# Silicon Labs ZAP

# ZAP

ZAP is a generic code generation engine and user interface for applications and libraries based on the Zigbee Cluster Library from Zigbee or the Data Model from Matter. The specification is developed by the Connectivity Standards Alliance.

ZAP allows you to perform the following operations:

- Perform SDK-specific customized generation of all global artifacts (constants, types, IDs, and so on) based on the ZCL/Data-Model specification.
- Perform SDK-specific customized generation of all user-selected configuration artifacts (application configuration, endpoint configuration, and so on) based on ZCL/Data-Model specification and customer-provided application configuration.
- Provide UI for the end-user to select a specific application configuration (endpoints, clusters, attributes, commands, and so on).
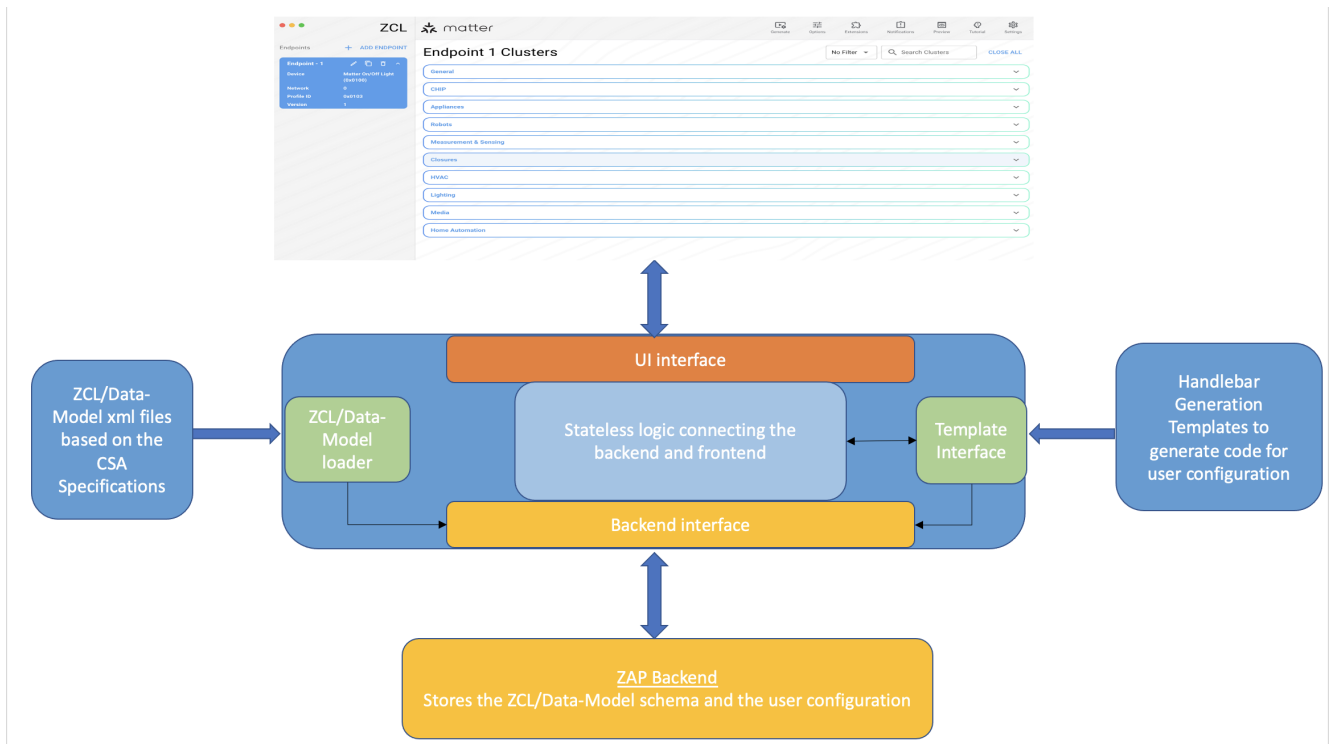


The content in these sections describes how to develop Zigbee and Matter applications by configuring the ZCL (Zigbee) or Data Model (Matter) Layers using ZAP.

ZAP Getting Started

# Getting Started with ZAP

These sections describe different methods to create Zigbee and Matter applications. Note that Simplicity Studio provides a way to create your Zigbee and Matter applications from end to end where all tools come pre-installed along with Simplicity Studio (including ZAP). You may also decide to explore other ways of creating your applications, as described here.

## Zigbee Development

Zigbee application developers can build their applications using Simplicity Studio, which already includes ZAP and other tools that help you build your application from end to end.

## Matter Development

Matter Application developers can build their applications using the following methods:

- Simplicity Studio: This includes ZAP and other tools which are needed to build the Matter application end to end.
- Github (Silicon Labs)
- Github (CSA)

Note: To update ZAP outside the Simplicity Studio release cycle, see update ZAP in Simplicity Studio and ZAP Installation Guide

# ZAP Installation

The following sections describe ZAP installation and how to update ZAP in Simplicity Studio IDE.

## Downloading the ZAP Executable (Recommended)

This is the recommended way of getting started with ZAP. You can get the latest ZAP binaries from https://github.com/project-chip/zap/releases. Prebuilt binaries come in two different versions.

- Official release: Verified builds with dedicated Matter and Zigbee test suites. The release name format is vYYYY.DD.MM.
- Pre-release: Builds with the latest features and bug fixes but these builds are NOT verified with dedicated Matter and Zigbee test suites. The release name format is vYYYY.DD.MM-nightly.

## Installing ZAP from Source

**Basic instructions to Install ZAP**

Because this is a node.js application, you need the node environment installed. The best way to do this is download the latest install of node, which includes node and npm. If you have an older version of node installed on your workstation, it may cause issues, particularly if it's very old. Make sure you have the latest node v16.x version with the npm that's included. Run `node --version` to check which version is picked up. v18.x is recommended.

After you have a desired version of node, you can run the following:

### Install the Dependencies

Use the following commands to install dependencies:

```
npm install
```

Note: For Windows-specific ZAP installation, see ZAP Installation for Windows OS

It is not uncommon to run into native library compilation problems at this point. There are various `src-script/install-*` scripts for different platforms. See FAQ information about which script to run on different platforms and then rerun `npm install`.

### Start the Application

Use the following commands to start up the application:

```
npm run zap
```

### Start the Front-End in Development Mode

Supports hot-code reloading, error reporting, and so on. Use the following commands to start the front-end in development mode:

```
quasar dev -m electron
```

or

```
npm run electron-dev
```

# ZAP Installation for Windows OS

## 1. Windows Powershell

In the desktop search bar, input `Windows Powershell` and run as administrator. Run all the following commands inside Powershell.

## 2. Chocolatey

Install from https://chocolatey.org/install.

Check if installed properly with the following commands:

```
choco -v
```

Install pkgconfiglite package with the following commands:

```
choco install pkgconfiglite
```

## 3. Install Node

Run the following commands to install:

```
choco install nodejs-lts
```

*The version has to be 18 to pass version check test, after install, check with `node -v`

*If you have installed Node already, and fail some tests similar to `cannot find Node`, reinstall Node with chocolatey again.

## 4. Follow the Basic Instructions to Install ZAP

Follow the ZAP installation instructions from source in ZAP Installation. While following the basic instructions for installing ZAP watch out for the following errors and how to resolve them:

### sqlite3

When running ZAP (e.g., `npm run zap`), if you see an error about `sqlite3.node` in a pop up window, run:

```
npm rebuild sqlite3
```

### electron-builder

When doing npm install, in post-install, if an error occurs on the following command related to `electron-builder install-app-deps`, `npx electron-rebuild canvas failed` or `node-pre-gyp`, the current `canvas` version is not compatible with Windows and the installation error will not cause a failure in running ZAP. node-canvas is working on the solution now and the issue will be solved in the near future.

```
"postinstall": "electron-builder install-app-deps && husky install && npm rebuild canvas --update-binary && npm run version-stamp"
```

## Canvas

If `npm run test` fails because of the error `Test suite failed to run. Cannot find module '../build/Release/canvas.node'` or `\zap\node_modules\canvas\build\Release\canvas.node is not a valid Win32 application.` , rebuild canvas as follows:

```
npm rebuild canvas --update-binary
```

## get index.html or Other Server Issues

If `npm run test` fails because of the error `get index.html request failed with status code 404` in unit tests or having server connection issues in e2e-ci tests, run the following commands:

```
npm run build
```

## Other

Check if the node version is v18 and try to install it with Chocolatey.

# Frequently Asked Questions

Q: How to start up UI in a development mode?

A:

You can start the UI in a development mode, which will result in a following setup:

- Separate quasar development HTTP server, which does live refresh on port 8080
- ZAP back end running on port 9070
- Chrome or other browser, running independently

To get to that setup, follow the instructions below.

1. First, run the ZAP development server, which starts on port 9070.

```
npm run zap-devserver
```

1. Next, run the quasar development server, which starts on port 8080.

```
quasar dev
```

1. Point your browser or run one against the proper URL with the `restPort` argument:

```
google-chrome http://localhost:8080/?restPort=9070
```

---

Q: How to make this work on Mac/Linux OS?

A:

`npm install` is used to download all required dependency packages.

If you see errors related to `node-gyp` and missing local libraries, like `pixman`, and so on, you are missing native dependencies to satisfy to compile non-prebuilt node binaries for some combination of platforms and versions. Npm on the cloud is constantly updating the list of provided binaries, so it's possible that you will pick them up just fine, but if you don't, these are instructions for different platforms:

**Fedora Core** with `dnf` :

```
dnf install pixman-devel cairo-devel pango-devel libjpeg-devel giflib-devel
```

or run script:

```
src-script/install-packages-fedora
```

**Ubuntu** with `apt-get` :

```
apt-get update
apt-get install --fix-missing libpixman-1-dev libcairo-dev libsdl-pango-dev libjpeg-dev libgif-dev
```

or run script:

```
src-script/install-packages-ubuntu
```

**OSX on a Mac** with Homebrew `brew` :

```
brew install pkg-config cairo pango libpng jpeg giflib librsvg
```

or run script:

```
src-script/install-packages-osx
```

---

Q: How to make this work on Windows OS?

A:

Make sure it's always up to date and there are no changes that haven't been committed. Tip: git pull, git status & git stash are your friends.

You must use Chocolately to make Zap work on Windows OS. Make sure to download the pkgconfiglite package.

```
choco install pkgconfiglite
```

If you have issues with cairo, for example if you get an error about cairo.h': No such file or directory, do the following:

1. Check if your computer is 32 or 64 bit.
2. Depending on that, download the appropriate package from this site
   https://github.com/benjamind/delarre.docpad/blob/master/src/documents/posts/installing-node-canvas-for-windows.html.md.
3. Create a folder on your C drive called GTK if it doesn't already exist.
4. Unzip the downloaded content into C:/GTK.
5. Copy all the dll files from C:/GTK/bin to your node_modules/canvas/build/Release folder in your zap folder.
6. Add C:/GTK to the path Environment Variable by going to System in the Control Panel and doing the following:

- Click on Advanced System Settings.
- In the advanced tab click on Environment Variables.
- In the section System Variables, find the PATH environment variable and select it.
- Click Edit and add C:/GTK to it.
- If the PATH environment variable does not exist, click New.

If jpeglib.h is not found, try the following:

1. On the terminal, run: choco install libjpeg-turbo
2. Make sure it's clean by using: git clean -dxff and run npm install again
3. if no errors occur and only warnings appear, try to use npm audit fix
4. if you can't run ZAP, go to file src-script/zap-start.js
5. Change
6. `const { spawn } = require('cross-spawn')` to `const { spawn } = require('child_process')`
7. Run npm and run zap.

References:

- https://github.com/fabricjs/fabric.js/issues/3611
- https://github.com/benjamind/delarre.docpad/blob/master/src/documents/posts/installing-node-canvas-for-windows.html.md

- https://chocolatey.org/packages/libjpeg-turbo#dependencies

---

**Q: I get an error "sqlite3_node" not found or similar.**

**A:** Rebuild your native sqlite3 bindings.

To fix this in most cases, run:

- `npm install`
- `./node_modules/.bin/electron-rebuild -w sqlite3 -p`

If it still doesn't get fixed, do:

- `rm -rf node_modules` and then try the above commands again.

Occasionally upgrading your `npm` also makes a difference:

- `npm install -g npm`

---

**Q: I get an error "The N-API version of this Node instance is 1. This module supports N-API version(s) 3. This Node instance cannot run this module."**

**A:** Upgrade your node version.

The solution for this is discussed in this Stack Overflow thread: https://stackoverflow.com/questions/60620327/the-n-api-version-of-this-node-instance-is-1-this-module-supports-n-api-version

---

**Q: My development PC doesn't work with ZAP for whatever reason. Can I use a docker container?**

**A:** Yes you can. TBD.

---

**Q: How do I run ZAP inside VSCode?**

**A:** If you VSCode in your path enter the `zap` repo and type `code .` This will open ZAP in VSCode. To run ZAP in debug mode, select the ZAP workspace and click on the `Run` icon on the left hand toolbar. You will have a couple of options to choose from to run ZAP, choose `Node.js Debug Terminal`. This will open a terminal window from which you can enter `npm run zap`, which will attach the debugger and run ZAP as you would normally from the command line. Congratulations, you should now see ZAP running in the debugger. You can set breakpoints in VSCode as you would in any other IDE.

---

**Q: UI unit test fails with some errors around canvas not build for the right version of node. What do I do?**

**A:** If you see the following error:

```
FAIL  test/ui.test.js
● Test suite failed to run
 The module 'canvas.node'
 was compiled against a different Node.js version using
 NODE_MODULE_VERSION 80. This version of Node.js requires
 NODE_MODULE_VERSION 72. Please try re-compiling or re-installing
 the module (for instance, using `npm rebuild` or `npm install`).
   at Object.<anonymous> (node_modules/canvas/lib/bindings.js:3:18)
```

then run: `npm rebuild canvas --update-binary`

# ZCL/Data-Model (ZAP) Fundamentals

This section contains information for new ZAP users.

- Click on the tutorial icon on the top right corner of the ZAP UI, which shows how to create a ZAP configuration. The tutorial will guide you through the following:
  - Create an endpoint
  - Select a device type
  - Configure a cluster
  - Configure an attribute
  - Configure a command
- For detailed reference, see Zigbee Cluster Configurator Guide

## ZAP User's Guide

# ZAP User's Guide

The sections under this guide provide more details about the different features provided by ZAP.

# Custom XML

## Adding Custom XML from the ZAP UI

- Click on "Extensions" icon in the ZAP UI.
- Click on the "+" add button to select a custom xml file
- The custom clusters, attributes, commands, etc should show up in the ZAP UI once the custom xml has been added.

## Creating your own custom XML in Zigbee

The section shows how to create your own custom clusters and extend existing standard clusters with custom attributes and commands.

**Manufacturer Specific Clusters:**

You can add manufacturer specific clusters to a standard profile. We provide an example of this below. In order to do this you must satisfy two obligations:

- The cluster ID MUST be in the manufacturer specific range, 0xfc00 - 0xffff.
- The cluster definition must include a manufacturer code which will be applied to ALL attributes and commands within that cluster and must be provided when sending and receiving commands and interacting with attributes.
- eg:

```xml
<cluster manufacturerCode="0×1002">
  <name>Sample Mfg Specific Cluster</name>
  <domain>General</domain>
  <description>This cluster provides an example of how the Application
  Framework can be extended to include manufacturer specific clusters.
  </description>
  <code>0xFC00</code>
  <attribute side="server" code="0×0000" define="ATTRIBUTE_ONE" type="INT8U" min="0×00" max="0xFF" writable="true" default="0×00" optional="true">ember sample attribute</attribute>
  <attribute side="server" code="0×0001" define="ATTRIBUTE_TWO" type="INT8U" min="0×00" max="0xFF" writable="true" default="0×00" optional="true">ember sample attribute 2</attribute>
  <command source="client" code="0×00" name="CommandOne" optional="true">
    <description>
    A sample manufacturer specific command within the sample manufacturer specific
    cluster.
    </description>
    <arg name="argOne" type="INT8U"/>
  </command>
</cluster>
```

**Manufacturer Specific Commands in Standard ZigBee Custer:**

You can add your own commands to any standard ZigBee cluster with the following requirements:

- Your manufacturer specific commands may use any command id within the command id range, 0x00 - 0xff.
- You must also provide a manufacturer code for the command so that it can be distinguished from other commands in the cluster and handled appropriately.
- eg(Extending the On/Off cluster with manufacturing commands):

```xml
<clusterExtension code="0×0006">
  <command source="client" code="0×00" name="SampleMfgSpecificOffWithTransition" optional="true" manufacturerCode="0×1002">
   <description>Client command that turns the device off with a transition given
    by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×01" name="SampleMfgSpecificOnWithTransition" optional="true" manufacturerCode="0×1002">
   <description>Client command that turns the device on with a transition given
    by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×02" name="SampleMfgSpecificToggleWithTransition" optional="true" manufacturerCode="0×1002">
   <description>Client command that toggles the device with a transition given
    by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×01" name="SampleMfgSpecificOnWithTransition2" optional="true" manufacturerCode="0×1049">
   <description>Client command that turns the device on with a transition given
    by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×02" name="SampleMfgSpecificToggleWithTransition2" optional="true" manufacturerCode="0×1049">
   <description>Client command that toggles the device with a transition given
    by the transition time in the Ember Sample transition time attribute.</description>
  </command>
</clusterExtension>
```

### Manufacturer Specific Attributes in Standard ZigBee Cluster:

You can add your own attributes to any standard ZigBee cluster with the following requirements:

- Your manufacturer specific attributes may use any attribute id within the attribute id range, 0x0000 - 0xffff.
- You must also provide a manufacturer code for the attribute so that it can be distinguished from other attributes in the cluster and handled appropriately.
- eg(Extending the On/Off cluster with manufacturing attributes):

```xml
<clusterExtension code="0×0006">
  <attribute side="server" code="0×0000" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME" type="INT16U" min="0×0000" max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1002">Sample Mfg Specific Attribute: 0×0000 0×1002</attribute>
  <attribute side="server" code="0×0000" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_2" type="INT8U" min="0×0000" max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1049">Sample Mfg Specific Attribute: 0×0000 0×1049</attribute>
  <attribute side="server" code="0×0001" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_3" type="INT8U" min="0×0000" max="0xFFFF" writable="true" default="0×00" optional="true" manufacturerCode="0×1002">Sample Mfg Specific Attribute: 0×0001 0×1002</attribute>
  <attribute side="server" code="0×0001" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_4" type="INT16U" min="0×0000" max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1049">Sample Mfg Specific Attribute: 0×0001 0×1040</attribute>
</clusterExtension>
```

# Custom XML Tags for Zigbee

The following document talks about each of the xml tags associated with Zigbee.

- Each xml file is listed between the `configurator` tags:

```
<configurator></configurator>
```

- Data types can be defined within the `configurator` tag. Zigbee currently supports the definition of bitmaps, enums, integers, strings or structs. Before defining more types make sure to check all the existing atomic types defined in types.xml and all the non-atomic types defined in the other xml files. You may define them as follows:
  - Bitmap:
    - name: name of bitmap type.
    - type: Bitmap with a size between 8-64 bits can be defined, all of which should be multiples of 8.
    - Each bitmap can have multiple fields with a name and a mask associated with it.
    - eg:

```
<bitmap name="bitmapName" type="BITMAP8">
    <field name="field1" mask="0×0F"/>
    <field name="field2" mask="0xF0"/>
</bitmap>```
```

  - Enum:
    - name: name of enum type.
    - type: Enum with a size between 8-64 bits can be defined, all of which should be multiples of 8.
    - Each enum can have multiple items with a name and a value associated with it.
    - eg:

```
<enum name="enumName" type="ENUM8">
    <item name="enumItem1" value="0×00"/>
    <item name="enumItem2" value="0×10"/>
    <item name="enumItem3" value="0×20"/>
</enum>```
```

  - Integer:
    - Integer types are already defined under atomic types which exist in types.xml. Their size can range from 8-64 bits and can be signed or unsigned.
    - eg:

```
<type id="0×28" name="int8s" size="1" description="Signed 8-bit integer" signed="true"/>
```

  - String:
    - String types are already defined under atomic types which exist in types.xml. Current string types include octet string, char string, long octet string and long char string
    - eg:

```
<type id="0×44" name="long_char_string" description="Long character string" discrete="true" string="true" char="true" long="true"/>
```

  - Struct:
    - name: name of struct type.
    - Each struct can have multiple items with a name and a type associated with it. The type can be any predefined types under data types.
    - eg:

```
<struct name="structname">
  <item name="structItem1" type="INT8U"/>
  <item name="structItem2" type="[Any defined type name in the xml files]"/>
</struct>
```

- Custom Clusters can be defined within the `configurator` tag.
  - name: name of the cluster
  - domain: domain of the cluster. The cluster will show up in the ZAP UI under this domain.
  - description: Descirption of the cluster
  - code: cluster code
  - define: cluster define which is used by code generator to define the cluster in a certain way
  - manufacturerCode: Used to define a manufacturing specific cluster. This has to be between 0xfc00 - 0xffff. The manufacturer code for the cluster needs to be defined as follows:

```
<cluster manufacturerCode="0x1002">
```

  - A manufacturing cluster automatically makes the attributes and commands under it of the same manufacturer code unless they explicitly list the manufacturer code.
  - introducedIn: Used to determine the spec version in which the cluster was introduced. This is used by code generator to add additional logic.
  - removedIn: Used to determine the spec version in which the cluster was removed. This is used by code generator to add additional logic.
  - singleton(boolean): Is used to determine a cluster as a singleton such that there is only one instance of that cluster shared across the endpoints.
  - attribute:
    - defines an attribute for the cluster
    - name: Name of attribute is mentioned between the attribute tag.

```
<attribute>attribute name</attribute>
```

    - side(client/server): The side of the cluster to which the attribute is associated too.
    - code: attribute code
    - manufacturer code: This can be used to define a manufacturer specific attribute outside the zigbee specification mentioned by the standard xml.
    - define: attribute define which is used by code generator to define an attribute in a certain way
    - type: the type of the attribute which can be any of the data types mentioned in the xml
    - default: default value for the attribute.
    - min: Minimum allowed value for an attribute
    - max: Maximum allowed value for an attribute
    - writable: Is attribute value writable or not. This can be used to prevent the attribute from being modified by write commands.
    - optional(boolean): Used to determine if an attribute is optional or not for the cluster.
    - min: Minimum allowed value for an attribute when it is an integer, enum or bitmap type.
    - max: Maximium allowed value for the attribute when it is an integer, enum or bitmap type
    - length: Used to specify the maximum length of the attribute when it is of type string.
    - minLength: Used to specify the minimum length of the attribute when it is of type string.
    - reportable(boolean): Tells if an attribute is reportable or not
    - isNullable(boolean): Allows null values for the attribute.
    - array(boolean): Used to declare an attribute of type array.
    - introducedIn: Used to determine the spec version in which the attribute was introduced. This is used by code generator to add additional logic.
    - removedIn: Used to determine the spec version in which the attribute was removed. This is used by code generator to add additional logic.
  - command:
    - define a command for a cluster
    - name: Name of command.

```
<command name="commandName"></command>
```

    - code: command code

- manufacturer code: This can be used to define a manufacturer specific command outside the zigbee specification mentioned by the standard xml.
- description: description of the command
- source(client/server): source of the command.
- optional(boolean): Used to determine if a command is optional or not for the cluster.
- introducedIn: Used to determine the spec version in which the command was introduced. This is used by code generator to add additional logic.
- removedIn: Used to determine the spec version in which the command was removed. This is used by code generator to add additional logic.
- command arguments:
  - Each command can have a set of command arguments
  - name: name of the command argument
  - type: type of the command argument which could be any of the types mentioned in the xml.
  - min: Minimum allowed value for an argument when it is an integer, enum or bitmap type.
  - max: Maximium allowed value for an argument when it is an integer, enum or bitmap type
  - length: Used to specify the maximum allowable length for a command argument when it is of type string.
  - minLength: Used to specify the minimum allowable length for a command argument when it is of type string.
  - array(boolean): To determine if the command argument is of type array.
  - presentIf(string): This can be a conditional string of logical operations based on other command arguments where you can expect the command argument if the conditional string evaluates to true. eg:

    ```
    <arg name="transitionTime" type="INT16U" presentIf="status==0"/>
    ```

    Note: Here `status` is another command argument name.
  - optional(boolean): Used to determine the command argument as optional.
  - countArg: Used when the command argument is of type array. This is used to mention the other command argument which denotes the size of array for this argument.

    ```
    <arg name="numberOfAg" type="INT8U"/>
    <arg name="zoneIds" type="INT8U" array="true" countArg="numberOfZones"/>
    ```

  - introducedIn: Used to determine the spec version in which the command argument was introduced. This is used by code generator to add additional logic.
  - removedIn: Used to determine the spec version in which the command argument was removed. This is used by code generator to add additional logic.
- Cluster Extension can be defined within the `configurator` tag.
  - Cluster extension is used to extend a standard cluster with manufacturing attributes and commands
  - eg

```xml
<clusterExtension code="0×0006">
  <attribute side="server" code="0×0000" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME" type="INT16U" min="0×0000"
max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1002">Sample Mfg Specific Attribute: 0×0000
0×1002</attribute>
  <attribute side="server" code="0×0000" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_2" type="INT8U" min="0×0000"
max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1049">Sample Mfg Specific Attribute: 0×0000
0×1049</attribute>
  <attribute side="server" code="0×0001" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_3" type="INT8U" min="0×0000"
max="0xFFFF" writable="true" default="0×00" optional="true" manufacturerCode="0×1002">Sample Mfg Specific Attribute: 0×0001
0×1002</attribute>
  <attribute side="server" code="0×0001" define="SAMPLE_MFG_SPECIFIC_TRANSITION_TIME_4" type="INT16U" min="0×0000"
max="0xFFFF" writable="true" default="0×0000" optional="true" manufacturerCode="0×1049">Sample Mfg Specific Attribute: 0×0001
0×1040</attribute>
  <command source="client" code="0×00" name="SampleMfgSpecificOffWithTransition" optional="true" manufacturerCode="0×1002">
    <description>Client command that turns the device off with a transition given
     by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×01" name="SampleMfgSpecificOnWithTransition" optional="true" manufacturerCode="0×1002">
    <description>Client command that turns the device on with a transition given
     by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×02" name="SampleMfgSpecificToggleWithTransition" optional="true" manufacturerCode="0×1002">
    <description>Client command that toggles the device with a transition given
     by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×01" name="SampleMfgSpecificOnWithTransition2" optional="true" manufacturerCode="0×1049">
    <description>Client command that turns the device on with a transition given
     by the transition time in the Ember Sample transition time attribute.</description>
  </command>
  <command source="client" code="0×02" name="SampleMfgSpecificToggleWithTransition2" optional="true" manufacturerCode="0×1049">
    <description>Client command that toggles the device with a transition given
     by the transition time in the Ember Sample transition time attribute.</description>
  </command>
</clusterExtension>
```

# Multiple Device Types Per Endpoint

This is a Matter-only feature where a user can select more than one device type per endpoint. The addition of multiple device types will add the cluster configurations within the device types to the endpoint configuration.

# Create New Endpoint

Endpoint
1

Profile ID
0x0103

Device
Matter On/Off Light (0x0100) ⊗    Matter Dimmable Light (0x0101) ⊗
|

Primary Device
Matter On/Off Light (0x0100)

| Device | Version |
| --- | --- |
| Matter On/Off Light (0x0100) | 1 |
| Matter Dimmable Light (0x0101) | 1 |

Network
0

CANCEL         CREATE

The above image shows that endpoint 1 has more than one device types selected. The "Primary Device" denotes primary device type that the endpoint will be associated with. The primary device type is always present at index 0 of the list of device types selected so selecting a different primary device type will change the ordering of the device types selected. The device type selections also have constraints based on the Data Model Specification. ZAP protects the users from choosing invalid combinations of device types on an endpoint using these constraints.

# Notifications

The following section defines how notifications are given to ZAP users in the UI.

## Package Notifications

Package notifications are the warnings or error messages associated for any specific package loaded into ZAP. For example, in the images below, clicking the warning icon under the status column will lead you to a dialog showing all notifications for that package.

## Session Notifications

Session notifications are the warnings or error messages which are associated with a user session. These warnings/errors can be seen by clicking on the `Notifications` button in the toolbar on top of the ZAP UI.

For example, the image below shows the session notifications page after an isc file was loaded into ZAP.

**SILICON LABS**
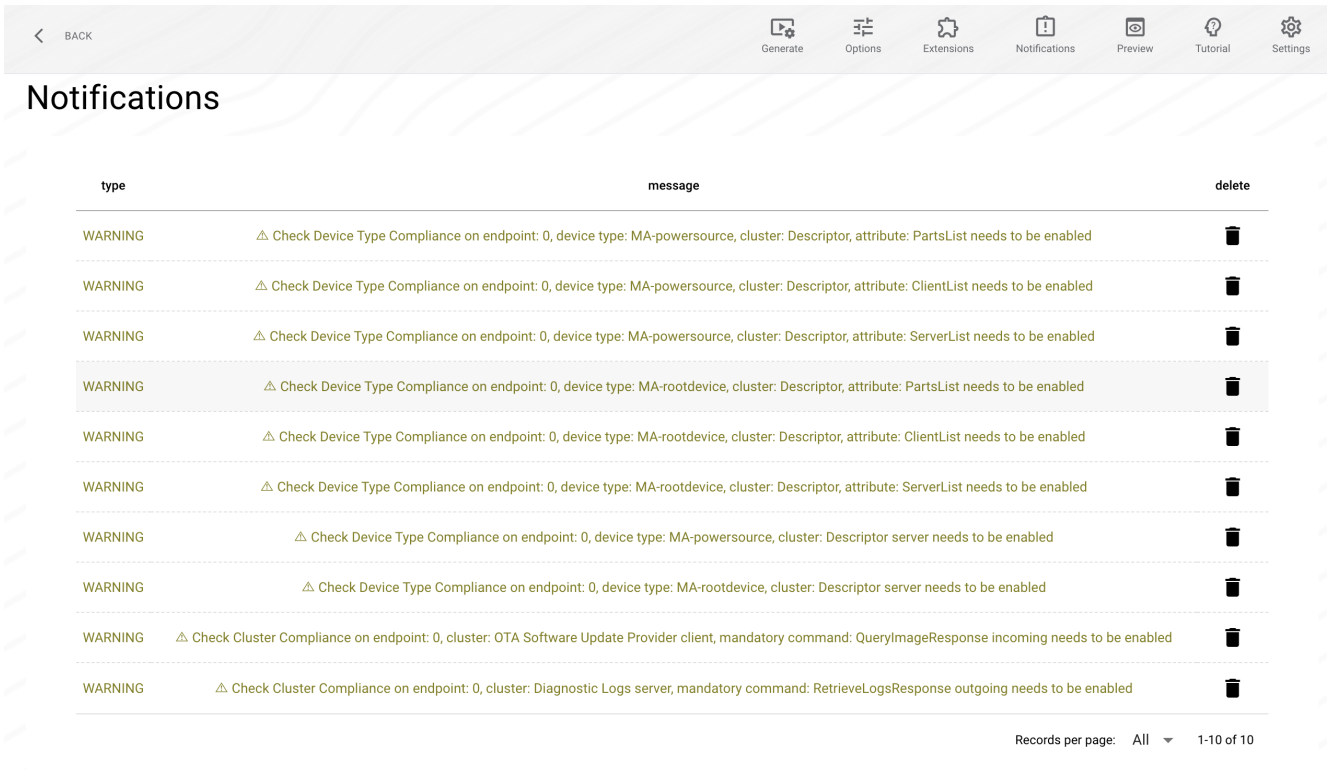
# Data Model and ZCL Specification Compliance

This feature in ZAP helps users see compliance failures for Data Model or ZCL with their existing ZAP configurations. The warning messages for compliance failures will appear on the Notifications pane in the ZAP UI and will also be logged onto the console when running ZAP through the CLI. The compliance feature currently provides warnings for device type compliance and cluster compliance on an endpoint.

## Compliance Warnings in the ZAP UI

When a user opens a .zap file using the ZAP UI they will see warnings in the notifications pane of the ZAP UI for all the compliance failures. For example, the image below shows the session notifications page after a .zap file was opened with compliance issues.



The compliance messages will go away once the issues are resolved using the ZAP UI such that you can keep track of only the remaining compliance issues. New warnings will also show up for compliance if user disables mandatory elements(cluster/commands/attributes) of the configuration. Specification compliance notifications will always keep track of any failures that are introduced into the ZAP configuration but note that the warnings which show up during the opening of a .zap file are more elaborate on why it failed compliance when compared to the warnings which show up while interacting with the UI. This is by design and a full compliance check is performed during the opening of a .zap file.

## Compliance Warnings on the Console

When a user opens a .zap file using the ZAP standalone UI or the ZAP CLI they will see warnings logged into the console/terminal for all the compliance failures. For example, the image below shows the session notification warnings on the console/terminal after a .zap file was opened with compliance issues.

```
Application is failing the Device Type Specification as follows:

 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-rootdevice, cluster: Descriptor server needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-powersource, cluster: Descriptor server needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-rootdevice, cluster: Descriptor, attribute: ServerList needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-rootdevice, cluster: Descriptor, attribute: ClientList needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-rootdevice, cluster: Descriptor, attribute: PartsList needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-powersource, cluster: Descriptor, attribute: ServerList needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-powersource, cluster: Descriptor, attribute: ClientList needs to be enabled
 - ⚠ Check Device Type Compliance on endpoint: 0, device type: MA-powersource, cluster: Descriptor, attribute: PartsList needs to be enabled

Application is failing the Cluster Specification as follows:

 - ⚠ Check Cluster Compliance on endpoint: 0, cluster: OTA Software Update Provider client, mandatory command: QueryImageResponse incoming needs to be enabled
 - ⚠ Check Cluster Compliance on endpoint: 0, cluster: Diagnostic Logs server, mandatory command: RetrieveLogsResponse outgoing needs to be enabled
```

## Access Control

# Access Control Features

ZAP supports access control on all ZCL entities. It's down to the implementation of the SDK to map these features to the required and supported access control SDK features. ZAP generally provides a data model and a mechanism to encode it in the meta-info files and propagate that data to the generation templates, without assigning specific meanings to the data points.

## Base Terms

ZAP access control defines three base terms, as follows:

1. **operation** : defined as something that can be done. Example: read, write, invoke.
2. **role**: defined as a privilege of an actor. Such as "View privilege", "Administrative role", and son on.
3. **modifiers**: defined as special access control conditions, such as *fabric sensitive* data or *fabric scoped* data.

The base terms are defined in the metadata XML under a top tag `<accessControl>` . The following is an example of access control base term definitions:

```xml
<accessControl>
 <operation type="read" description="Read operation"/>
 <operation type="write" description="Write operation"/>
 <operation type="invoke" description="Invoke operation"/>
 <modifier type="fabric-scoped" description="Fabric-scoped data"/>
 <modifier type="fabric-sensitive" description="Fabric-sensitive data"/>
 <role type="view" description="View privilege"/>
 <role type="operate" description="Operate privilege"/>
 <role type="manage" description="Managing privilege"/>
 <role type="administer" description="Administrative privilege"/>
</accessControl>
```

This example defines three operations, *read*, *write* and *invoke*, two modifiers and four roles.

## Access Triplets

Each individual access condition can be defined with an access triplet in the XML. Access triplet is a combination of an *operation*, *role* and *modifier*. They are optional, so you can only have one of these. A missing part of triplet generally means permissivenes, which is implementation-specific for the given SDK. An entity that defines it's access can have one or more access triplets.

The following is an example:

```xml
<attribute side="server" code="0×0000" define="AT1" type="INT64U" writable="false" optional="true">
 <description>at1</description>
 <access op="write" role="manage" modifier="fabric-scoped"/>
</attribute>
```

This is a definition of an attribute that has an access triplet, declaring it allows *write* operation by a *manage* role, with *fabric-scoped* modifier applied.

## Default Permissions

ZCL entities can define their own individual permissions. However, there is also a global definition of default permissions for given types. These are assumed for the given entity, unless it provides any specific permissions of its own.

Default permissions are declared via a `<defaultAccess>` tag at the top level of the XML file.

Example:

```xml
<defaultAccess type="command">
  <access op="invoke"/>
</defaultAccess>
<defaultAccess type="cluster">
  <access op="read"/>
  <access op="write"/>
</defaultAccess>
<defaultAccess type="attribute">
  <access op="read" role="view"/>
  <access op="write" role="operate"/>
</defaultAccess>
```

# Template Helpers

The basic template helper to use is the `{{#access}} ... {{/access}}` iterator. This iterator iterates over all given access triplets. It supports the following two options:

- `entity="attribute/command/event"` - if the entity can't be determined from context, this sets the entity type.
- `includeDefault="true/false"` - determines if default values are included or not.

The following is an example:

```
{{#zcl_clusters}}
Cluster: {{name}} [{{code}}]
{{#zcl_attributes}}
  - attribute: {{name}} [{{code}}]
  {{#access entity="attribute"}}
    * Op: {{operation}} / Role: {{role}} / Modifier: {{accessModifier}}
  {{/access}}
{{/zcl_attributes}}
{{#zcl_commands}}
  - command: {{name}} [{{code}}]
  {{#access entity="command"}}
    * Op: {{operation}} / Role: {{role}} / Modifier: {{accessModifier}}
  {{/access}}
{{/zcl_commands}}
{{#zcl_events}}
  - event: {{name}} [{{code}}]
  {{#access entity="event"}}
    * Op: {{operation}} / Role: {{role}} / Modifier: {{accessModifier}}
  {{/access}}
{{/zcl_events}}

{{/zcl_clusters}}
```

# Launching ZAP for Matter or Zigbee Applications

The following sections describe launching ZAP in standalone mode with the Matter or Zigbee-specific metadata. The idea is to launch ZAP with the correct arguments related to XML metadata (the clusters and device types definitions as per the CSA specifications) and the generation templates, which are used to generate the appropriate code.

## Launching ZAP with Matter

The following script picks up the correct metadata from the Matter SDK when launching ZAP.

https://github.com/project-chip/connectedhomeip/blob/master/scripts/tools/zap/run_zaptool.sh

Note: You can also take to the following Zigbee approach to launch ZAP in Matter.

## Launching ZAP with Zigbee

The following command launches ZAP with the ZCL specifications and generation templates from the SDK.

```
[zap-path] -z [sdk-path]/gsdk/app/zcl/zcl-zap.json -g [sdk-path]/gsdk/protocol/zigbee/app/framework/gen-template/gen-templates.json
```

- zap-path: This is the path to the ZAP source or executable
- sdk-path: This is the path to the SDK

## Launching ZAP without Metadata

Remember that when launching ZAP directly through an executable or from source using `npm run zap` you are launching ZAP with test metadata for Matter/Zigbee built in within ZAP and not the actual metadata coming from the Matter and Zigbee SDKs mentioned above. Therefore, remember to create your ZAP configurations by using the SDK metadata and not by opening ZAP directly with the built in test metadata.

# Generating Code for Matter, Zigbee or a Custom SDK

The following sections describe how to generate code using ZAP.

## Generate Code Using ZAP UI

Launch the ZAP UI as per the instructions in Launching ZAP for Matter or Zigbee and click on the Generate button in the top menu bar.

## Generate Code without the UI

The following instructions provide different ways of generating code through CLI without launching the ZAP UI.

### Generating Code from ZAP Source

Run the following command to generate code using ZAP from source:

```
node src-script/zap-generate.js --genResultFile --stateDirectory ~/.zap/gen -z ./zcl-builtin/silabs/zcl.json -g ./test/gen-template/zigbee/gen-templates.json -i ./test/resource/three-endpoint-device.zap -o ./tmp
```

### Generating Code from ZAP Executable

Run the following command to generate code using ZAP executable:

```
[zap-path] generate --genResultFile --stateDirectory ~/.zap/gen -z ./zcl-builtin/silabs/zcl.json -g ./test/gen-template/zigbee/gen-templates.json -i ./test/resource/three-endpoint-device.zap -o ./tmp
```

### Generating Code from ZAP CLI Executable

Run the following command to generate code using ZAP CLI Executable:

```
[zap-cli-path] generate --genResultFile --stateDirectory ~/.zap/gen -z ./zcl-builtin/silabs/zcl.json -g ./test/gen-template/zigbee/gen-templates.json -i ./test/resource/three-endpoint-device.zap -o ./tmp
```

**SILICON LABS**

# Update ZAP

## Update ZAP in Simplicity Studio

This mechanism can be used when working with Matter extension or Zigbee from the Silicon Labs SDK releases. ZAP can be updated within Simplicity Studio without a Simplicity Studio release by downloading the latest ZAP executable (recommended) or pulling the latest from ZAP source as shown in ZAP Installation Guide. After you have the latest ZAP based on your currently used OS, you can update ZAP within Studio as an adapter pack. Follow the instructions below after downloading the latest ZAP:

- Go to Simplicity Studio and select `Preferences > Simplicity Studio > Adapter Packs.`
- Click `Add...` and browse to the expanded ZAP folder you downloaded and click `Select Folder`.
- Click `Apply and Close` and then the newly-added ZAP will be used whenever a .zap file is opened.

Note: Sometimes there might be older instances of ZAP already running even after updating to the latest ZAP. Make sure to end all existing ZAP instances such that the newly fetched ZAP is used instead of an old instance, which is still working in the background.

## Update ZAP for Matter Development in Github

When working with the Matter or Matter-Silicon Labs repos on Github, set the environment variables with respect to ZAP to create/generate new ZAP configurations or re-generate existing sample ZAP configurations after applying changes to them. Set the `ZAP_DEVELOPMENT_PATH` to ZAP from source by pulling the latest or set `ZAP_INSTALLATION_PATH` to ZAP executable you downloaded last in your local directory. Note that when both `ZAP_DEVELOPMENT_PATH` and `ZAP_INSTALLATION_PATH` are set, `ZAP_DEVELOPMENT_PATH` is used.

The following are examples that show the above environment variables in use:

- Launching ZAP using Matter specification
- Regenerating all the sample ZAP configurations for Matter applications

Note: When using ZAP executables, ensure you are using an official release over a nightly release for more stability. See `Downloading the ZAP Executable` in ZAP Installation Guide