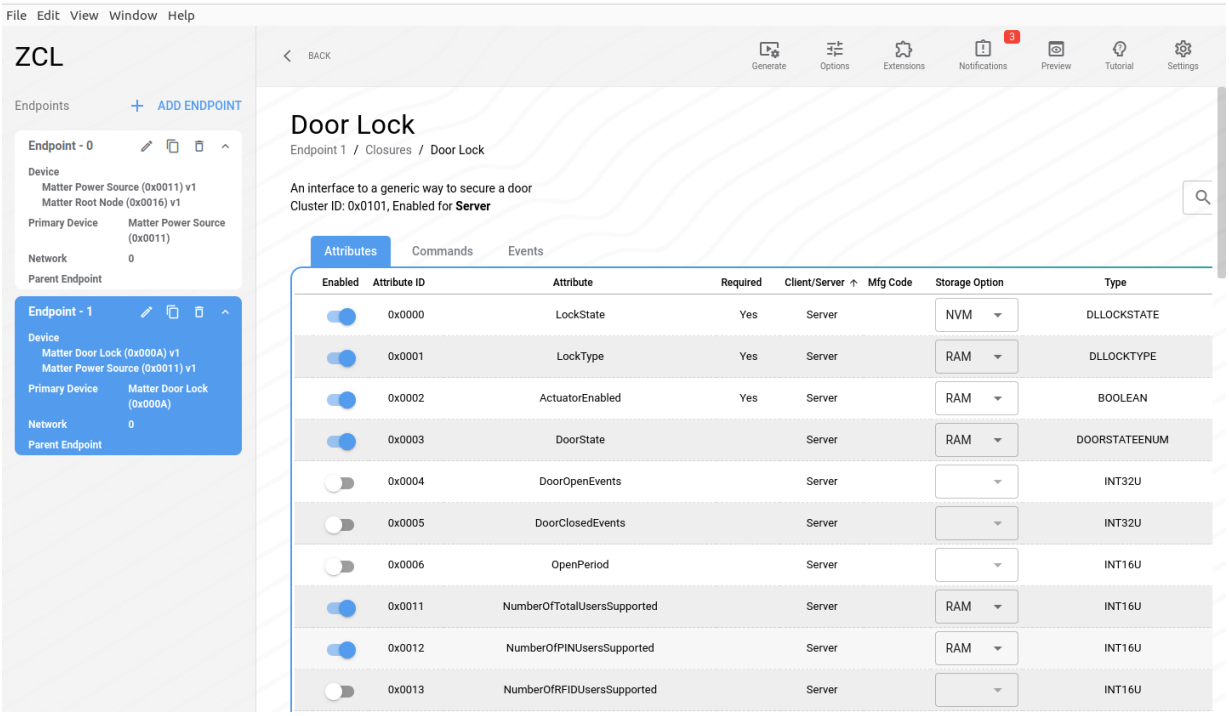

Figure 15. Door Lock Attributes

5.1.2 Generate Code Using ZAP

After configuring the attributes (and command/event) configurations in ZAP, you can use the tool to generate the source code. This process typically creates files that define your device's data model, provide access functions for attributes, and include placeholders or stubs for you to implement the actual device logic (how to physically lock/unlock the door when a LockDoor command is received).
The generated code ensures that your device's software stack accurately reflects the capabilities you declared in ZAP, laying the groundwork for successful Matter certification.

5.2 Add Door Lock Cluster in ZAP Tool

This section provides a step-by-step guide on how to configure your door lock device's data model in the ZAP tool, focusing specifically on the Door Lock Cluster. This process ensures your device properly exposes its functionalities to the Matter ecosystem.
To add Door Lock Cluster:

1. Launch ZAP and load your project.
2. Go to the Functional Endpoint.
3. Enable the Door Lock Cluster: Within Endpoint 1's configuration area, you'll find a list of available clusters. Locate the Door Lock Cluster (Cluster ID: 0x0101). In the **Enabled** column for this cluster, select **Server** from the dropdown menu. This designates your device as the entity providing door lock functionality.
4. Configure Door Lock attributes: After enabling the cluster, click the settings icon (⚙) next to the Door Lock Cluster to open its detailed configuration panel. Go the **Attributes** tab (or section) within this panel, for details see Figure 16.
   - Mandatory attributes:
     - Endpoint 0: this is a special, mandatory endpoint for all Matter devices. This is the mandatory "root node"
     - LockState (ID: 0x0000): reports the current state of the lock (locked, unlocked). Select the corresponding checkbox. In Figure 16, ZAP often shows mandatory attributes or prevents them from being disabled. For **Storage Option**, select **NVM** (Non-Volatile Memory). It is paramount that your door's lock state persists across power cycles. If power is lost and restored, the lock must return to its last known state. You might set an initial **Default Value** here, typically representing the UNLOCKED or LOCKED state on factory reset.

- LockType (ID: 0x0001): defines the physical type of lock (deadbolt, handle lock, latch bolt). Ensure its checkbox is enabled. Select **NVM** from the **Storage Option**, as the physical type of lock does not change after manufacturing. You can set a Default Value that matches your product's lock type.

- Optional Attributes (enable only if implemented in firmware):
  - DoorState (ID: 0x0003): If your door lock includes a sensor that detects whether the physical door is open or closed, enable this attribute. For **Storage Option**, select **External/Application-Managed**. This means your device's firmware actively reads the sensor and updates this attribute in real-time. ZAP generates the necessary accessor functions for your application code to interact with this attribute, but it does not manage its storage internally.
  - NumberOfTotalUsers (ID: 0x0011), NumberOfPINUsers (ID: 0x0012): If your lock supports managing multiple user PINs or access codes remotely, enable these attributes to report the capacity. These is typically stored in NVM as they reflect the lock's persistent configuration.

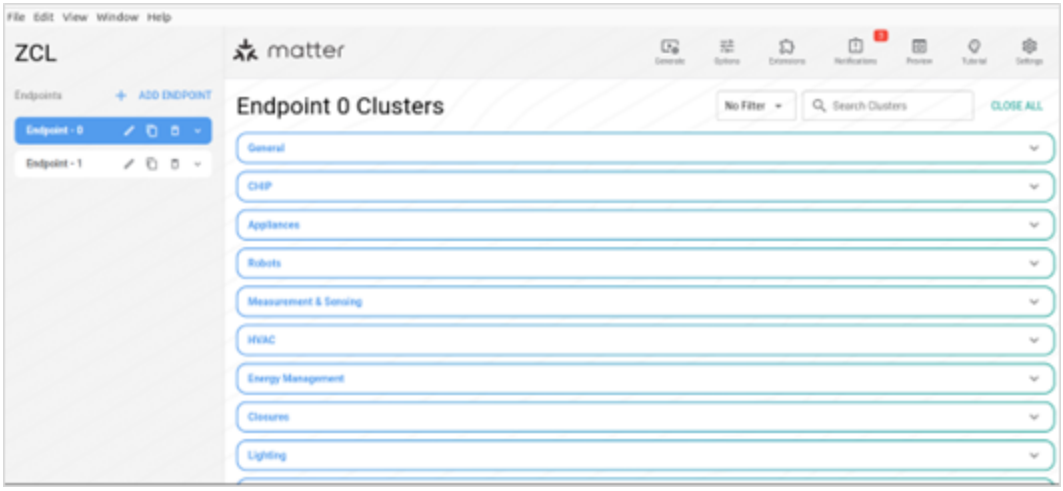

Figure 16. Door Lock Attributes in ZAP tool

- Other Optional attributes: only enable those that your door lock's firmware implements to avoid failed certification tests
- Configure Door Lock Commands: switch to the Commands tab within the Door Lock Cluster's configuration.
- Mandatory Commands:
  - LockDoor (ID: 0x0000): allows a controller to remotely lock the door. Select the **Enabled** checkbox. Your firmware must implement the logic to physically actuate the lock when this command is received.
  - UnlockDoor (ID: 0x0001): allows a controller to remotely unlock the door. Select the **Enabled** checkbox. Your firmware must implement the logic to physically actuate the unlock mechanism.
- Optional Commands (enable only if implemented):
  - SetWeekDaySchedule (ID: 0x0B), GetWeekdaySchedule (ID: 0x0C), ClearWeekDaySchedule (ID: 0x0D): If your lock supports the time-based access schedules for a door lock, enable these commands. They allow a system to grant or revoke access to specific users on days of the week and during defined time windows. Your firmware needs to implement the backend logic for this command.
  - Other Optional Commands: enable any other commands listed if your device supports them (ClearAllUsers, SetCredential).

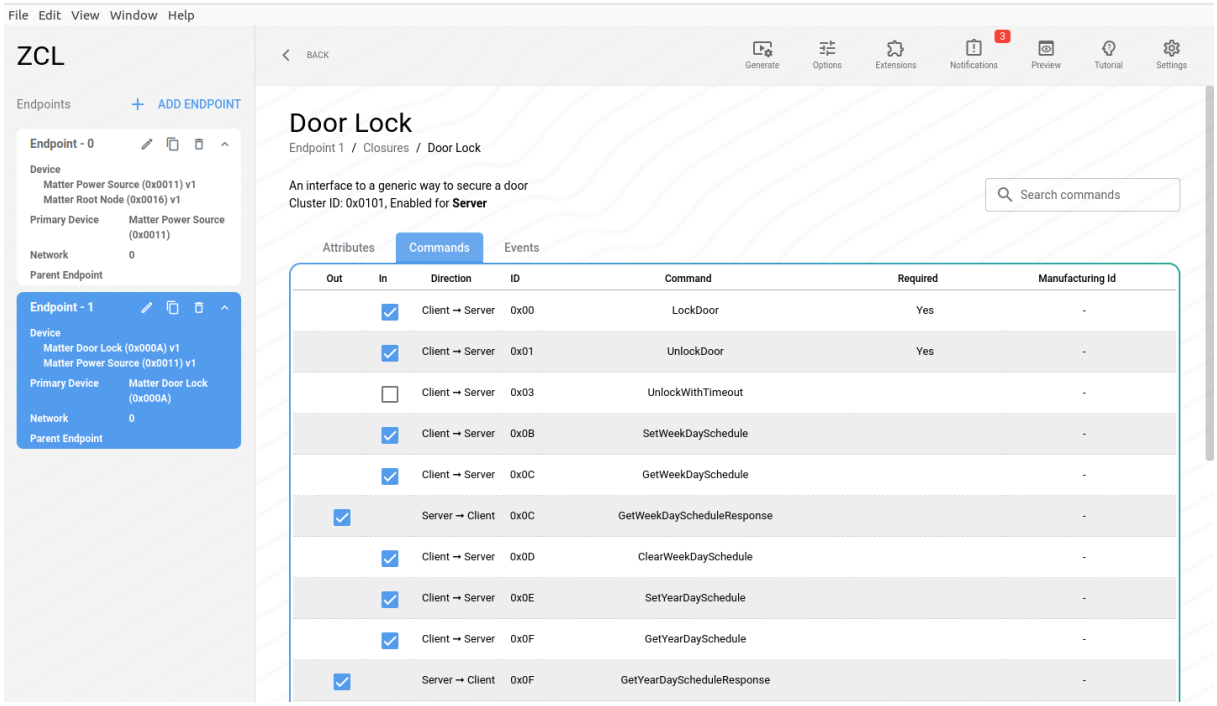

Figure 17. Door Lock Commands

To configure Door Lock events:

- a. Go to the **Events** tab within the Door Lock Cluster's configuration:
  - Optional events (enable only if implemented):
    - DoorLockAlarm (ID: 0x0000): signals security alarms (tamper detection, forced door). If your device generates these alerts, enable this event.

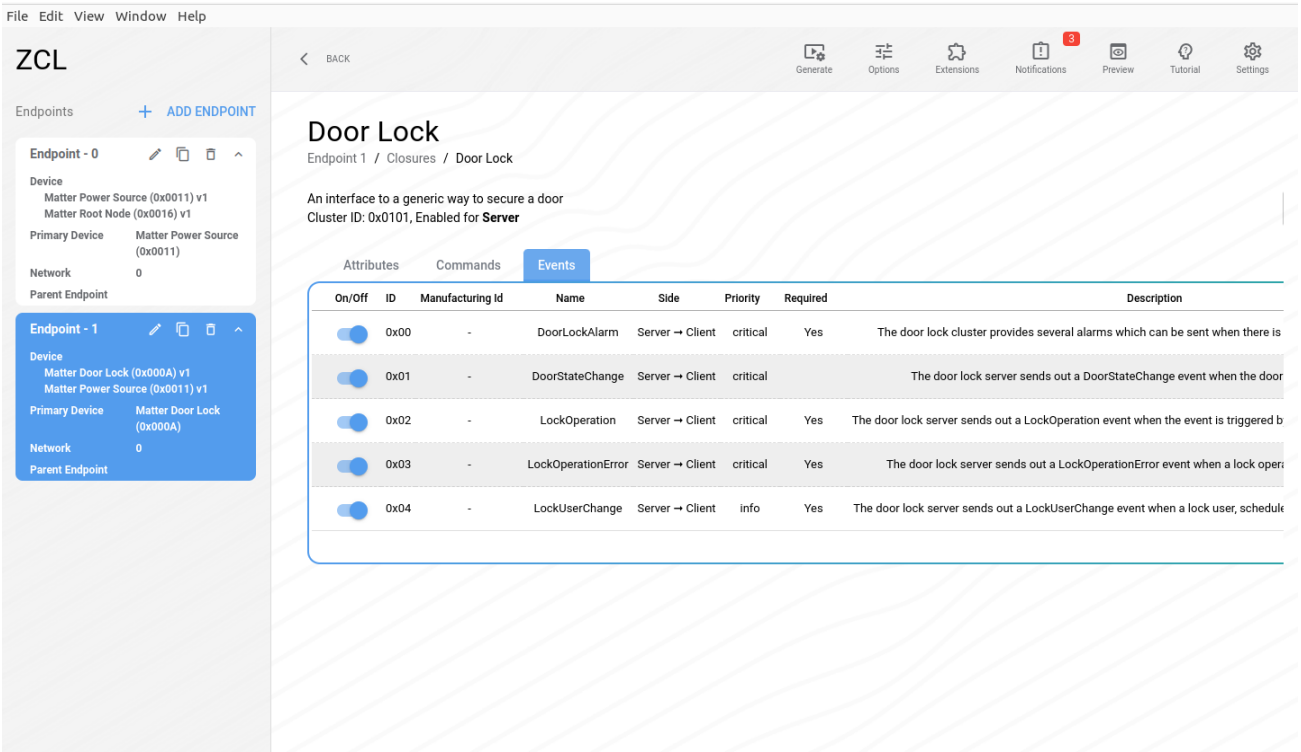

Figure 18. Door Lock Events

- When you configure all the relevant attributes, commands, and events for your Door Lock Cluster (and any other clusters), save your .zap file.
- To generate the zap file, use the following command:
  ./scripts/tools/zap/generate.py <.zap>
- To generate code, go to **.matter file** location and give the following command:
  ../../../scripts/codepregen.py --input-glob "*lock-app*" out/.

This produces the Matter SDK-specific source code files that reflect your configurations. These files include the data structures for your attributes and stubs for the command handlers, which your application firmware implements to provide the actual physical control and status reporting. Correctly configuring the Door Lock Cluster in ZAP is a pivotal step. It directly dictates the device's behavior on the Matter network and is fundamental to passing the rigorous certification tests.

6. Test Setup

Figure 19 shows a typical setup required for testing matter DUT against TH tool for certification:

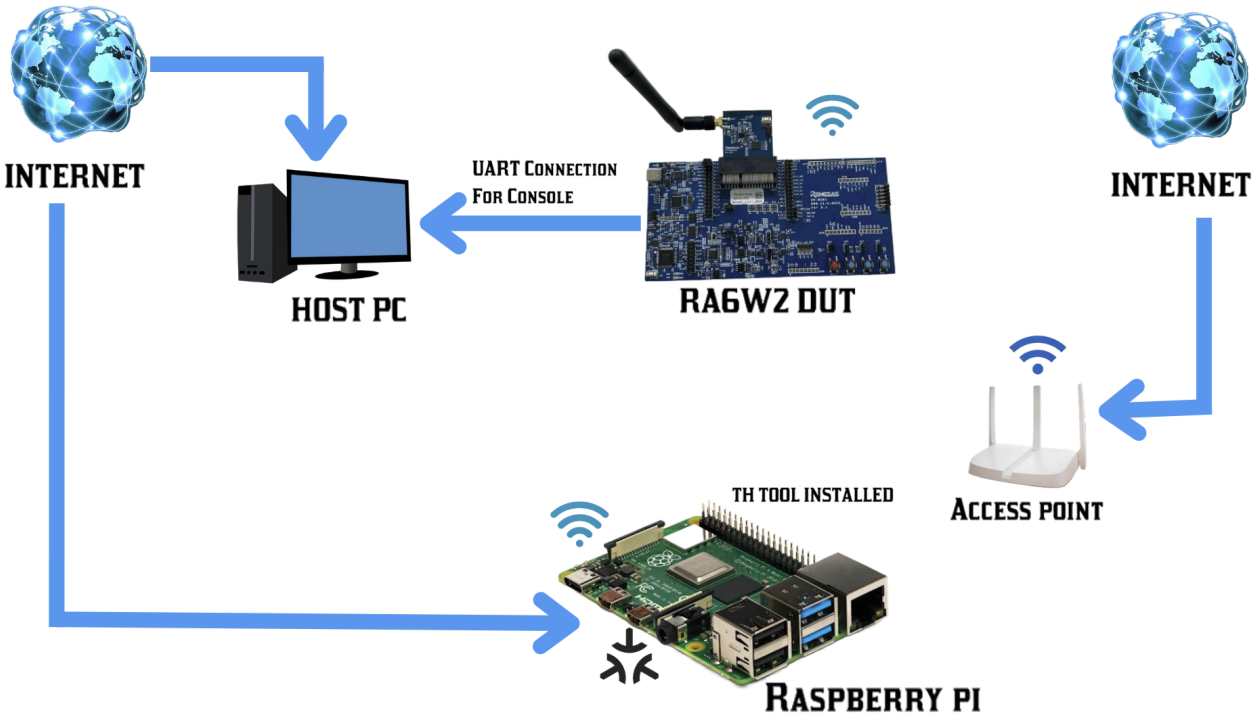

Figure 19. Matter Certification setup

- RA6W2 DUT: is a Wi-Fi-enabled embedded device (RA6W2), which communicates with the network through an access point. It is also connected through UART to the Host PC for console access, debugging, and command-line interactions.
- Host PC: is used primarily for monitoring the DUT and interacting with it through a UART console. It can also access the internet and may be used to analyse logs or perform auxiliary tasks during testing.
- Access Point (AP): DUT and Raspberry Pi are connected to the same local network through a Wi-Fi access point. The AP also provides internet access to these devices, which is essential for certain tests involving connectivity or cloud interactions. Ensure IPv6 is enabled on the Access Point. Matter protocol requires IPv6 (link-local and global) for commissioning and secure communication.
- Raspberry Pi (Test Harness Node): acts as the Test Harness (TH) node and has the official CSA Test Harness tool installed. It communicates with the DUT over the same Wi-Fi network and performs automated Matter protocol compliance tests. The CSA Test Harness tool must be installed and configured following the official guidelines provided in the CSA Matter Test Harness User Guide (PDF) available to CSA members through the Connectivity Standards Alliance (CSA) Portal.
- Connect the DUT and the RPI are connected to the same network, IPv6 is enabled in the Access point and IPv6 is leased on to the DUT and RPI as well. Also ensure that the RPI has a built-in Bluetooth LE controller (hci0) and that it is detected.

7. Matter Device Attestation Certificates

Matter's robust security model is built upon a chain of trust established through various cryptographic certificates. These certificates play a critical role in verifying the authenticity and integrity of a Matter device during the commissioning process, ensuring that only genuine, CSA-certified products can join a Matter fabric. Understanding the roles of Device Attestation Certificates (DACs), Product Attestation Intermediate (PAI) certificates, and Product Attestation Authority (PAA) certificates are fundamental to Matter security and certification.

- Device Attestation Certificates (DAC): the Device Attestation Certificate (DAC) is unique to each individual Matter product. It's akin to a birth certificate for your specific device, proving its authenticity and that it was manufactured by a legitimate vendor. The DAC is signed by a Product Attestation Intermediate (PAI) certificate, forming the next link in the chain of trust.
- Product Attestation Intermediate (PAI) Certificates: the Product Attestation Intermediate (PAI) certificate acts as an intermediary certificate that links multiple DACs to a single Product Attestation Authority (PAA) certificate. Think of it as a factory-level certificate that signs all the DACs for a specific product line or a batch of products from a manufacturer. The PAI signs DACs and is signed by a PAA certificate.
- Product Attestation Authority (PAA) Certificates: the Product Attestation Authority (PAA) certificate sits at the root of the Matter device attestation chain of trust. It is the ultimate trust anchor for a manufacturer, issued by a root Certificate Authority approved by the Connectivity Standards Alliance (CSA). The PAA is self-signed or signed by a root CA, and it signs PAI certificates.

During the Matter commissioning process, the commissioner (a smartphone app, smart hub, or the Test Harness) performs a critical device attestation verification. This involves:

- Retrieving the DAC: the commissioner requests the DAC from the DUT through a secure channel (usually over Bluetooth LE during the initial commissioning phase).
- Building the Certificate Chain: the commissioner then receives the PAI that signed the DAC, and finally, the PAA that signed the PAI. This forms a complete chain: DAC ← PAI ← PAA.
- Verification: the commissioner verifies the cryptographic signatures at each step of this chain, ensuring that:
  - The DAC is indeed signed by the PAI.
  - The PAI is indeed signed by the PAA.
  - The PAA is a trusted root certificate (pre-loaded and verified by the commissioner, often updated through the Distributed Compliance Ledger - DCL).
  - The **Vendor ID and Product ID** in the DAC match what the device is reporting through its Basic Information Cluster.
- Trust Establishment: If the entire chain is valid and verifiable, the commissioner establishes trust with the DUT, allowing it to securely join the Matter fabric.

7.1 Generate Certificate

Your DUT must be pre-provisioned with its unique DAC (and often its corresponding PAI key) during its manufacturing process or during firmware flashing for development. To generate the certificates for testing:

1. Pull latest Matter 1.4 from `connectedhomeip` repo

   ```
   cd connectedhomeip
   git checkout tags/v1.4.0.0
   git pull
   ```

2. Build chip-cert, which is the tool used for several operations on credentials for Matter devices.

   ```
   cd src/credentials
   source ../../scripts/activate.sh
   gn gen out
   ninja -C out
   ```

3. Export your custom VID/PID as environment variables to decrease chances of clerical error when editing your command arguments, here the VID for Renesas Electronics Corporation is 0x1562:

   ```
   export VID=0x1562
   export PID=0x0005
   ```

7.1.1 Generate CD

Generate the CD using chip-cert. Currently the Commissioner only validates that the VID and PID match the data exposed elsewhere by the device: the Basic Information Cluster, DAC and DAC origin (when it has it). You may leave the other fields unchanged:

```
src/credentials/out/chip-cert gen-cd \
--key credentials/test/certification-declaration/Chip-Test-CD-Signing-Key.pem \
--cert credentials/test/certification-declaration/Chip-Test-CD-Signing-Cert.pem \
--out credentials/test/certification-declaration/Chip-Test-CD-${VID}-${PID}.der \
--format-version "1" \
--vendor-id "${VID}" \
--product-id "${PID}" \
--device-type-id "0x1234" \
--certificate-id "ZIG20141ZB330001-24" \
--security-level "0" \
--security-info "0" \
--version-number "9876" \
--certification-type "1"
```

7.1.2 Generate PAA

```
src/credentials/out/chip-cert gen-att-cert \
--type a \
--subject-cn "Renesas PAA CN" \
--subject-vid "${VID}" \
--lifetime "4294967295" \
--out-key credentials/test/attestation/Chip-Test-PAA-${VID}-Key.pem \
--out credentials/test/attestation/Chip-Test-PAA-${VID}-Cert.pem
```

7.1.3 Generate PAI

```
src/credentials/out/chip-cert gen-att-cert --type i \
--subject-cn "Renesas Test PAI" \
--subject-vid "${VID}" \
--lifetime "4294967295" \
--ca-key credentials/test/attestation/Chip-Test-PAA-${VID}-Key.pem \
--ca-cert credentials/test/attestation/Chip-Test-PAA-${VID}-Cert.pem \
--out-key credentials/test/attestation/"test-PAI-${VID}-key".pem \
--out credentials/test/attestation/"test-PAI-${VID}-cert".pem
```

7.1.4 Generate DAC

```
src/credentials/out/chip-cert gen-att-cert --type d \
--subject-cn "Renesas Test DAC 0" \
--subject-vid "${VID}" \
--subject-pid "${PID}" \
--lifetime "4294967295" \
--ca-key credentials/test/attestation/"test-PAI-${VID}-key".pem \
--ca-cert credentials/test/attestation/"test-PAI-${VID}-cert".pem \
--out-key credentials/test/attestation/"test-DAC-${VID}-${PID}-key".pem \
--out credentials/test/attestation/"test-DAC-${VID}-${PID}-cert".pem
```

To generate DAC public key from DAC cert:

```
openssl x509 -in test-DAC-${VID}-${PID}-cert.pem -pubkey -noout > test-DAC-${VID}-${PID}-pub.pem
```

You can also convert the generated certs from .PEM format to .DER format using openssl:

```
openssl x509 -outform der -in Chip-Test-PAA-${VID}-Cert.pem -out Chip-Test-PAA-${VID}-Cert.der
```

7.1.5 Verify DAC, PAA, PAI Chain

If no errors appear in the output, it means that the certificate attestation chain is successfully verified:

```
src/credentials/out/chip-cert validate-att-cert \
--dac credentials/test/attestation/"test-DAC-${VID}-${PID}-cert".pem \
--pai credentials/test/attestation/"test-PAI-${VID}-cert".pem \
--paa credentials/test/attestation/Chip-Test-PAA-${VID}-Cert.pem
```

You can also see Matter test certificates, Google's developer guidance for complete assistance on certificate generation.
The default Certificates and Keys can be replaced with the generated Certificates and Keys in the code after converting .der into array format (test-PAI-${VID}-cert.der, test-DAC-${VID}-${PID}-cert.der, test-DAC-${VID}-${PID}-pub.der, test-DAC-${VID}-${PID}-key.der and cd.bin).
The array gPaiCert[], gDacCert[], gDacPubKey[], gDacPrivKey[] and kCdForAllExamples[] should be filled with the generated Certificates and Keys in:

```
<root_directory> \ra\matter\connectedhomeip\examples\platform\renesas\RnDeviceAttestationCreds.cpp.
```

7.2 e² studio Test VID/PID Modification and Certification Regeneration

Using e² studio, developers can modify the Vendor ID (VID) and Product ID (PID) during the development phase of Matter devices. When these VID/PID values are updated, e² studio automatically regenerates the Device Attestation Certificate (DAC) and rebuilds the complete certificate chain (DAC → PAI → PAA) to maintain conformity with Matter's security model.
This process allows development teams to test different product configurations, validate commissioning behavior, and verify attestation logic without needing CSA-issued production certificates. The regenerated certificates are intended **only for internal development and testing** and cannot be used for production hardware seeking Matter certification.

1. Open e²studio and load the Matter Application project template.
2. In the Stacks Configuration dialog, to add the desired test configuration (VID,PID), go to Matter app port (rm_matter_app_port_w) > Property > add Vendor ID: 0xFFF1, > Product ID: 0x8005, see Figure 20.

For more information, see Ref. 6

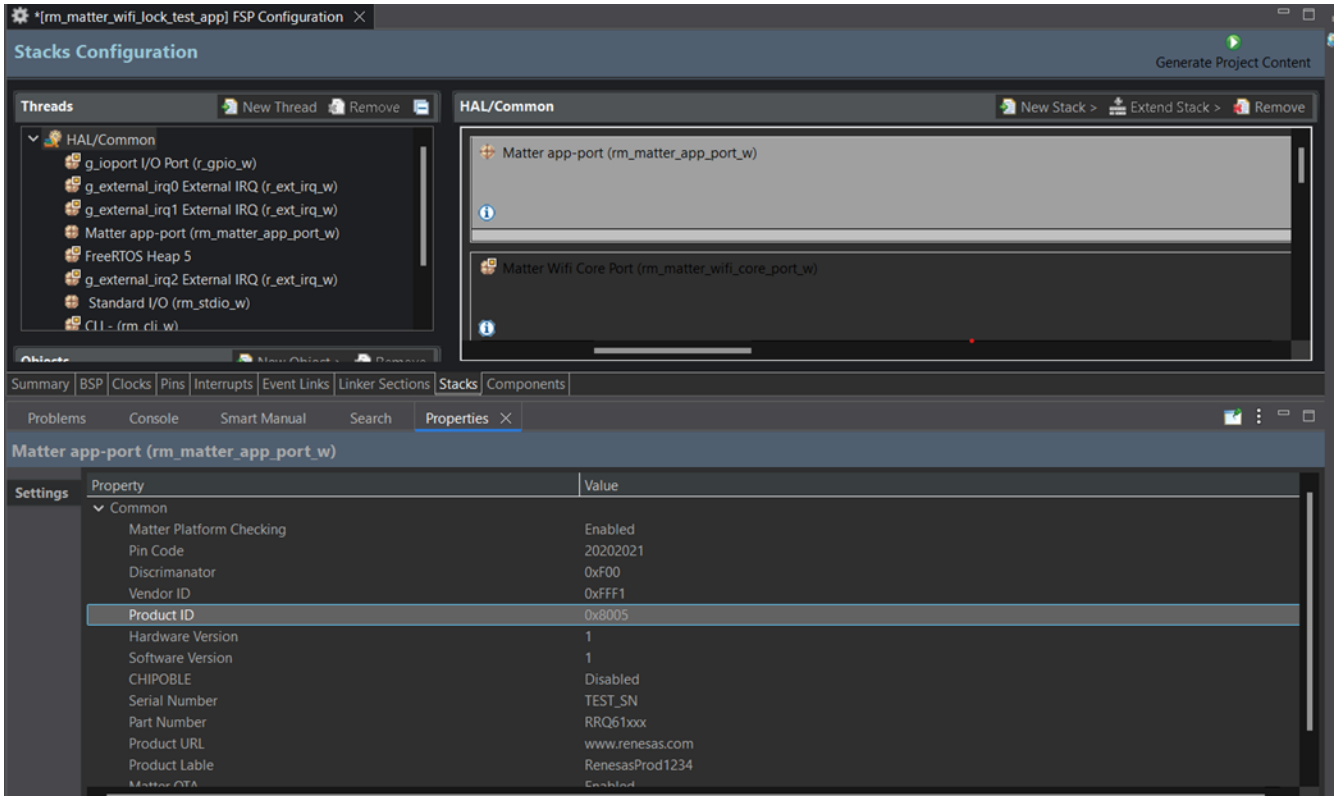

Figure 20. e^2 studio configuration

After completing the configuration changes, build the project and validate the generated image by performing a commissioning procedure. The required certificates with the test VID and PID are already pre-generated in .der format. These can be flashed to the RA6W1 S-Flash by following the procedure described in Section 7.3 Flash Matter Certificates on to S-Flash of RA6W1.
After flashing the image and writing the test certs successfully, you can commission the RA6W device using a Matter-compatible smartphone application, smart hub, or the Matter Test Harness tool to ensure proper functionality.

7.3 Flash Matter Certificates on to S-Flash of RA6W1

This section provides complete guide to using cli_programmer.exe for managing Matter-related certificates on the RA6W1 platform.

Pre-Requisites

- Use a Windows machine for executing the flash commands.
- All certificates must be in binary format (.der).
- Required CLI tool: cli_programmer.exe (included in the BA_new_flash_downloader package).
- Primary commands supported by cli_programmer.exe:
  - Mandatory (M): you must implement this feature. If your device does not support it, it fails certification.
  - read_qspi: Used to read a binary block from QSPI memory.
  - erase_qspi: Used to erase a region of QSPI memory.
  - write_qspi: Used to write binary data to QSPI memory.

To flash the certificates on to RA6W2:

1. Extract the contents of BA_new_flash_downloader.zip.
2. Open a Command Prompt dialog.
3. Go to the binaries folder and copy all the required certificate files (der) into this directory.
4. To erase the full range of the Matter certificate region in QSPI memory, execute the following command:
   .\cli_programmer.exe -s 230400 COM10 erase_qspi 0x00390000 0x00395000 (Provide the correct COM port of

your RA6W1 device). This command erases all 20 kB of memory (from 0x00390000 to 0x00394FFF), which is where all Matter-related certificates are stored.

Each certificate has a specific memory address. The following table shows the address and sample command for each, see Table 1.

Table 1. Flash commands

Certificate type	Address	File name	Command
Certification Declaration (CD)	0x00390000	Chip-Test-CD-${VID}-${PID}.der	.\cli_programmer. exe -s 230400 COM10 write_qspi 0x00390000 Chip-Test-CD-${VID}-${PID}.der
Device Attestation Certificate (DAC)	0x00391000	test-DAC-${VID}-${PID}-cert.der	.\cli_programmer. exe -s 230400 COM10 write_qspi 0x00391000 test-DAC-${VID}-${PID}-cert.der
Product Attestation Intermediate (PAI)	0x00392000	test-PAI-${VID}-cert.der	.\ cli_programmer. exe -s 230400 COM10 write_qspi 0x00392000 test-PAI-${VID}-cert.der
DAC Private Key	0x00393000	test-DAC-${VID}-${PID}-key.der	.\ cli_programmer. exe -s 230400 COM10 write_qspi 0x00393000 test-DAC-${VID}-${PID}-key.der
DAC Public Key	0x00394000	test-DAC-${VID}-${PID}-pub.pem	.\ cli_programmer. exe -s 230400 COM10 write_qspi 0x00394000 test-DAC-s${VID}-${PID}-pub.pem

To read a certificate (DAC) from SFlash and store it to a binary file, use the following command:

```
.\cli_programmer.exe -s 230400 COM10 read_qspi 0x00391000 data_o 1000
```

This reads 4096 B from address 0x00391000 and saves it to the file named data_o.

To update the DAC certificate:

1. Erase the memory region where DAC is stored:
   ```
   .\cli_programmer.exe -s 230400 COM10 erase_qspi 0x00391000 0x00392000
   ```
2. Flash the new DAC certificate, using the following command
   ```
   .\cli_programmer.exe -s 230400 COM10 write_qspi 0x00391000 dac_cert.der
   ```

7.4 Relevance to Test Harness

For certification testing with the Test Harness (TH):

- Pre-provisioning DUTs: your DUT must be pre-provisioned with its unique DAC (and often its corresponding PAI key) during its manufacturing process or during firmware flashing for development.
- PAA Certificate Trust Store: Test Harness itself needs access to a trust store containing the public PAA certificates of all manufacturers whose devices it needs to test. This allows the TH to verify the attestation chain of your DUT during the commissioning phase of the test run. You may need to manually copy your PAA certificates (in DER and PEM format) to a specific directory on the Test Harness machine (/var/paa-root-certs/).
- Certification Requirements: successful verification of this certificate chain is a fundamental requirement for Matter certification. The Test Harness includes specific test cases dedicated to validating the device attestation process. Any errors in the DAC, PAI, or PAA, or issues with their provisioning on the DUT, lead to test failures.

You can also use OpenSSL to inspect your generated certificates:

```
openssl x509 -noout -text -in \
credentials/test/attestation/test-DAC-${VID}-${PID}-cert.pem
```

To check the SKID of the PAA cert, use the following command:

```
openssl x509 -in Chip-Test-PAA-${VID}-Cert.pem -noout -text | grep "Subject Key Identifier" -A1
```

When the SKID is generated, you need to rename the Chip-Test-PAA-${VID}-Cert.pem with its SKID.pem and save it in the PAA-Trust storage path in the RPI.

In Figure 21, A1D27748606B2ED93C939EED359033056E238157 is the SKID of the PAA.cert and the PAA-Trust store path is /home/ubuntu/PAA.

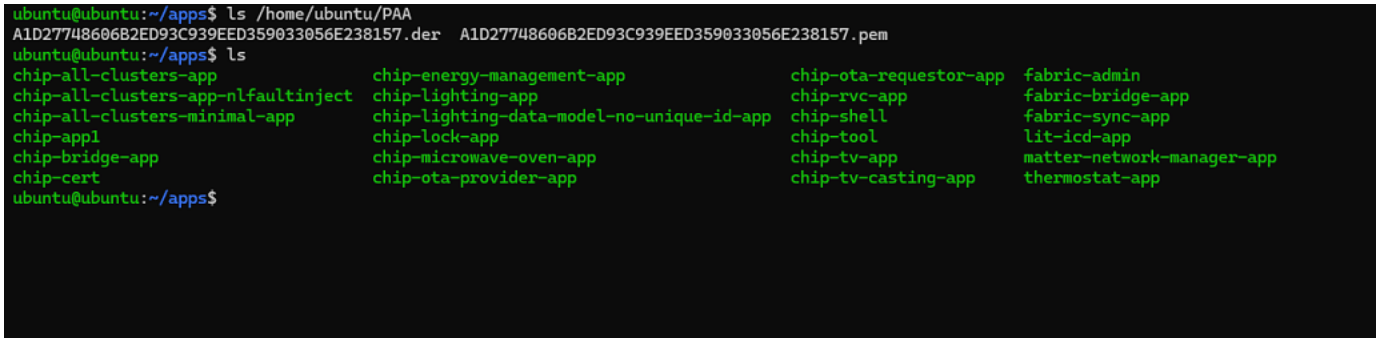

Figure 21. PAA Certificate

Proper management and provisioning of these certificates are therefore paramount for both device security and achieving Matter certification.

8. Test Harness Tool

The Matter Test Harness is the official and indispensable tool used for validating Matter device compliance with the CSA specification. It is the primary environment where your device's declared features, as outlined in your PICS plan, are rigorously tested to ensure interoperability and correct behavior within the Matter ecosystem. Understanding its role and operation is crucial for successful certification.
The core purpose of the Test Harness is to provide a standardized, automated, and comprehensive testing environment for Matter products. It acts as a Matter Controller that interacts with your Device Under Test (DUT) to verify that it correctly implements the clusters, attributes, commands, and events declared in your PICS plan.
This includes:

- Functional Verification: ensuring that features work as expected (a LockDoor command successfully locks the door).
- Behavioural Conformance: checking that the device responds according to the Matter specification (sending correct events, handling errors appropriately).
- Interoperability Assurance: confirming that the device can communicate and work seamlessly with other Matter-certified devices and controllers.
- PICS Validation: directly using your PICS document to select and execute relevant test cases, making the testing process efficient and targeted.

The Matter Test Harness environment typically comprises several key components:

- Test Harness Software: the core application, often a web-based UI running on a dedicated machine (like a Raspberry Pi or a Linux workstation), that orchestrates the testing. It's built on the Matter SDK and includes various test scripts (written in Python or YAML) that embody the certification requirements.
- Test Environment: includes the physical setup for testing, often involving a Matter-enabled device, Wi -Fi Access Point that provides the network for the DUT.
- DUT) Your Matter product (the door lock) that is being certified. The Test Harness communicates with this device.
- PICS Files: prepared PICS document is fed into the Test Harness, which uses it to dynamically determine which specific tests need to be run for your DUT.
- Logs and Reports: the Test Harness generates detailed logs of test execution and comprehensive test reports, indicating pass/fail status for each test case. These reports are essential for submission to the certification body.

8.1 Install Test Harness Tool

Setting up the Matter Test Harness (TH) and the necessary environment for certification testing is a multi-step process that requires careful attention to hardware, software, and network configuration.

Hardware Requirements

The Test Harness typically runs on a dedicated host machine, most commonly a Raspberry Pi 4 (or newer) or a Linux workstation (Ubuntu 20.04/22.04). Key hardware components include:

- Host Machine: Raspberry Pi 4 (or a Linux PC with sufficient resources).
- SD Card (for Raspberry Pi): At least 64 GB, Class 10 or higher for the operating system.
- Power Supply: Appropriate power supply for the Raspberry Pi.
- Network Connectivity: Ethernet connection for stability.

To install the Test harness tool for Mater 1.4 Certification, follow the instruction provided in the official CSA site or check the Test harness tool GitHub. To install the tool for Matter 1.4 certification, use the tag "v2.11+fall2024" during installation.
When the installation is completed, set your Test Harness machine and DUT on the same Matter network (Thread or Wi-Fi), and check if the Test Harness itself is available for commissioning. The Test Harness acts as a commissioner over IP to interact with the DUT once it is on the network. You can access the TH tool by or by taking SSH of the RPI using its network IP.

```
ubuntu@ubuntu:~/apps$ ls
chip-all-clusters-app                chip-energy-management-app                chip-ota-requestor-app  fabric-admin
chip-all-clusters-app-nlfaultinject  chip-lighting-app                         chip-rvc-app            fabric-bridge-app
chip-all-clusters-minimal-app        chip-lighting-data-model-no-unique-id-app chip-shell              fabric-sync-app
chip-app1                            chip-lock-app                             chip-tool               lit-icd-app
chip-bridge-app                      chip-microwave-oven-app                   chip-tv-app             matter-network-manager-app
chip-cert                            chip-ota-provider-app                     chip-tv-casting-app     thermostat-app
ubuntu@ubuntu:~/apps$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.27.10.11  netmask 255.255.255.0  broadcast 172.27.10.255
        inet6 fe80::da3a:ddff:fe91:942f  prefixlen 64  scopeid 0x20<link>
        ether d8:3a:dd:91:94:2f  txqueuelen 1000  (Ethernet)
        RX packets 9198436  bytes 1428858053 (1.4 GB)
        RX errors 0  dropped 290108  overruns 0  frame 0
        TX packets 210010  bytes 14935828 (14.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@ubuntu:~/apps$ iw dev
phy#0
        Unnamed/non-netdev interface
                wdev 0x2
                addr da:3a:dd:91:94:30
                type P2P-device
                txpower 31.00 dBm
        Interface wlan0
                ifindex 3
                wdev 0x1
                addr d8:3a:dd:91:94:30
                ssid RenesasMatter1
                type managed
                channel 11 (2462 MHz), width: 20 MHz, center1: 2462 MHz
                txpower 31.00 dBm
```

Figure 22. Test Harness tool

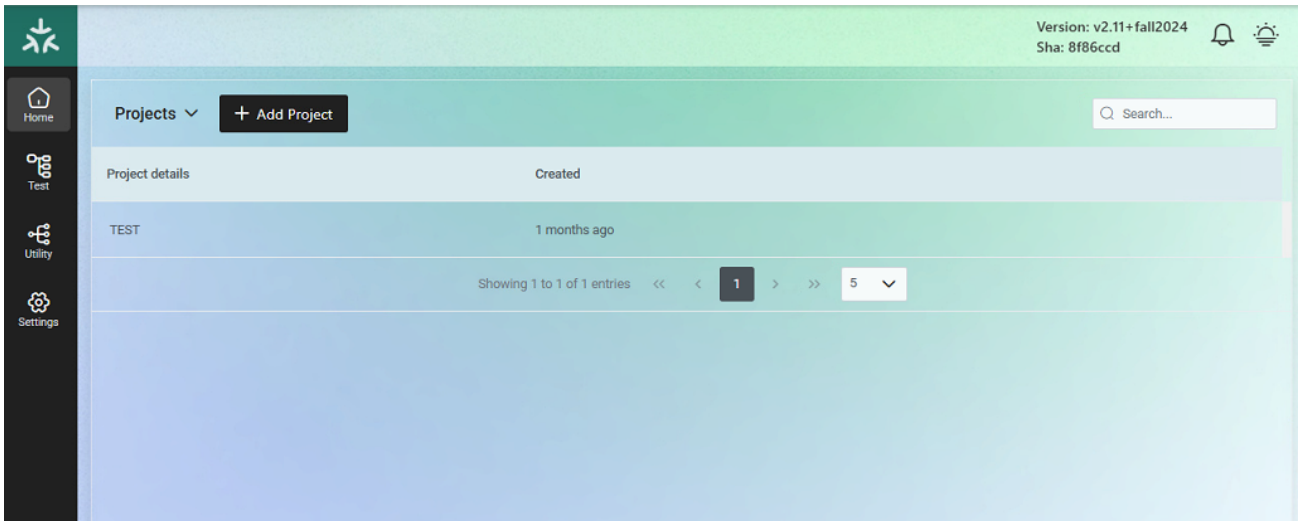

Figure 23. TH tool

After accessing the, upload your device's PICS file into the Test Harness. This step is critical as it informs the harness about your device's claimed capabilities.

8.2 DUT Preparation

To start testing the product that is the Door lock, load your DUT is with Matter firmware that corresponds to its PICS declaration and ZAP configuration.

- For details on setting up the RA6W2 DUT and its configuration, see Ref. 2
- For building Matter image to program RA6W2 DUT, see Ref. 3.

The DUT should be in a commissionable state (broadcasting Bluetooth LE advertisements with its discriminator and setup PIN code).

You need to have the DUT's setup PIN code and discriminator available, as you need them for commissioning through the Test Harness.

You also need to add the details of the PIN Code, Discriminator, SSID of the access point its connected to, the password, the type of pairing that your device supports (on network or Bluetooth LE-Wi-Fi) under the **Project Config**, see Figure 24.

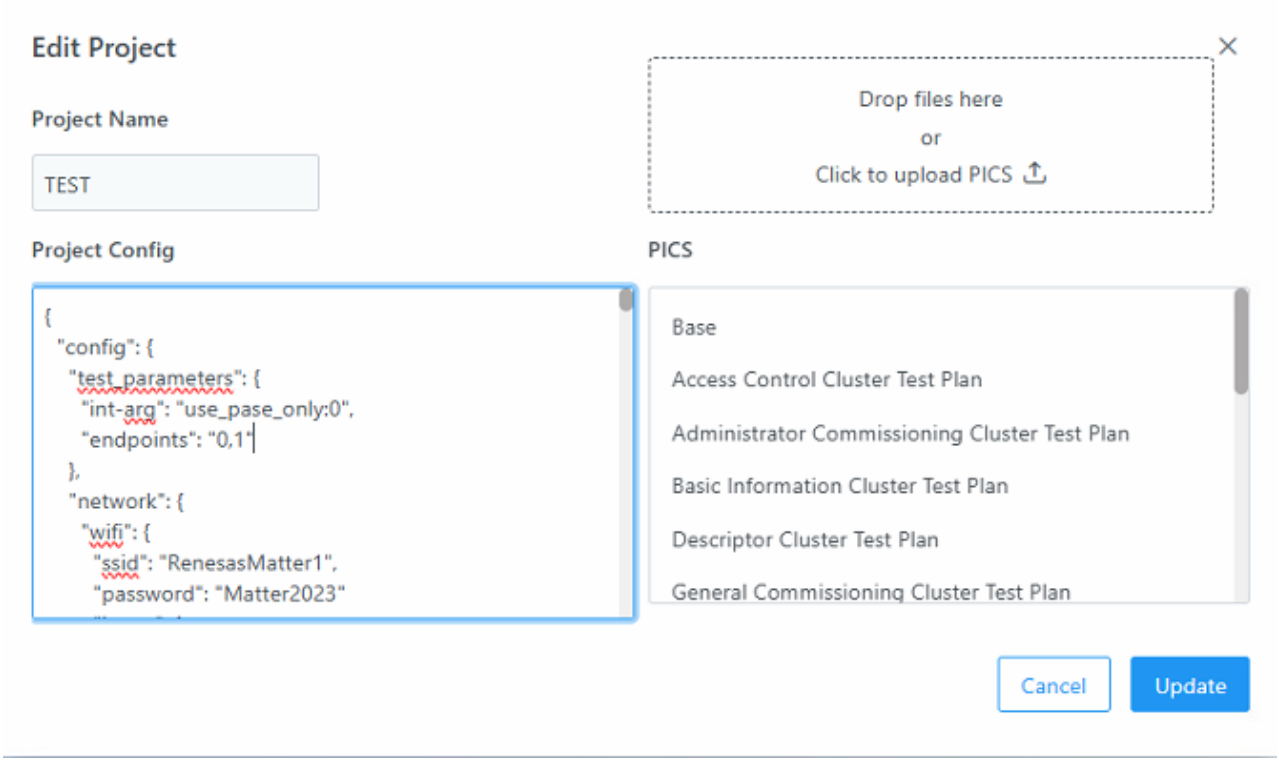

Figure 24. Project configuration in TH tool

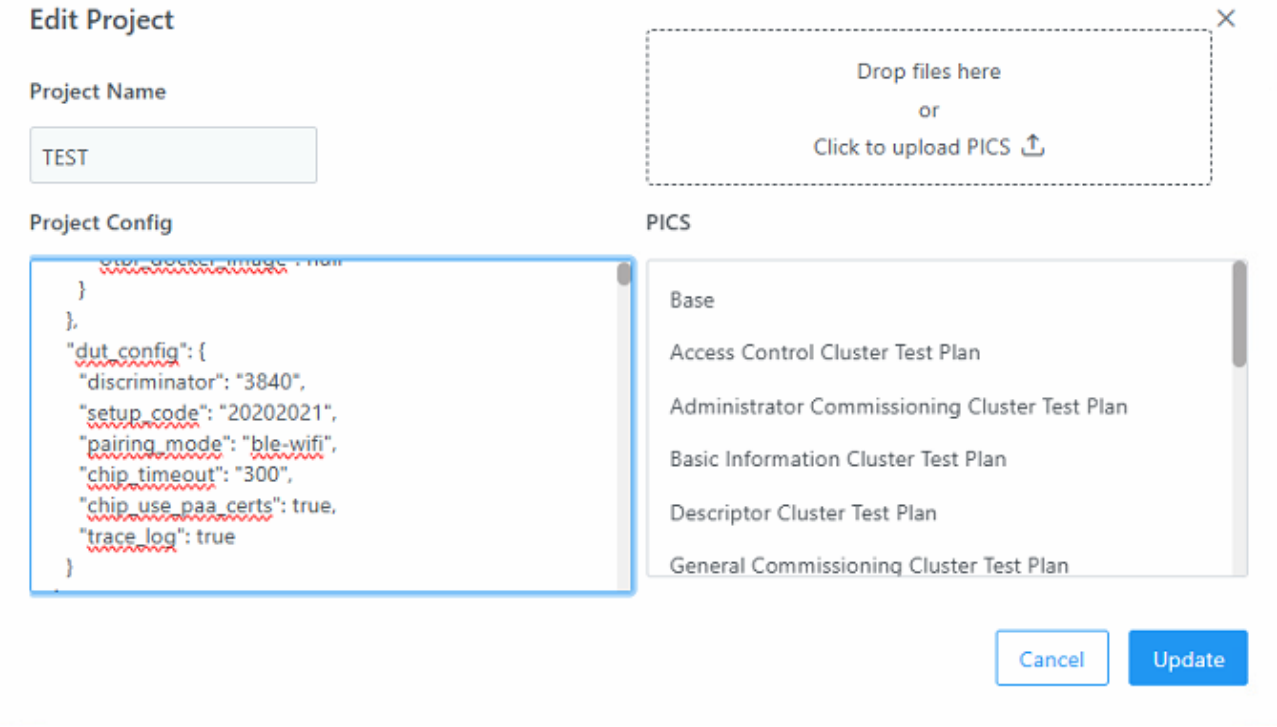

Figure 25. DUT configuration in TH tool

9. Test Case Execution

When your Matter Test Harness is set up and your DUT is ready, the next crucial step in the certification journey is to execute the test cases. The Matter certification suite in the Test Harness comprises various types of tests, each designed to validate different aspects of your device's compliance with the Matter specification. Understanding these categories is essential for effective pre-testing and official certification. The Test Harness runs tests that can be broadly categorized by their level of automation and the interaction required from the tester:

- Automated Tests: run entirely within the TH without any manual intervention from the operator.
- Semi-Automated Tests: involve a combination of automated steps by the TH and specific actions or verifications that the human operator must perform.
- Manual Tests: require the operator to perform a series of steps and then manually verify the DUT's behavior and input the results into the TH.
- Python Tests: while Python is the underlying language for many automated tests, this category specifically refers to tests written directly as Python scripts that might offer more flexibility or lower-level control, often executed directly or integrated into the harness.

Before the test execution, load the relevant PICS file to list the required test cases. Depending on the PICS file loaded, the test suites list is updated accordingly.

Before starting the test case execution, you also need to access the TH and create a project:

1. Click **Add Project**, see Figure 26.
2. In the Edit Project dialog, enter the project name as "Test Project" and edit the Project Config settings to provide additional details, see Figure 27.

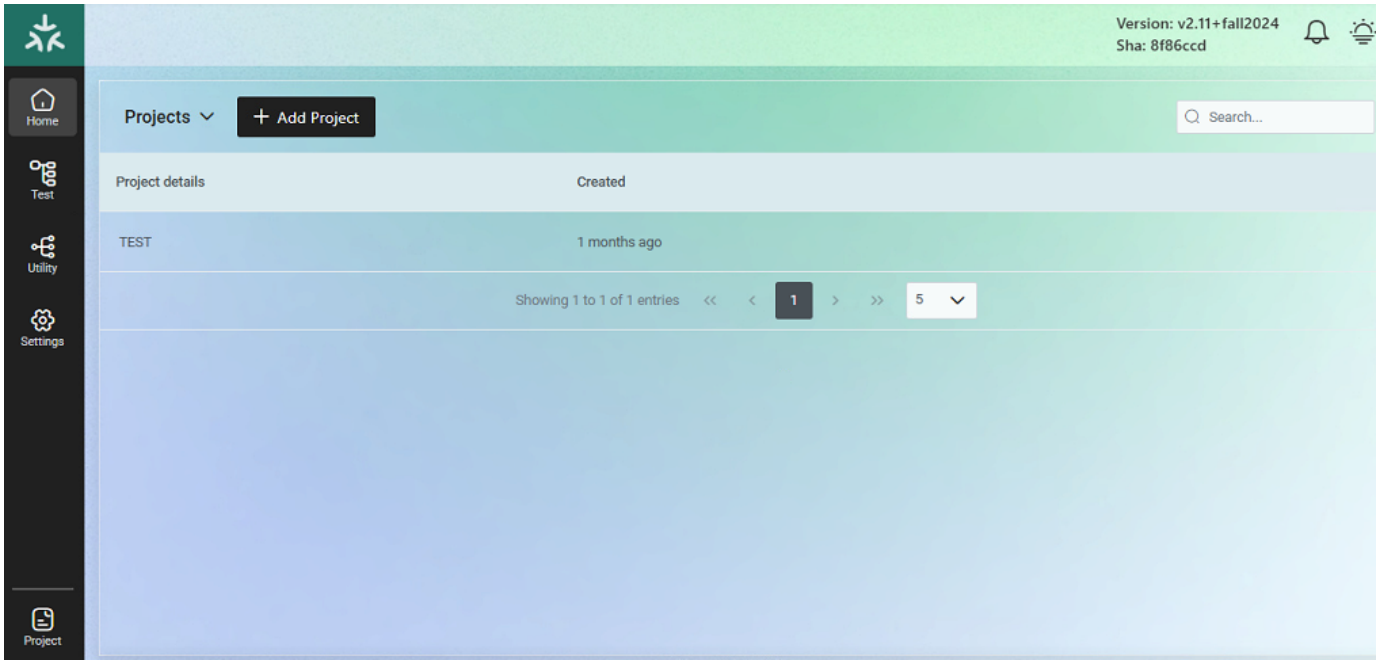

Figure 26. Create project in TH tool

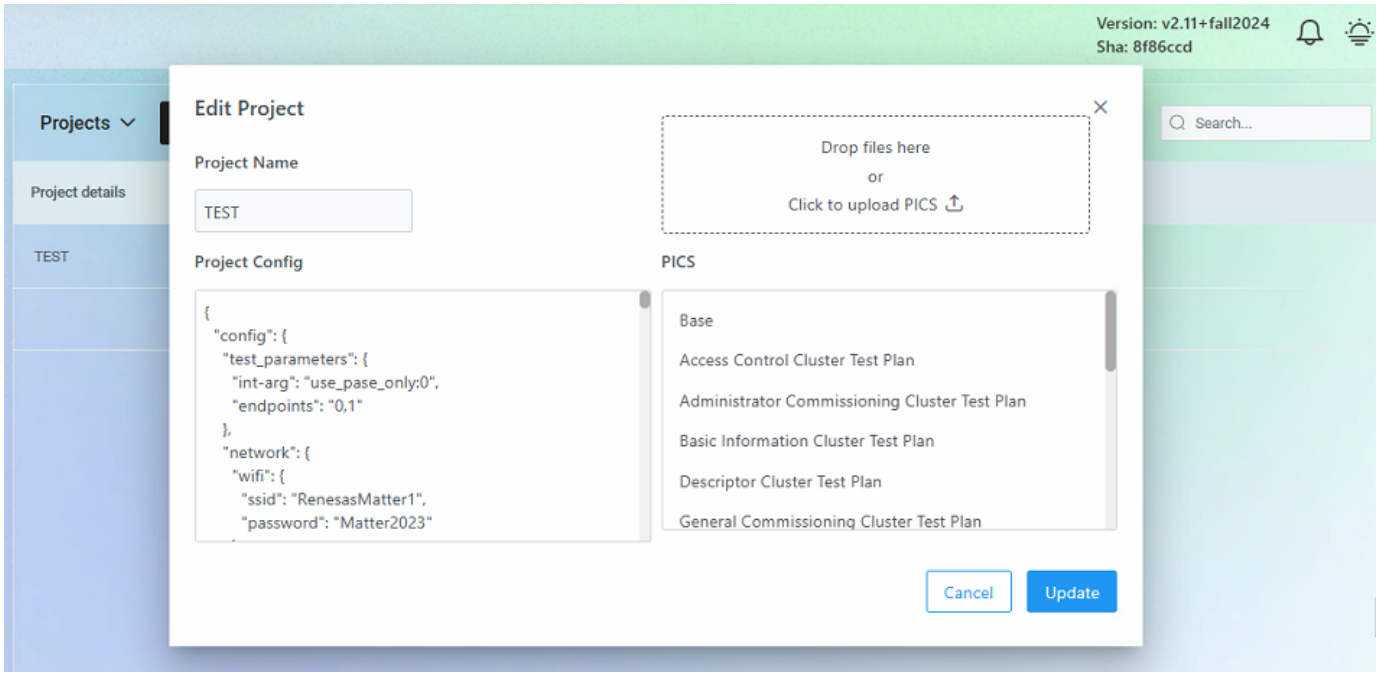

Figure 27. Edit Project dialog

3. To configure the pairing as Bluetooth LE - Wi-Fi method (RA6W2 supports Bluetooth LE provisioning), provide details: discriminator, setup code and set the Pairing mode as Bluetooth LE - Wi-Fi under "Project Config in the TH UI":

```
"dut_config": {
     "discriminator": "3840",
      "setup_code": "20202021",
      "pairing_mode": "ble-wifi",
      "chip_timeout": "300",
      "chip_use_paa_certs": true,
      "trace_log": true
    }
```

For the case that the DUT requires a PAA certificate to perform a pairing operation, input "true" for the flag "chip_tool_use_paa_certs" in the Project Config of TH UI, as shown in Figure 27.

4. Include the desired PAA certificates in the default path "/var/paa-root-certs/", in the Raspberry-Pi.
5. Input the test parameters like endpoint on the DUT where the cluster to be tested is implemented and input the manual pairing code or QR code parameter as well.

```
"test_parameters":{
"manual-code":"34970112332",
 "endpoints": "0,1"
}
```

6. To complete the creation, click **Update** and click **Create**, see Figure 27.
   The newly created project is listed under the **Project details**.
7. To configure the project to load the required PICS file for the cluster to be tested, click **Edit**, and select **Update**.
   Now the Test Project is ready for execution.
8. Click **Go to Test-Run** and create a new Test Run batch, see Figure 28.

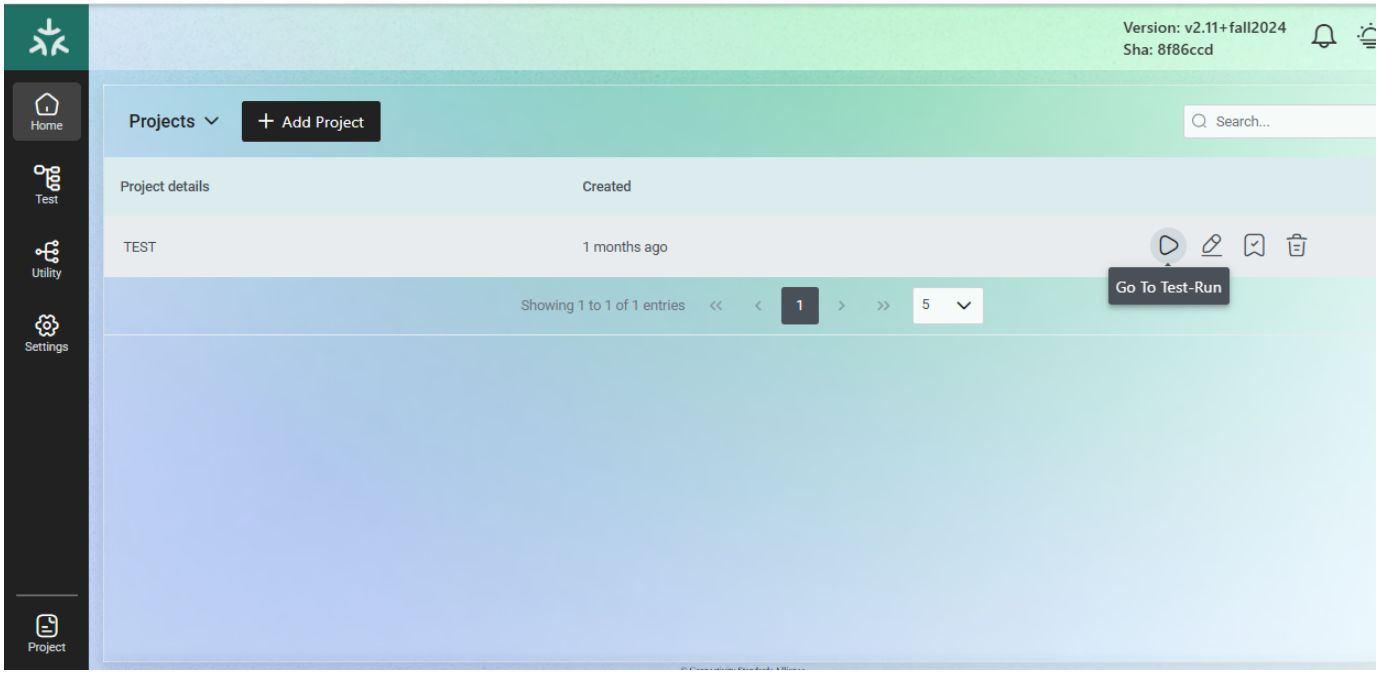

Figure 28. Test run selection

A Test Run can be created in Regular mode or Certification mode. The test cases are automatically selected based on the PICS files provided in the Project Configuration. For a Test Run in Regular mode, it is possible to change this selection, but in Certification mode that selection is unchangeable — a test case must be executed if and only if the PICS files indicate that it is applicable.
For internal testing, choose the Regular mode and proceed to the test run. Depending on the PICS file loaded, the test suite is updated accordingly.

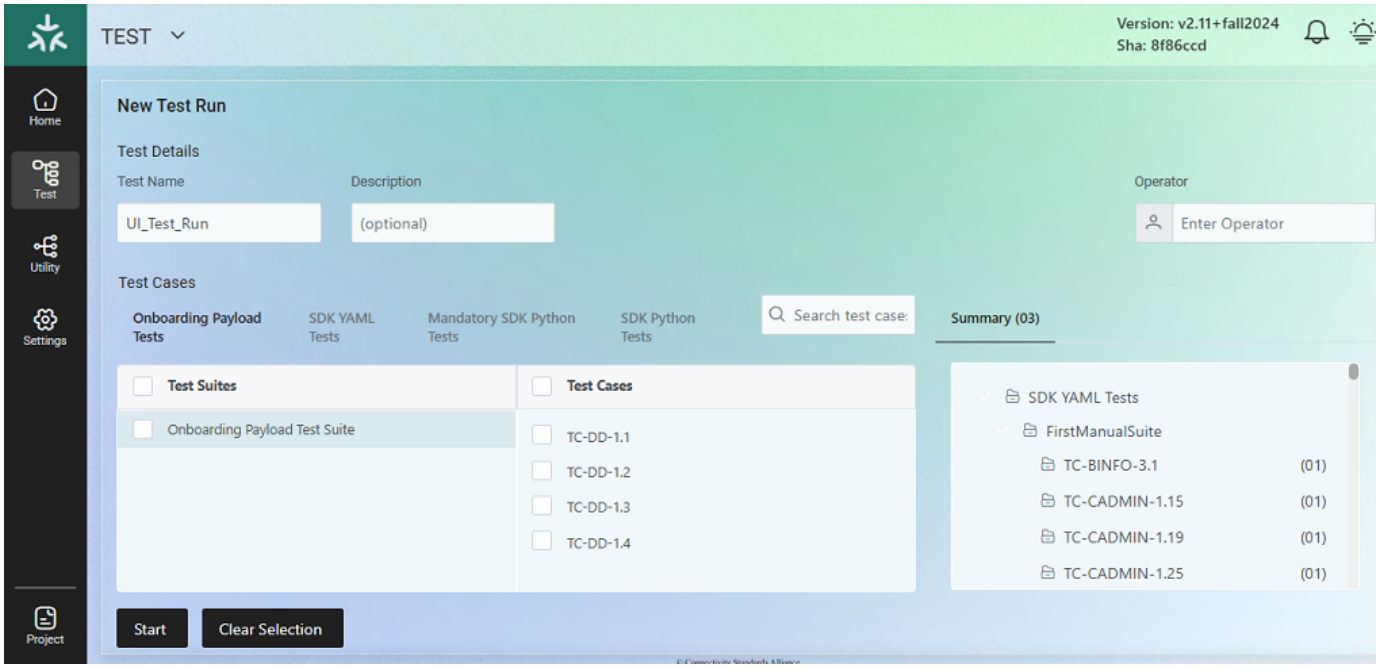

Figure 29. Test case selection

For further information on TH tool, test case execution, see Ref. 6

9.1 Automated Test Cases

To run the Automated test cases for certification, go to **SDK YAML Tests** tab. The automated and semi-automated test cases are listed in FirstChipToolSuite. The Automated test cases are listed as the TC-XX without any suffix, for example,TC-DRLK-1.1. Automated test case execution does not require any manual intervention.

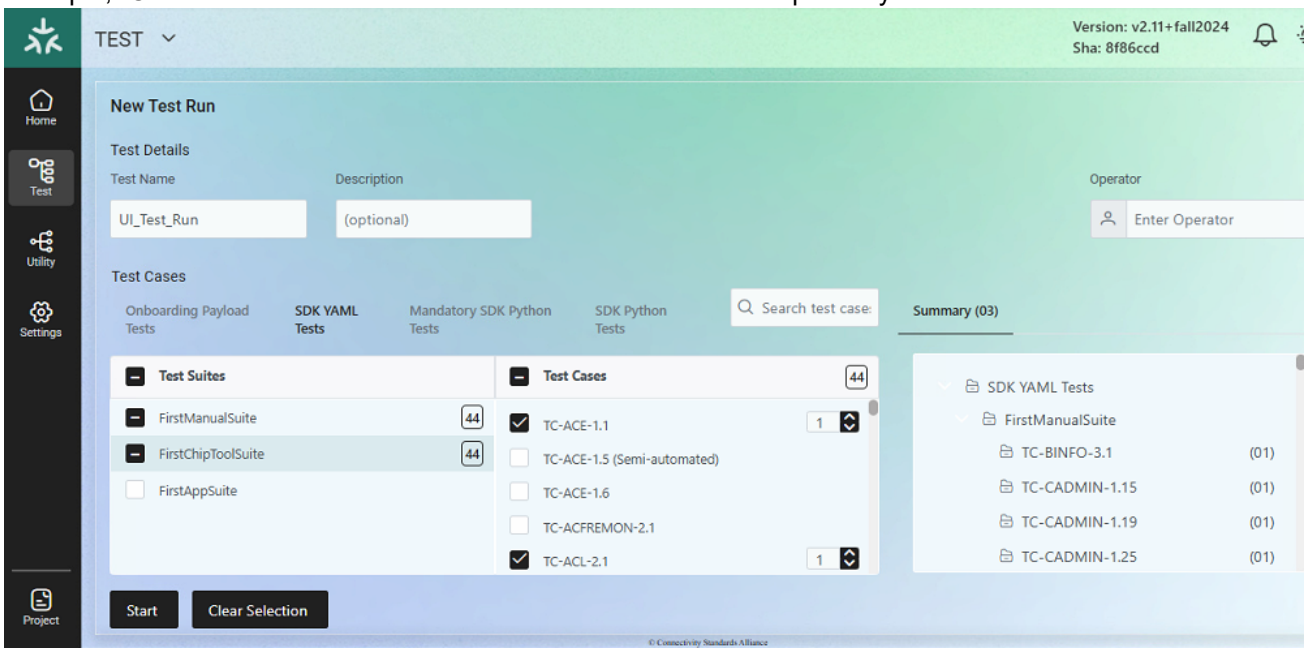

Figure 30. Automated test cases

To select and run the desired automated test cases:

1. Select the desired automated test suite or individual tests from TH UI.
2. There is a search option available to search for a particular test case.
3. When the desired test cases are selected, provide a test name, enter the operator's name (for internal run, you can provide any dummy operator name).
4. Bring up the TH by sending the following command./chip-all-clusters-app -Wi-Fi (when using the Bluetooth Le-Wi-Fi pairing) on the Raspberry Pi terminal.
5. Click **Start**.

```
Last login: Mon Aug 18 06:36:48 2025 from 172.27.10.188
ubuntu@ubuntu:~$ cd apps/
ubuntu@ubuntu:~/apps$ ls
chip-all-clusters-app                chip-energy-management-app                chip-ota-requestor-app  fabric-admin
chip-all-clusters-app-nlfaultinject  chip-lighting-app                        chip-rvc-app            fabric-bridge-app
chip-all-clusters-minimal-app        chip-lighting-data-model-no-unique-id-app chip-shell              fabric-sync-app
chip-app1                            chip-lock-app                             chip-tool               lit-icd-app
chip-bridge-app                      chip-microwave-oven-app                   chip-tv-app             matter-network-manager-app
chip-cert                            chip-ota-provider-app                     chip-tv-casting-app     thermostat-app
ubuntu@ubuntu:~/apps$
ubuntu@ubuntu:~/apps$
ubuntu@ubuntu:~/apps$ ./chip-all-clusters-app --wifi
[1755689374.696291][4006:4006] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1755689374.712668][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_kvs
[1755689374.735665][4006:4006] CHIP:DL: ChipLinuxStorage::Init: Attempt to re-initialize with KVS config file: /tmp/chip_kvs
[1755689374.767924][4006:4006] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1755689374.773016][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755689374.773293][4006:4006] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1755689374.779222][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755689374.779444][4006:4006] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1755689374.783365][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755689374.844876][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755689374.845005][4006:4006] CHIP:DL: NVS set: chip-factory/unique-id = "2550C5E429FC0209"
[1755689374.850841][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755689374.850947][4006:4006] CHIP:DL: NVS set: chip-factory/vendor-id = 65521 (0xFFF1)
[1755689374.858932][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755689374.859029][4006:4006] CHIP:DL: NVS set: chip-factory/product-id = 32769 (0x8001)
[1755689374.865423][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755689374.865522][4006:4006] CHIP:DL: NVS set: chip-counters/reboot-count = 1 (0x1)
[1755689374.873636][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755689374.873741][4006:4006] CHIP:DL: NVS set: chip-counters/total-operational-hours = 0 (0x0)
[1755689374.880515][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755689374.880612][4006:4006] CHIP:DL: NVS set: chip-counters/boot-reason = 0 (0x0)
[1755689374.898534][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755689374.898636][4006:4006] CHIP:DL: NVS set: chip-config/regulatory-location = 0 (0x0)
[1755689374.907377][4006:4006] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755689374.907498][4006:4006] CHIP:DL: NVS set: chip-config/location-capability = 2 (0x2)
```

Figure 31. Run all-cluster-app on RPI

The Test Harness takes over, executing the sequence of operations on the DUT and automatically checking for expected responses and behaviours. The results (Pass/Fail) are logged in real-time and compiled into the final report.

9.2 Semi-automated Test Cases

The Semi-automated test cases are listed as TC--XX(Semi-automated). During the Semi-automated test case execution, the TH executes a portion of the test, then pauses and provides instructions to the operator for a specific action or observation. When the action is performed, the operator confirms it in the Test Harness, allowing the automation to resume. Semi-automated test cases are listed in FirstChipToolSuite.
To run the Semi-automated test cases:

1. Select the required Semi-automated test case to be executed and ensure other test cases are not selected.
2. Bring up the TH by sending the following command. /chip-all-clusters-app -Wi-Fi (when using the Bluetooth Le-Wi-Fi pairing) on the Raspberry Pi terminal.
3. Click **Start**.
   The Test Harness begins an automated sequence. The operator performs the instructed action or observation on the DUT.
4. Check for the response of the command in the Chip-tool log and compare with the expected response from the TH prompt. The operator clicks the **Continue** or **Confirm** in the Test Harness. The Test Harness proceeds with the next automated steps, verifying the outcome of the manual action.
5. At the end of the test execution, upload the Chip-tool logs that were saved in the previous steps.

9.3 Manual Test Cases

Manual tests are used for complex interactions or subjective evaluations that cannot be reliably automated by the Test Harness. These tests provide step-by-step instructions for the operator to perform on the DUT and then require the operator to manually assess the outcome and record the pass/fail result directly in the Test Harness or a separate verification document.

The manual tests are listed under FirstManualSuite, see Figure 32.

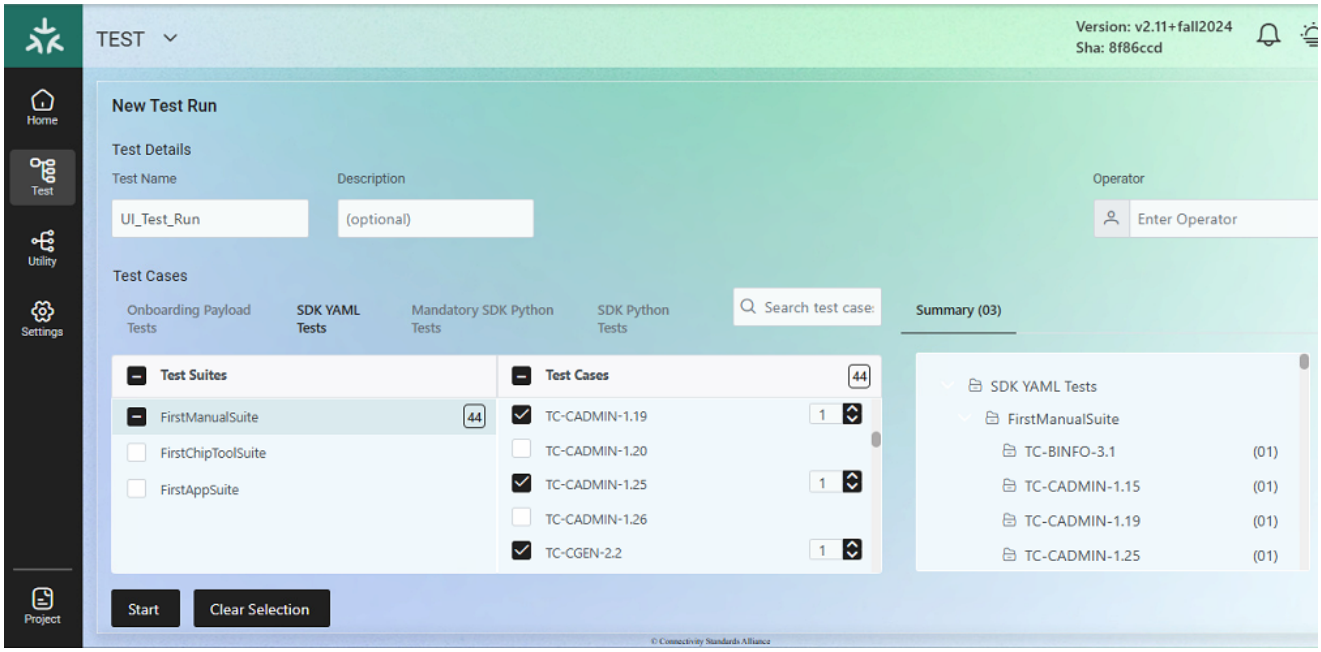

Figure 32. Manual test cases

You can also verify the manual test cases, see Ref. 6, an extensive Excel spreadsheet provided by the CSA that contains all the manual tests and steps to verify each test case.
For manual commissioning the RA6W2 DUT using Bluetooth LE-Wi-Fi method, use the following command in TH:

```
./chip-tool pairing ble-wifi <Node_ID> <Wi-Fi_SSID> <Wi-Fi PW>  <Setup Pincode> <Discriminator> --paa-trust-store-path </path to trust store> --trace_decode 1
```

9.4 Python Test Case

Many of the Matter certification tests, particularly the complex ones and the underlying logic for automated and semi-automated tests, are implemented as Python scripts. These are part of the Test Harness software.
These are Python scripts (located in connectedhomeip/src/python/tests within the SDK) that use Matter's Python bindings to directly interact with devices. While often integrated into the Test Harness UI, they can sometimes be run independently for debugging or custom testing.
You can run the Python test directly from the TH tool or you can also run them from inside the docker.

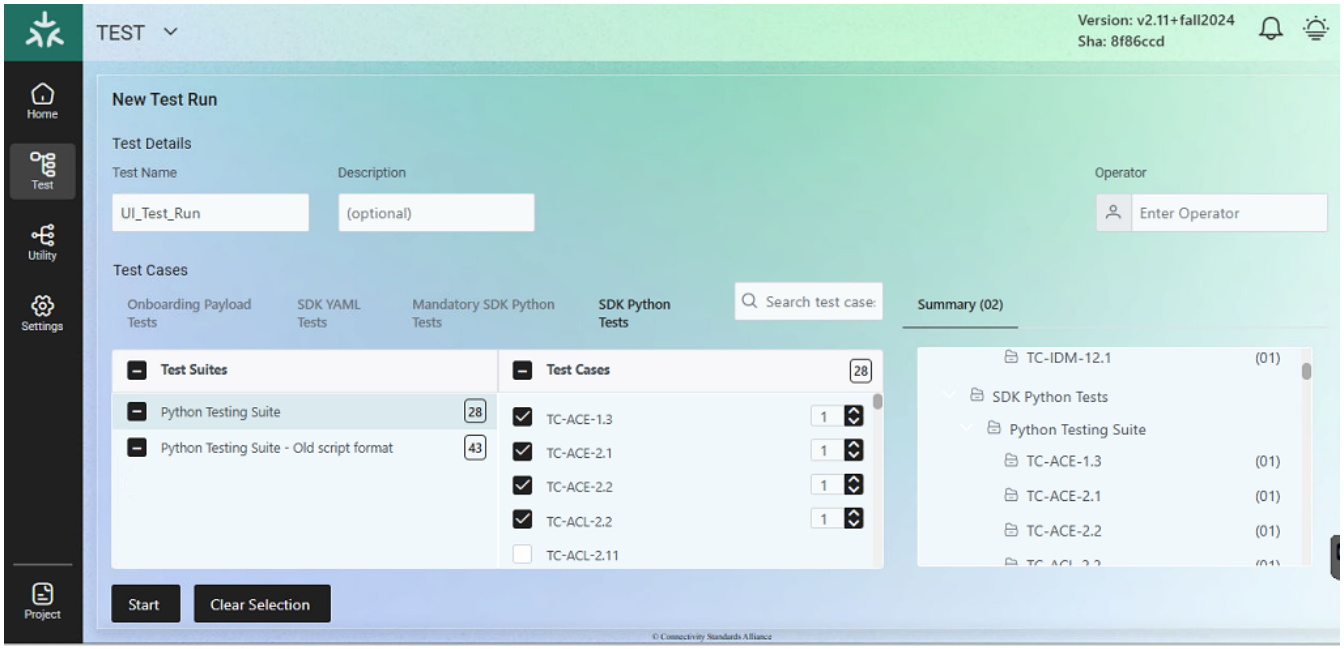

Figure 33. Python tests in TH tool

When the Python test is run, you get a Pass/Fail report, see Figure 34.

Matter Test Harness - Test Run Report

Test Run Information

Test Name	Status	Date	Time	Operator
UI_Test_RunIDM_12_2025_06_27_10_24_25	passed	Jun 27, 2025	10:24:25 - 10:24:25 (1m23s)	alpha

Result Statistics

Passed -1 Error -0 Cancelled -0 Failed -0 pending -0

Result Summary

S.No	Public ID	State	Time Elapsed
1	**Python Testing Suite - Old script format**	Passed	1m23s
1.1	TC_IDM_1_2	Passed	1m21s
1.1.1	Start Python test	Passed	1m5s
1.1.2	Show test logs	Passed	0m0s

Figure 34. Python test report

To run the Python tests inside the docker, do the following steps, you can also see Ref. 6.

1. Run the following commands from the Raspberry Pi terminal:

```
cd certification-tool
./backend/test_collections/matter/scripts/update-paa-certs.sh
```

2. Ensure that the PAA's are available locally at /var/paa-root-certs.
   to run the docker, use the following command:

```
docker run -v $PATH_TO_PAA_ROOTS:/paa_roots -v /var/run/dbus/system_bus_
socket:/var/run/dbus/system_bus_socket -v /home/ubuntu/certification-tool/backend/test_
collections/matter/sdk_tests/sdk_checkout/python_testing:/root/python_testing/ -v
$(pwd):/launch_dir --privileged --network host -it connectedhomeip/chip-cert-bins:<SDK SHA
RECOMMENDED>
```

Python scripts are available in /root/python_testing/scripts/sdk.

```
ubuntu@ubuntu:~/connectedhomeip$
ubuntu@ubuntu:~/connectedhomeip$ docker run -v /var/paa-root-certs:/paa_roots -v /var/run/dbus/system_bus_socket:/var/run/dbus/system_bus_
socket -v /home/ubuntu/certification-tool/backend/test_collections/matter/sdk_tests/sdk_checkout/python_testing:/root/python_testing/ -v $
(pwd):/launch_dir --privileged --network host -it connectedhomeip/chip-cert-bins:f2e5de7_latest
root@ubuntu:~# ls
chip-all-clusters-app               chip-lighting-app                          chip-shell          lit-icd-app
chip-all-clusters-app-nlfaultinject chip-lighting-data-model-no-unique-id-app  chip-tool           matter-network-manager-app
chip-all-clusters-minimal-app       chip-lock-app                              chip-tv-app         python_env
chip-app1                           chip-microwave-oven-app                    chip-tv-casting-app python_lib
chip-bridge-app                     chip-ota-provider-app                      fabric-admin        python_testing
chip-cert                           chip-ota-requestor-app                     fabric-bridge-app   thermostat-app
chip-energy-management-app          chip-rvc-app                               fabric-sync-app
root@ubuntu:~#
root@ubuntu:~# cd /root/python_testing/scripts/sdk/
root@ubuntu:~/python_testing/scripts/sdk# ls
DRLK_PICS                TC_DRLK_2_9.py              TC_OCC_3_1.py          TC_TestEventTrigger.py
EP0                      TC_DRLK_2_9_old.py          TC_OCC_3_2.py          TC_VALCC_2_1.py
EP1                      TC_DeviceBasicComposition.py TC_OPCREDS_3_1.py     TC_VALCC_3_1.py
MinimalRepresentation.py TC_DeviceConformance.py     TC_OPCREDS_3_2.py      TC_VALCC_3_2.py
TCP_Tests.py             TC_ECOINFO_2_1.py           TC_OPSTATE_2_1.py      TC_VALCC_3_3.py
TC_ACE_1_2.py            TC_ECOINFO_2_2.py           TC_OPSTATE_2_2.py      TC_VALCC_3_4.py
```

Figure 35. Python test SDK docker run

3. To run the TC_ACE_1.3.py from the docker, run the following command:

```
python3 TC_ACE_1_3.py --commissioning-method ble-wifi --wifi-ssid RenesasMatter1 --wifi-
passphrase Matter2023 --discriminator 3840 --passcode 20202021 --storage-path admin_
storage.json
```

For every test run using the TH (Automated/Semi-Automated/Python), you can download the reports as well as logs for further analysis of the test cases, see Figure 37 and Figure 38.

```
root@ubuntu:~/python_testing/scripts/sdk# python3 TC_ACE_1_3.py --commissioning-method ble-wifi --wifi-ssid RenesasMatter1 --wifi-passphra
se Matter2023 --discriminator 3840 --passcode 20202021 --storage-path admin_storage.json
[1755695648.390320][22:22] CHIP:CTL: Setting attestation nonce to random value
[1755695648.477737][22:22] CHIP:CTL: Setting CSR nonce to random value
[1755695648.716683][22:22] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_kvs
[1755695648.763206][22:22] CHIP:DL: Wrote settings to /tmp/chip_kvs
[1755695648.785124][22:22] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_factory.ini
[1755695648.790905][22:22] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755695648.791272][22:22] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_config.ini
[1755695648.795857][22:22] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755695648.796219][22:22] CHIP:DL: ChipLinuxStorage::Init: Using KVS config file: /tmp/chip_counters.ini
[1755695648.801347][22:22] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755695648.870380][22:22] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755695648.870530][22:22] CHIP:DL: NVS set: chip-factory/unique-id = "8346C6F99767624D"
[1755695648.876472][22:22] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755695648.876619][22:22] CHIP:DL: NVS set: chip-factory/vendor-id = 65521 (0xFFF1)
[1755695648.885421][22:22] CHIP:DL: Wrote settings to /tmp/chip_factory.ini
[1755695648.885556][22:22] CHIP:DL: NVS set: chip-factory/product-id = 32769 (0x8001)
[1755695648.890916][22:22] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755695648.891046][22:22] CHIP:DL: NVS set: chip-counters/reboot-count = 1 (0x1)
[1755695648.898091][22:22] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755695648.898219][22:22] CHIP:DL: NVS set: chip-counters/total-operational-hours = 0 (0x0)
[1755695648.911643][22:22] CHIP:DL: Wrote settings to /tmp/chip_counters.ini
[1755695648.911780][22:22] CHIP:DL: NVS set: chip-counters/boot-reason = 0 (0x0)
[1755695648.918496][22:22] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755695648.918640][22:22] CHIP:DL: NVS set: chip-config/regulatory-location = 0 (0x0)
[1755695648.927549][22:22] CHIP:DL: Wrote settings to /tmp/chip_config.ini
[1755695648.927695][22:22] CHIP:DL: NVS set: chip-config/location-capability = 2 (0x2)
[1755695648.929428][22:22] CHIP:DL: Got Ethernet interface: eth0
[1755695648.930783][22:22] CHIP:DL: Found the primary Ethernet interface:eth0
[1755695648.932240][22:22] CHIP:DL: Got WiFi interface: wlan0
[1755695648.939487][22:22] CHIP:DL: Found the primary WiFi interface:wlan0
```

Figure 36. Python docker test example

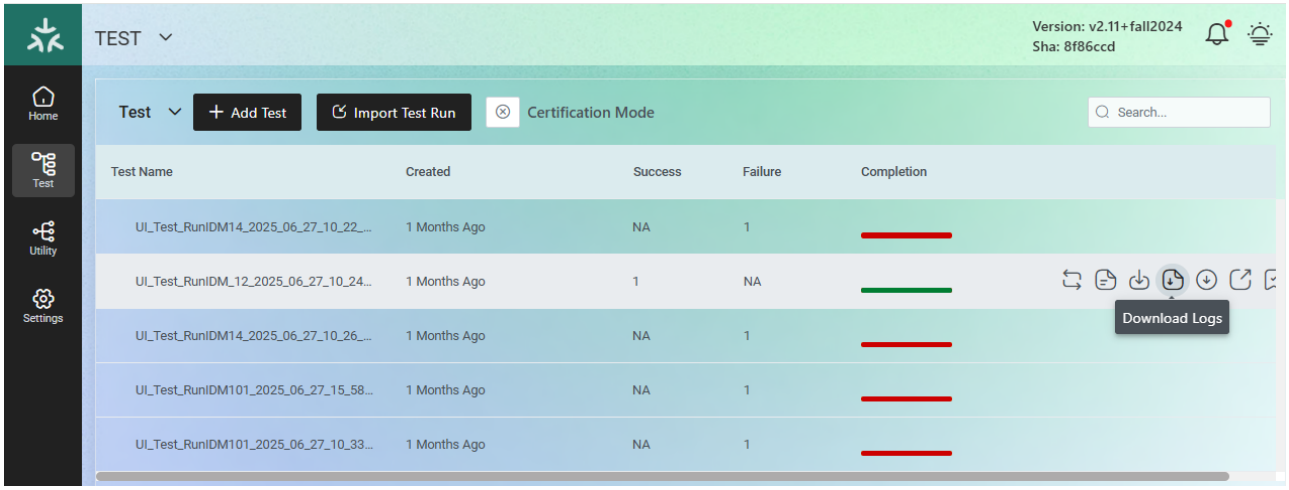

Figure 37. TH Test logs

Every log has a detailed test step (Figure 39), the current configuration derived from PICS, commissioning details, step-by-step verification of the test cases as well as the final test result which helps you to analyze log and debug in case of failures, see Figure 40.

```
INFO       | 2025-06-27 10:24:25.243943 | Run Test Runner is Ready
INFO       | 2025-06-27 10:24:25.245337 | TH Version: v2.11+fall2024
INFO       | 2025-06-27 10:24:25.246432 | TH SHA: 8f86ccd
INFO       | 2025-06-27 10:24:25.247330 | TH SDK SHA: f2e5de7
INFO       | 2025-06-27 10:24:25.256345 | Test Run Executing
INFO       | 2025-06-27 10:24:25.265679 | ################################################################################
INFO       | 2025-06-27 10:24:25.267090 | Test Suite Executing: Python Testing Suite - Old script format
INFO       | 2025-06-27 10:24:25.268233 | Suite Setup
INFO       | 2025-06-27 10:24:25.269225 | Python Test Version: f2e5de7e7978dd0748ba014f7b6ce593dad2fc88
INFO       | 2025-06-27 10:24:25.270186 | Setting up SDK container
INFO       | 2025-06-27 10:24:26.524020 | th-sdk container started with configuration: {'privileged': True, 'detach': True, 'network': 'host', 'name': 'th-
sdk', 'command': 'tail -f /dev/null', 'volumes': {'/var/run/dbus/system_bus_socket': {'bind': '/var/run/dbus/system_bus_socket', 'mode': 'rw'},
PosixPath('/var/tmp'): {'bind': '/logs', 'mode': 'rw'}, PosixPath('/var/paa-root-certs'): {'bind': '/paa-root-certs', 'mode': 'ro'},
PosixPath('/var/credentials/development'): {'bind': '/credentials/development', 'mode': 'ro'}, PosixPath('/home/ubuntu/certification-
tool/backend/test_collections/matter/sdk_tests/sdk_checkout/python_testing'): {'bind': '/root/python_testing', 'mode': 'rw'}, 'mapped_data_model_volume':
{'bind': '/root/python_testing/data_model', 'mode': 'rw'}, PosixPath('/home/ubuntu/certification-
tool/backend/test_collections/matter/sdk_tests/support/python_testing/models/rpc_client/test_harness_client.py'): {'bind':
'/root/python_testing/scripts/sdk/test_harness_client.py', 'mode': 'rw'}}}
INFO       | 2025-06-27 10:24:26.553239 | Create PICS file for DUT
INFO       | 2025-06-27 10:24:26.555946 | Sending command to SDK container: /bin/sh -c "echo 'ACL.S=1
ACL.C=0
ACL.S.A0000=1
ACL.S.A0001=0
ACL.S.A0002=1
ACL.S.A0003=1
ACL.S.A0004=1
ACL.S.A0005=0
ACL.S.A0006=0
ACL.S.E00=1
ACL.S.E01=1
ACL.S.E02=0
ACL.S.C01.Tx=0
ACL.S.C00.Rsp=0
ACL.S.F00=0
ACL.S.F01=0
```

Figure 38. TH log snippet

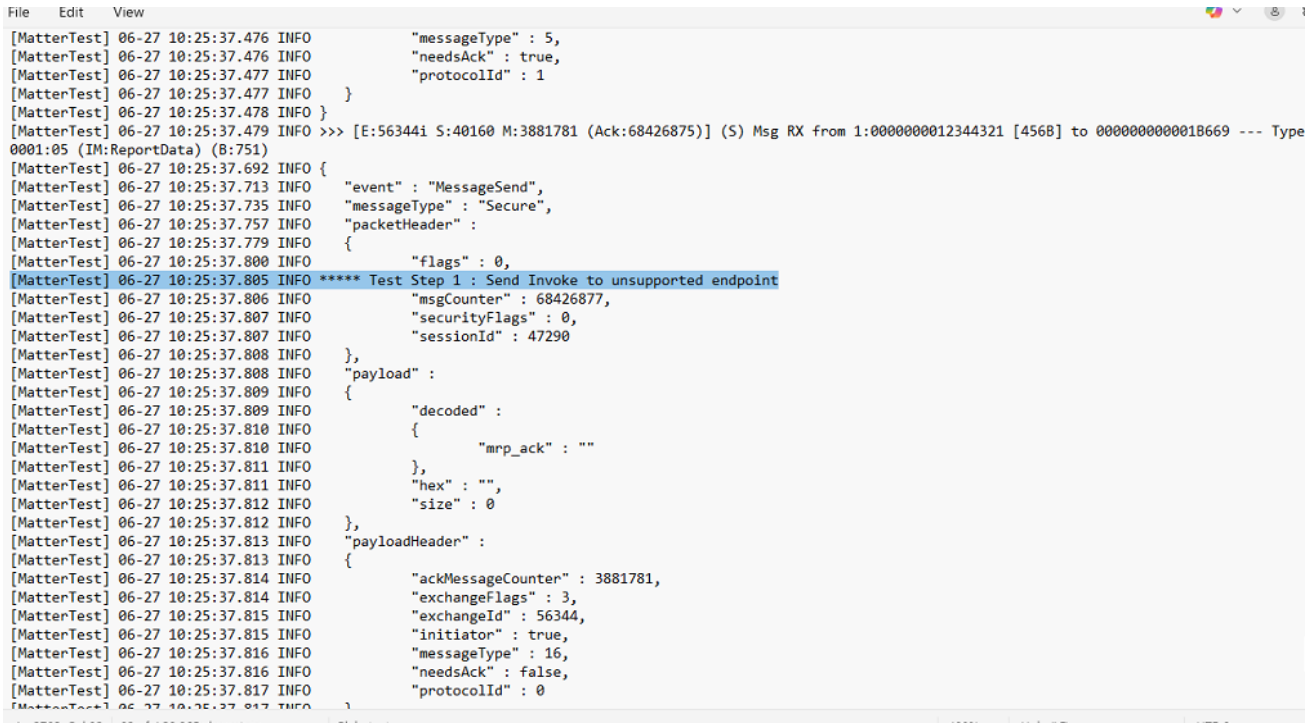

Figure 39. Detailed test steps

```
INFO     | 2025-06-27 10:25:47.288572 | ---- End of Python test logs ----
INFO     | 2025-06-27 10:25:47.292149 | Test Step Completed [PASSED]: Show test logs
INFO     | 2025-06-27 10:25:47.294268 | --------------------------------------------------------------------------------
INFO     | 2025-06-27 10:25:48.405391 | Test Cleanup
INFO     | 2025-06-27 10:25:48.407389 | Test Case Completed [PASSED]: TC-IDM-1.2
INFO     | 2025-06-27 10:25:48.408469 | ================================================================================
INFO     | 2025-06-27 10:25:48.412150 | Suite Cleanup
INFO     | 2025-06-27 10:25:48.413882 | Stopping SDK container
INFO     | 2025-06-27 10:25:48.851778 | Stopping Border Router
INFO     | 2025-06-27 10:25:48.854853 | Test Suite Completed [PASSED]: Python Testing Suite - Old script format
INFO     | 2025-06-27 10:25:48.856089 | ################################################################################
INFO     | 2025-06-27 10:25:48.860783 | Test Run Completed [PASSED]
```

Figure 40. Final test result

Appendix A Matter Certification FAQ

A.1 Typical Issues and Resolutions

During Matter certification testing, devices often encounter a range of common issues. Understanding these problems and their typical resolutions can significantly streamline your development and pre-certification efforts.

- Commissioning Failures: one of the most frequent hurdles is failing to commission the DUT on the Matter network. This might be due to multiple reasons:
  - Incorrect Setup Payload/Discriminator/PIN Code: the commissioning information (QR code, manual pairing code) has to match the values compiled into your DUT's firmware and that these are correctly entered into the Test Harness or chip-tool.
  - Bluetooth LE Advertising Issues: verify the DUT is actively advertising Matter commissioning packets. Check DUT logs for Bluetooth LE errors. The host running the Test Harness must have a functional and correctly configured Bluetooth LE adapter.
  - Network Connectivity Problems: for Wi-Fi DUTs, ensure the DUT can connect to the specified Wi-Fi network and that the Test Harness is on the same network, and that IPv6 is leased from the network.
  - Certificate Mismatch/Invalidity: DAC, PAI, and PAA certificates on the DUT must be valid and correctly provisioned. The Test Harness's PAA trust store must contain the PAA certificate that signed your DUT's PAI. Invalid or expired certificates prevent attestation and commissioning.
  - Factory Reset Needed: a DUT that is previously commissioned to another fabric or is in an unknown state might need a factory reset to enter a commissionable state again.
  - Before running the test cases ensure clearing the temporary chip files from the TH tool using the following command:
    ```
    rm -rf /tmp/chip*
    ```
- Attribute Read/Write failures: tests often fail when the Test Harness tries to read or write attributes on the DUT due to the following:
  - PICS mismatch: the most common cause might be due to the attribute is declared as supported in your PICS file, but it is not actually implemented in the firmware, or its ID is incorrect.
  - Incorrect ZAP configuration: check your ZAP configuration to ensure the attribute is enabled, has the correct data type, and the appropriate storage type (NVM, RAM, or External).
  - Data Type mismatch: the value written or read does not match the attribute's defined data type in the Matter specification or ZAP configuration.
  - Read-only/Write-only misconfiguration: Attempting to write to a read-only attribute or read from a write-only one. Ensure permissions are correctly set in ZAP and implemented in firmware.
- Command execution failures: tests fail due to failure in command execution; DUT responds with error status or DUT does not respond to the commands from TH:
  - PICS mismatch: the command is declared as supported in PICS but not implemented in firmware.
  - Incorrect ZAP Configuration: the command might not be enabled in ZAP for the specified cluster and endpoint, or its ID is incorrect.
  - Prerequisite Not Met: some commands have prerequisites (a lock must be in a certain state for a specific command to work). The test setup needs to meet these conditions.
- Event reporting issues: problems arise when DUT fails to generate expected events and the test harness eventually times out waiting for an event:
  - PICS mismatch: the event is declared as supported but not implemented in firmware.
  - Subscription Issues: the Test Harness is not successfully subscribed to the event, or the DUT is not correctly handling event subscriptions.
  - For the test case TC-DRLK-2.10, the DUT should have the provision to trigger Alarm event and DoorStateChange event. To trigger these events from RA6W2 (when built with door lock cluster) use the following command:
    ```
    matter trigger_alarm [Alarm Code]
    matter trigger_door_state [Door State]
    ```

A.2 General Debugging Tips

While running the test cases, you might encounter different issues or errors. To ensure smooth certification process you need to understand how to debug the various issues. Consider the following general points while debugging an

issue:

- Enable verbose logging: increases logging levels on both your DUT and the Test Harness. Detailed logs are invaluable for pinpointing the exact point of failure.
- Use chip-tool: the chip-tool is an excellent debugging companion. You can use it to manually send commands, read attributes, and subscribe to events, replicating Test Harness steps to isolate issues.
- Review Matter Specification: see the official Matter Core Specification and the relevant Cluster Specifications.

A.3 OTA Implementation

This verifies whether the **DUT** supports OTA software update using a vendor-specific OTA Provider.

A.3.1 Configure Server and Install XAMPP

To install XAMPP (on Windows):

1. Open your browser and go to: https://www.apachefriends.org/index.html.
2. Click **Download** > XAMPP for Windows.
3. After installation is completed, click **Finish** and the **XAMPP Control Panel** opens.
4. In the **XAMPP Control Panel** dialog, click **Start** next to Apache.

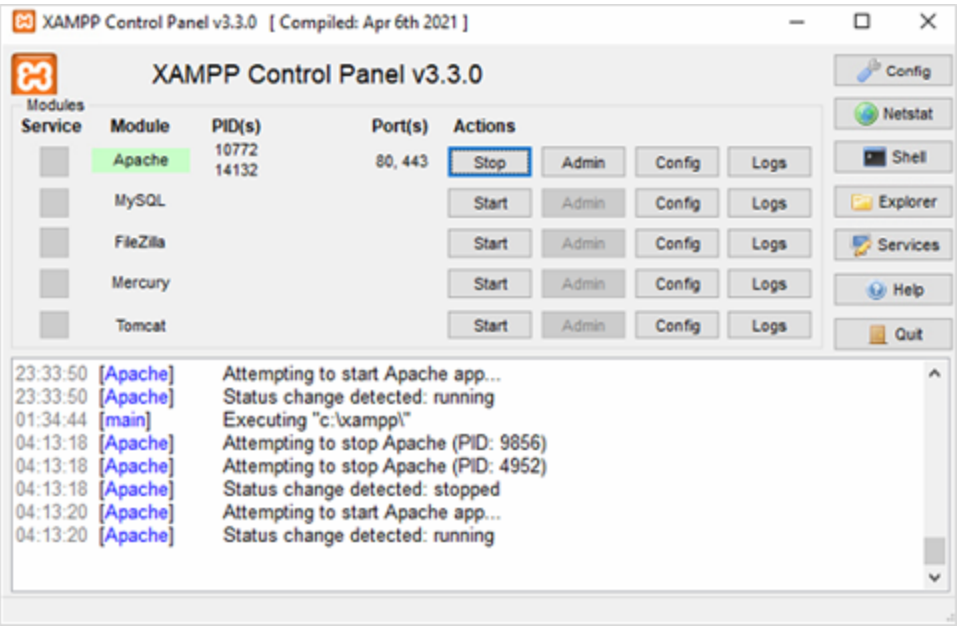

Figure 41. XAMPP Control Panel

5. Connect local Apache server and RA6W2 to the same network.
6. Put the download_ota.img in C:\xampp\htdocs.
7. Download the .img locally by opening the link from the browser.

A.3.2 OTA Test Case: [TC-SU-5.1] Verify Vendor Specific OTA Implementation

To run the OTA test case:

1. Open a terminal in TH tool and commission DUT over Bluetooth LE-Wi-Fi:

```
./chip-tool pairing ble-wifi <node-id> <ssid> <password> <setup-passcode> <discriminator> --
paa-trust-store-path <path-to-paa>
```

2. Download the download_ota.img from the server:

```
ota_update rtos https://192.168.X.XX/download_ota.img
```

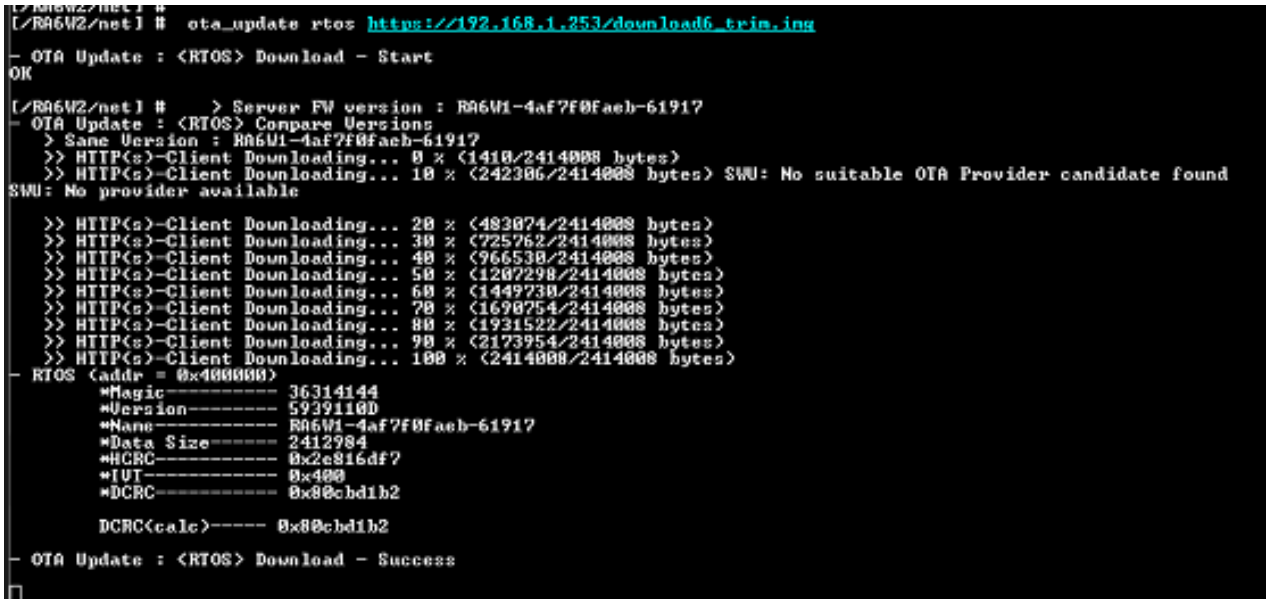

Figure 42. OTA Update on RA6W2

3. To renew or trigger an OTA update in a Matter OTA setup:

```
ota_update renew
```

Figure 43. OTA Update success

After the OTA-transfer is finished successfully, the Boot index becomes 1.

4. Verify the software version is updated by running the following command in the TH tool:

```
./chip-tool basicinformation read software-version 1 0
[1653636406.637617][11116:11121] CHIP:TOO: Endpoint: 0 Cluster: 0x0000_0028 Attribute 0x0000_
000A DataVersion: 1527020963
[1653636406.637708][11116:11121] CHIP:TOO: SoftwareVersion: 2
```

A.4 Recommended Test Houses

For official Matter certification, you must submit your product to an Authorized Test Laboratory (ATL). These labs are accredited by the CSA) to perform the comprehensive testing required to ensure your device complies with Matter specification. Choosing the right ATL is an important decision that can impact the efficiency and cost of your certification journey.

The approximate time taken for Matter certification testing can vary significantly based on several factors, including the complexity of your device, the completeness of your pre-testing and the number of device types and features being certified. While precise timelines should always be confirmed directly with the chosen ATL, a typical certification process at an ATL can range from a few days to several weeks for the actual testing phase, not including the time for document review and official certification approval by the CSA. Thorough pre-testing is the best way to minimize the time spent at the ATL.

The CSA maintains an official and up-to-date list of all ATLs for Matter certification. It is crucial to see this list to select an accredited and current partner for your product's testing. You can find this list on the official CSA website.
For our certification tests we worked with **Allion Labs** due to their strong focus on interoperability and technical expertise in the Matter ecosystem.

10. Revision History

Revision	Date	Description
1.02	Mar 11, 2026	Updated Section 5.2 Add Door Lock Cluster in ZAP Tool.
1.01	Dec 1, 2025	Updated Section 1.2 e2 studio Test VID/PID Modification and Certification Regeneration.
1.00	Aug 31, 2025	Initial version.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

CorporateHeadquarters
TOYOSU FORESIA, 3-2-24 Toyosu
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks
Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact Information
For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/