UG278: Zigbee Cluster Library over IP (ZCL/IP) User's Guide

ZCL/IP is an implementation of the Zigbee alliance dotdot specification, a generalization of all of the application layer functionality developed both in Zigbee PRO and the zigbee Cluster Library (ZCL) for use over a variety of transports. Using ZCL/IP, Silicon Labs has developed a Thread application layer, which is provided with the Silicon Labs Thread stack. This document provides details on working with ZCL/IP.

For background information about dotdot and ZCL/IP for Thread, see *UG103.11: Thread Fundamentals*.

KEY POINTS

- Getting started with ZCL/IP
- ZCL/IP configuration
- Using DTLS with ZCL/IP
- Using OTA Upgrade with ZCL/IP

1. Introduction

Zigbee Cluster Library over IP (ZCL/IP) is a transport-agnostic application networking layer protocol, developed and certified by the zigbee alliance as part of dotdot, a more generalized application layer specification. ZCL/IP's purpose is to establish a common protocol layer so that devices on different communication mediums can interact with each other seamlessly. The protocol is still under development, but making significant progress toward release to the public.

ZCL/IP is based on the zigbee Cluster Library (ZCL), which is a data model of behaviors and states that devices take on. In the ZCL, these behaviors are referred to as *commands* and these states are referred to as *attributes*. For more information about the ZCL, see *UG103.02: Zigbee Fundamentals*.

ZCL/IP takes the commands and attributes described in the ZCL and defines an over-the-air protocol to use to exchange this information between devices. ZCL/IP uses the Constrained Application Protocol (CoAP) as its transport layer. In ZCL/IP messages, the CoAP protocol data unit contains information about the ZCL command or attributes to which the message pertains, and the service data unit contains information further describing the command or attributes. CoAP is very similar to HTTP. CoAP operates under a request/response communication scheme, and each message contains a code that specifies a request action (GET, PUT, POST, DELETE, etc.), response success (2.01 Created, 2.04 Changed, 2.05 Content, etc.), or response failure (4.00 Bad Request, 4.04 Not Found, 5.00 Internal Server Error). For more information about CoAP, see RFC 7252.

CoAP can theoretically run over any transport layer. Therefore, ZCL/IP theoretically runs over any transport, network, and physical layer. The Silicon Labs ZCL/IP implementation for Thread uses UDP as a transport layer, as this functionality is already included in the Silicon Labs Thread stack. For more information about Thread and the Silicon Labs Thread stack, see *UG103.11: Thread Fundamentals*.

2. Getting Started

2.1 Using ZCL/IP Example Applications

Whether you have had previous experience with zigbee (perhaps through working with the Silicon Labs EmberZNet stack) or are completely new to zigbee, the best place to get started with ZCL/IP is through the SL-Thread ZCL/IP examples applications in Simplicity Studio. A number of ZCL/IP examples are Included in the SL-Thread release, but the best starting places are the Light and Switch example applications. These example applications both bring up communications on the same Thread network, and can then be controlled through development board buttons. The button interface for these example applications is explained in detail in the sample applications description, shown on the **General** tab of Simplicity Studio's IDE, also known as AppBuilder. While using the buttons to send ZCL/IP messages between the Light and Switch sample apps, Silicon Labs recommends you have a Network Analyzer packet capture running, as this is an excellent way to better understand the ZCL/IP messaging format. This messaging format will be new to users with and without previous experience with zigbee. The Light and Switch sample apps demonstrate many different kinds of ZCL/IP messages, such as, but not limited to, commands, notifications (reporting), and EZ-Mode. Implementing a Light/Switch simple network and using Network Analyzer are described in detail in *QSG113: Getting Started with Silicon Labs Thread*.

You can use the widgets provided in the AppBuilder **ZCL over IP** tab to configure the application by adding and removing endpoints, device types, clusters, commands, and attributes. Adding and removing information on the **ZCL over IP** tab in AppBuilder will result in changes to the code generated by AppBuilder, and therefore different configuration of the ZCL/IP application.

2.2 Using Existing EmberZNet ZCL Applications

You may wish to migrate some of the functionality from existing ZCL applications running over the EmberZNet PRO stack to ZCL/IP applications running over the Silicon Labs Thread stack. This functionality may include cluster implementations, special ZCL logic, and ZCL configuration. As the ZCL/IP protocol runs over a different stack, cluster implementations have been rewritten to support the Silicon Labs Thread Stack. Special ZCL logic from previous applications can certainly be reused in ZCL/IP applications. You can add source files to an AppBuilder application through the **Additional Files** table found in the **Other** tab. You may be able to reuse generic ZCL logic from previous application since the ZCL data model has stayed the same for ZCL/IP. Since AppBuilder uses the same user interface for both the Silicon Labs EmberZNet stack and the Silicon Labs Thread stack, you should be able to copy a previous application's ZCL configuration by choosing the same options in the **ZCL over IP** tab of AppBuilder.

3. Configuration

3.1 CoAP

The Silicon Labs ZCL/IP implementation relies on the Constrained Application Protocol or CoAP for its transport layer. The Constrained Application Protocol is defined in RFC 7252. Silicon Labs includes a CoAP implementation along with the Silicon Labs Thread Stack. The CoAP implementation used for ZCL/IP is located at:

```
<stack install location>/app/coap
```

The CoAP implementation provides APIs for reliably sending and receiving messages over the Thread network.

Sending a message using CoAP

Messages can be sent over CoAP using the API `emberCoapSend`. Helper APIs named `emberCoapGet`, `emberCoapPut`, `emberCoapPost`, and `emberCoapDelete` also end up calling `emberCoapSend`.

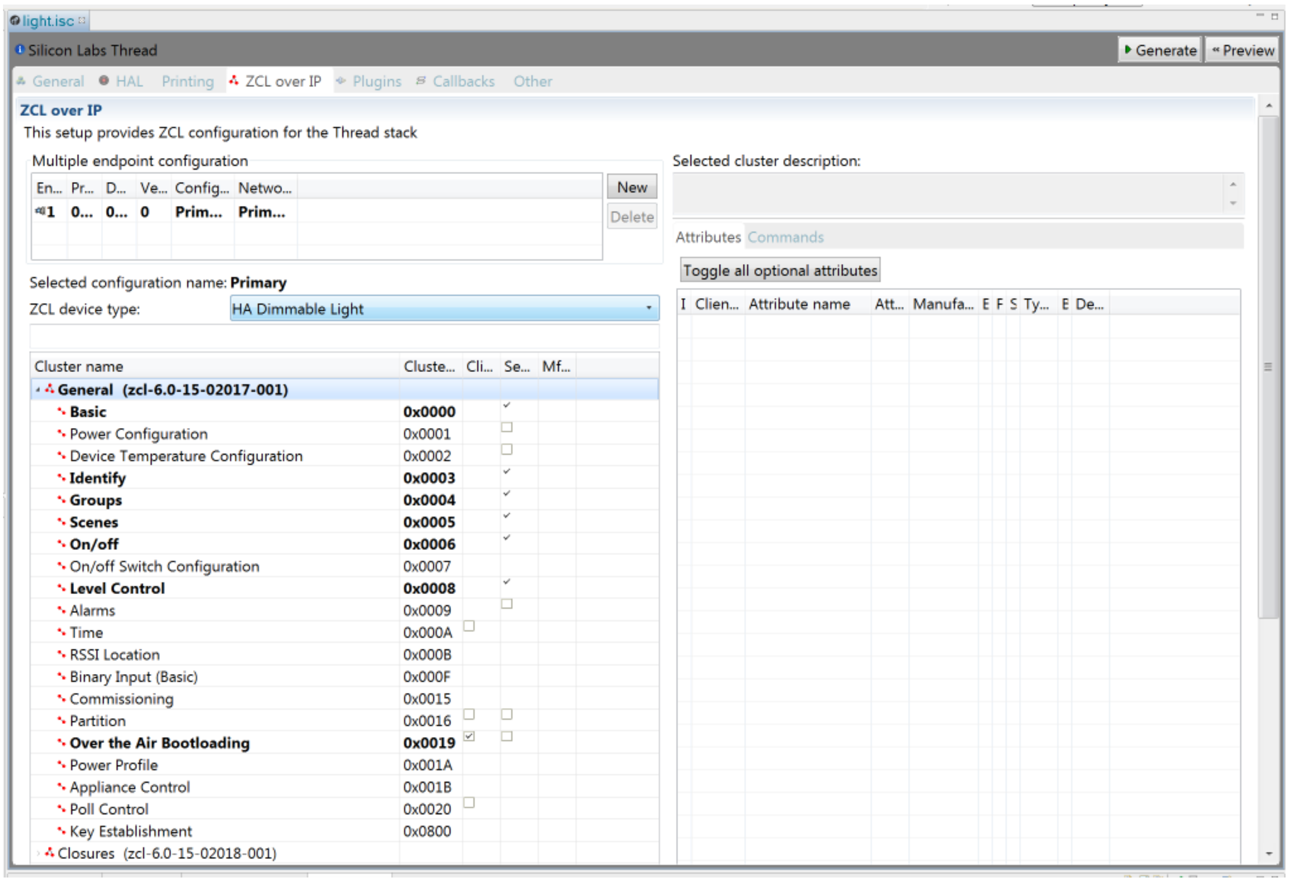
Receiving a message using CoAP

All CoAP messages meant for the application are passed to the function `emberCoapRequestHandler`. The Silicon Labs Thread Application Framework has a plugin named CoAP Dispatch that multiplexes the calls to `emberCoapRequestHandler` based on the method with which the CoAP message was sent and the URI to which the CoAP message was sent.

For more information about the Silicon Labs Thread Stack CoAP API, see the CoAP module in the Silicon Labs Thread Stack API Reference Guide.

3.2 ZCL

The zigbee Cluster Library is configured through the **ZCL over IP** tab in AppBuilder. If you are familiar with EmberZNet, configuration is the same as for any zigbee application.



ZCL over IP

This setup provides ZCL configuration for the Thread stack

Multiple endpoint configuration

En...	Pr...	D...	Ve...	Config...	Netwo...
1	0...	0...	0	Prim...	Prim...

Selected cluster description:

Selected configuration name: **Primary**

ZCL device type: HA Dimmable Light

I	Clien...	Attribute name	Att...	Manufa...	E	F	S	Ty...	E	De...

Cluster name	Cluste...	Cli...	Se...	Mf...
General (zcl-6.0-15-02017-001)				
Basic	**0x0000**		✓	
Power Configuration	0x0001		☐	
Device Temperature Configuration	0x0002		☐	
Identify	**0x0003**		✓	
Groups	**0x0004**		✓	
Scenes	**0x0005**		✓	
On/off	**0x0006**		✓	
On/off Switch Configuration	0x0007			
Level Control	**0x0008**		✓	
Alarms	0x0009		☐	
Time	0x000A	☐		
RSSI Location	0x000B			
Binary Input (Basic)	0x000F			
Commissioning	0x0015			
Partition	0x0016	☐	☐	
Over the Air Bootloading	**0x0019**	☑	☐	
Power Profile	0x001A			
Appliance Control	0x001B			
Poll Control	0x0020	☐		
Key Establishment	0x0800			
Closures (zcl-6.0-15-02018-001)				

All of the code for Silicon Labs ZCL/IP implementation is shipped with the Thread stack and is located at:

<stack install location>/app/thread/plugin/zcl

This directory includes an ever-growing list of plugins that comprise the Silicon Labs ZCL/IP application layer implementation. The most important of these and probably the best place to begin looking into the code is the ZCL Core plugin.

The ZCL Core plugin provides the foundation for the ZCL/IP implementation including but not limited to such basic functionality as:

- Attribute management
- Binding management
- Command handling and dispatching
- Endpoint management
- Group management
- Reporting configuration management
- Notification handling and dispatching
- Device discovery
- Application provisioning
- General ZCL/IP utilities

For more information about the ZCL Core plugin API, see the ZCLIP module in the Silicon Labs Thread Stack API Guide.

4. Using DTLS with ZCL/IP

As the Silicon Labs ZCL/IP implementation uses CoAP as the underlying mechanism for messaging, it can also use Secure CoAP functionality in order to send data securely between two nodes.

4.1 Transport Layer Security (TLS) Methodologies

Silicon Labs has implemented Secure CoAP functionality on two different TLS (transport layer security) code bases: the Silicon Labs custom implementation and ARM's mbed TLS.

Silicon Labs Thread and ZCL/IP communications security is, by default, provided through the Silicon Labs implementation of TLS. It is a stable and well-tested protocol that has been in use since 2010.

This implementation provides the following features required by Thread and ZCL/IP while keeping application image size small enough to flash onto devices with less than 256 kB data storage, including the EM3x and EFR3xG1 families.

- An SSL / TLS (3.3) implementation
- AES crypto support
- ECC support for ECJPAKE, and ECDHE, ECDSA support for secp256r1 (standardized 256-bits NIST) curves
- SHA-256 hash algorithm
- Pre-compiled X.509 certificates
- ECC private and public keys: PKCS#8 unencrypted private key and PKCS#8 public key modes.

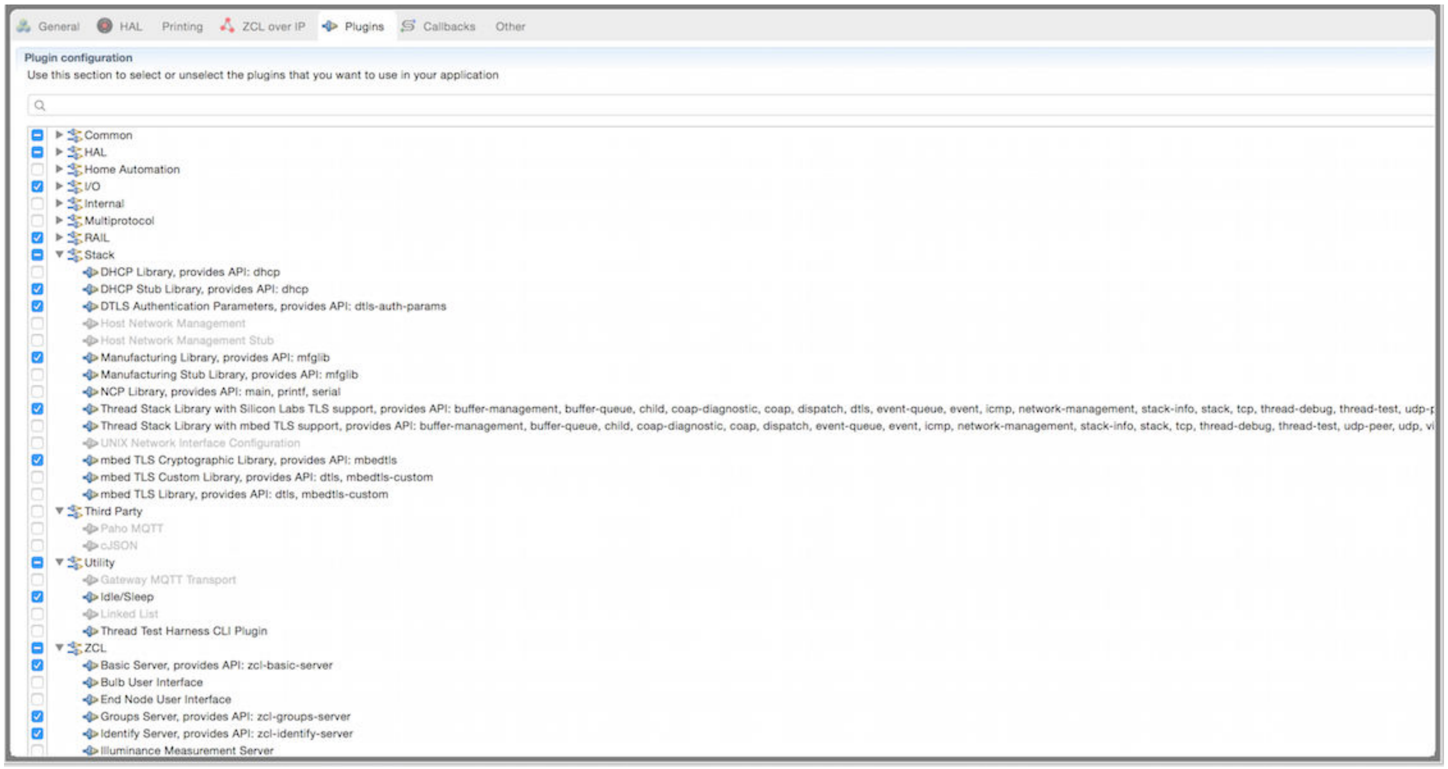
For more information on Thread and dotdot security requirements, see *UG103.11: Application Development Fundamentals: Thread*, and the zcl/ip base device specification.

Beginning with Silicon Labs Thread release 2.3, users who are developing for any device with >256 kB data storage, such as the EFR32xG12, have the option of using ARM mbed TLS for communications security. ARM mbed TLS is a widely-used and highly configurable implementation that supports additional features, including:

- SSL version 3
- Variety of key exchange methods (RSA, DHE-RSA, ECDHE-RSA, ECDH-RSA, PSK, DHE-PSK, and others)
- Variety of cryptographic algorithms (AES, 3DES, DES, XTEA, and others)
- Various modes of operations (CBC, EBC, CFB, and others)
- Various hash algorithms (MD2, MD4, SHA-1, SHA-256, and others)
- ECC support including ECDHE, ECDSA and ECJPAKE, among others
- NIST standardized CTR_DRBG and HMAC_DRBG random number generators
- Supports a variety of underlying technologies – X.509 certificates; RSA and ECC private and public key reading

For more information on ARM mbed TLS, see https://tls.mbed.org/core-features and https://tls.mbed.org/standard-compliance. For more information on using mbed TLS when implementing a secure CoAP solution, see *AN1083: Creating and Using a Secure CoAP Connection with ARM's mbed TLS*.

Plugins in the Stack group on the **Plugin**s tab control whether Silicon Labs TLS or ARM mbed TLS is used in your application. The stack group is shown in the following figure.



Plugin configuration

Use this section to select or unselect the plugins that you want to use in your application

- ▬ Common
- ▬ HAL
- ☐ Home Automation
- ✓ I/O
- ☐ Internal
- ☐ Multiprotocol
- ✓ RAIL
- ▬ Stack
  - ☐ DHCP Library, provides API: dhcp
  - ✓ DHCP Stub Library, provides API: dhcp
  - ✓ DTLS Authentication Parameters, provides API: dtls-auth-params
  - ☐ Host Network Management
  - ☐ Host Network Management Stub
  - ✓ Manufacturing Library, provides API: mfglib
  - ☐ Manufacturing Stub Library, provides API: mfglib
  - ☐ NCP Library, provides API: main, printf, serial
  - ✓ Thread Stack Library with Silicon Labs TLS support, provides API: buffer-management, buffer-queue, child, coap-diagnostic, coap, dispatch, dtls, event-queue, event, icmp, network-management, stack-info, stack, tcp, thread-debug, thread-test, udp-p
  - ☐ Thread Stack Library with mbed TLS support, provides API: buffer-management, buffer-queue, child, coap-diagnostic, coap, dispatch, event-queue, event, icmp, network-management, stack-info, stack, tcp, thread-debug, thread-test, udp-peer, udp, vi
  - ☐ UNIX Network Interface Configuration
  - ✓ mbed TLS Cryptographic Library, provides API: mbedtls
  - ☐ mbed TLS Custom Library, provides API: dtls, mbedtls-custom
  - ☐ mbed TLS Library, provides API: dtls, mbedtls-custom
- ☐ Third Party
  - ☐ Paho MQTT
  - ☐ cJSON
- ▬ Utility
  - ☐ Gateway MQTT Transport
  - ✓ Idle/Sleep
  - ☐ Linked List
  - ☐ Thread Test Harness CLI Plugin
- ▬ ZCL
  - ✓ Basic Server, provides API: zcl-basic-server
  - ☐ Bulb User Interface
  - ☐ End Node User Interface
  - ✓ Groups Server, provides API: zcl-groups-server
  - ✓ Identify Server, provides API: zcl-identify-server
  - ☐ Illuminance Measurement Server

For Silicon Labs TLS, select the following two precompiled libraries. This is the default configuration for example applications:

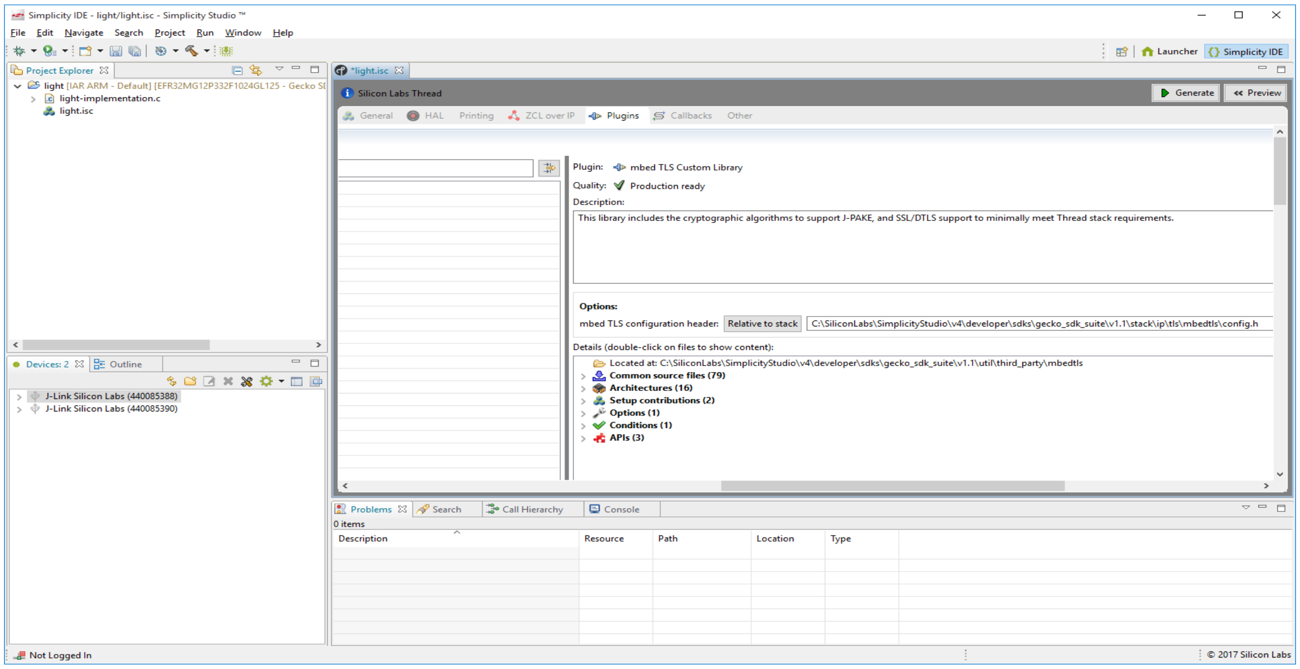
- Thread Stack Library with legacy TLS support
- mbed TLS Cryptographic Library: Provides hardware crypto acceleration support, use only with the thread-stack library.

For ARM mbed TLS in an implementation that provides all functionality required by ZCL/IP, select the following two precompiled libraries:

- Thread Stack Library with mbedTLS support
- mbed TLS Library

For ARM mbed TLS in a custom stack implementation that you can build, select the following two libraries:

- Thread Stack Library with mbed TLS support
- mbed TLS Custom Library: Includes a configurable option to use a custom mbed TLS header. By default, this option takes a custom header that is configured for minimal stack (commissioning) support. The header can be overwritten using one with additional options.



Plugin: mbed TLS Custom Library
Quality: Production ready
Description:
This library includes the cryptographic algorithms to support J-PAKE, and SSL/DTLS support to minimally meet Thread stack requirements.

Options:
mbed TLS configuration header: Relative to stack C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.1\stack\ip\tls\mbedtls\config.h

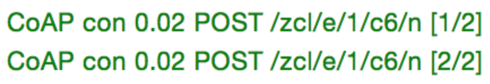
Details (double-click on files to show content):
Located at: C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.1\util\third_party\mbedtls
Common source files (79)
Architectures (16)
Setup contributions (2)
Options (1)
Conditions (1)
APIs (3)

4.2 Example DTLS Usage in ZCL IP

The DTLS-CLI plugin, which provides secure CoAP functionality, is enabled by default in both the Light and Switch example applications.

When the DTLS-CLI plugin is enabled and a DTLS session is established, the ZCL/IP messaging API sends messages over the secure session. When a secure message is received, it is parsed through the `emberProcessCoap` function in app/thread/plugin/dtls-cli/dtls-cli.c

To see this in operation, create the Light and Switch examples, start a network, and observe interactions in Network Analyzer, as described in *QSG113: Getting Started with Silicon Labs Thread*. Notice that, when toggling the light from the switch, Network Analyzer shows regular CoAP messages.



```
CoAP con 0.02 POST /zcl/e/1/c6/n [1/2]
CoAP con 0.02 POST /zcl/e/1/c6/n [2/2]
```

To set up and observe a DTLS connection:

1. Get the IP Address of the light using: `info`

```
light> info
network status: 0x03
eui64: >000B57FFFE25FFA3
network id: precommissioned
node type: 0x02
extended pan id: >4F8EC75FB4E1EFC6
pan id: 0x8394
channel: 24
radio tx power: 4 dBm
ula prefix: fd01::/64
local ip 0: fd01::e48a:723f:8730:140d
local ip 1: fe80::70d7:25e3:b9d0:4f6a
```

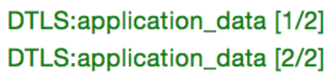
2. Open the DTLS connection using: `dtls open <ip> <port>`

```
switch> dtls open "fd01::e48a:723f:8730:140d" 5684
```

3. DTLS Session established callbacks will fire on both nodes

```
light> secure session available: 1 (server)
switch > secure session available: 1 (client)
```

4. Use the application securely: When toggling the light from the switch, Network Analyzer shows Secured messages.



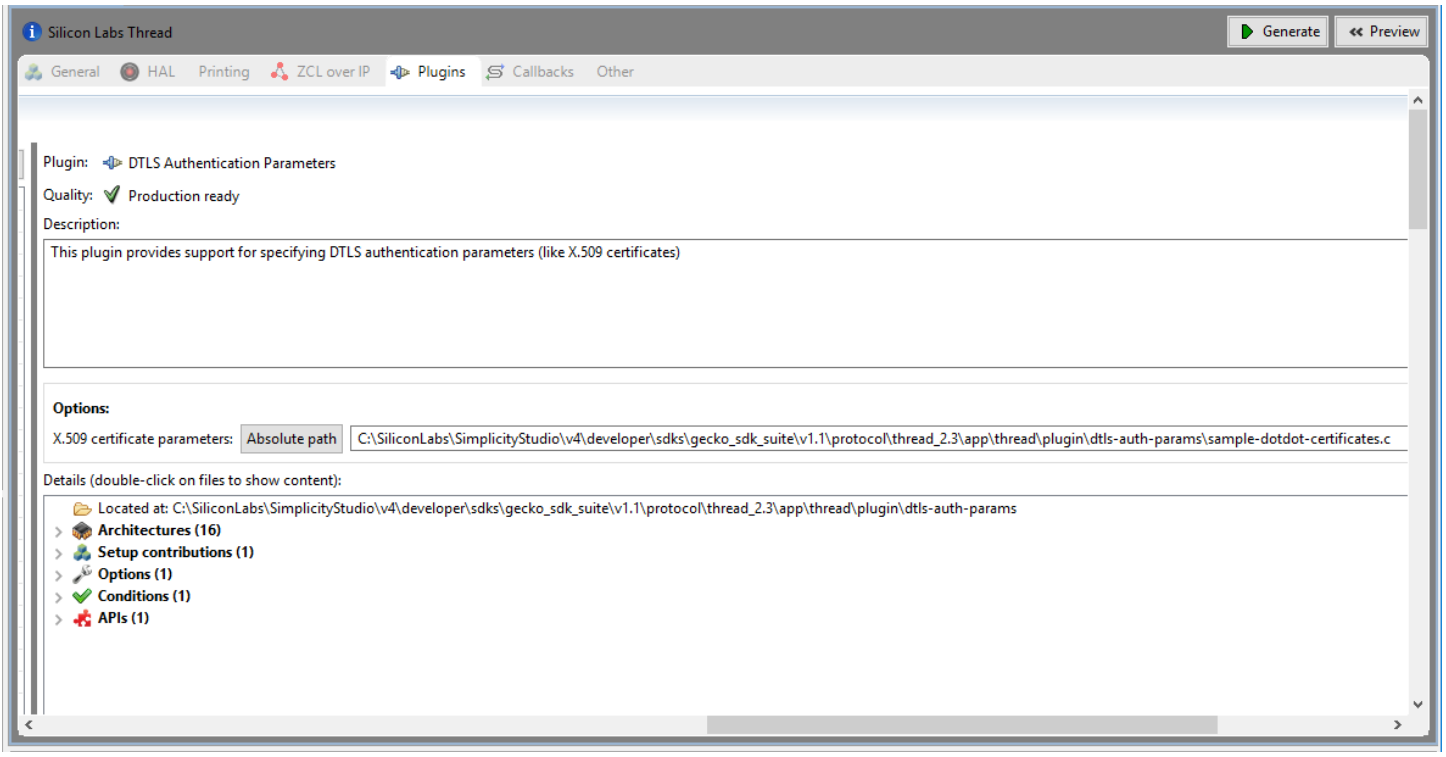
```
DTLS:application_data [1/2]
DTLS:application_data [2/2]
```

4.3 DTLS Authentication Parameters

Regardless of whether you are using Silicon Labs TLS or ARM mbed TLS, the DTLS Authentication Parameters (dtls-auth-params) plugin must be selected. This plugin provides a stub implementation of the certificates required for authentication, as well as "real" certificates for the ZCL examples. Refer to the following files for the code used to generate the stub and demo certificates:

- app/thread/plugin/dtls-auth-params/stub-certificates.c: Stub / empty certificate definitions
- app/thread/plugin/dtls-auth-params/sample-dotdot-certificates.c: Sample X.509 certificates being used to demo ZCL apps.

The following examples show how to create custom X.509 certificates for use with the dtls-auth-params plugin. The examples are intended for a Linux or other Unix (Mac OS X, etc.) environment that has the OpenSSL libraries installed. If you are using a different environment, consult documentation for how to create .pem files.



Plugin: DTLS Authentication Parameters

Quality: Production ready

Description:

This plugin provides support for specifying DTLS authentication parameters (like X.509 certificates)

Options:

X.509 certificate parameters: Absolute path C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.1\protocol\thread_2.3\app\thread\plugin\dtls-auth-params\sample-dotdot-certificates.c

Details (double-click on files to show content):

Located at: C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.1\protocol\thread_2.3\app\thread\plugin\dtls-auth-params

- Architectures (16)
- Setup contributions (1)
- Options (1)
- Conditions (1)
- APIs (1)

4.3.1 Creating Your Own Certificate Authority

1. Generate a Certificate Authority (CA) private key using an Elliptical Curve (EC).

```
openssl ecparam -genkey -name secp256r1 -out ca-key.pem
```

2. Create a CA certificate using the above private key.

```
openssl req -x509 -new -SHA256 -nodes -key ca-key.pem -days 3650 -out ca-cert.pem
```

3. Create a Certificate Signing Request (CSR) using the private key.

```
openssl req -new -SHA256 -key ca-key.pem -nodes -out ca-csr.pem
```

4. Verify the details in the CSR.

```
openssl req -in ca-csr.pem -noout -text
```

5. Generate an unencrypted pkcs8 format key for the DTLS plugin by executing this command:

```
openssl pkcs8 -topk8 -nocrypt -in ca-key.pem -out ca-pkcs8-key.pem
```

4.3.2 Creating Your Own Device Certificate

To create your own device certificate (in the format that dotdot applications require), use the Privacy-Enhanced Mail (PEM) files generated in section 4.3.1 Creating Your Own Certificate Authority, or from another CA authority, and execute this command:

```
openssl x509 -req -SHA256 -days 3650 -in ca-csr.pem -CA ca-cert.pem -CAkey ca-key.pem
-CAcreateserial -out device-cert.pem
```

4.3.3 Converting PEM Files into C Files for Use in a DTLS Application

The last step is to convert the PEM files generated in the two previous steps into a .c file that the DTLS Authentication Parameters (dtls-auth-params) plugin can accept as an input.

1. Save all the .pem files generated in the previous two procedures to your hard drive.
2. Change to the directory `app/thread/plugin/dtls-auth-params`
3. Execute this command.

   ```
   ./translate-certificates.sh ca-cert.pem ca-key.pem ca-pkcs8-key.pem device-cert.pem
   ```

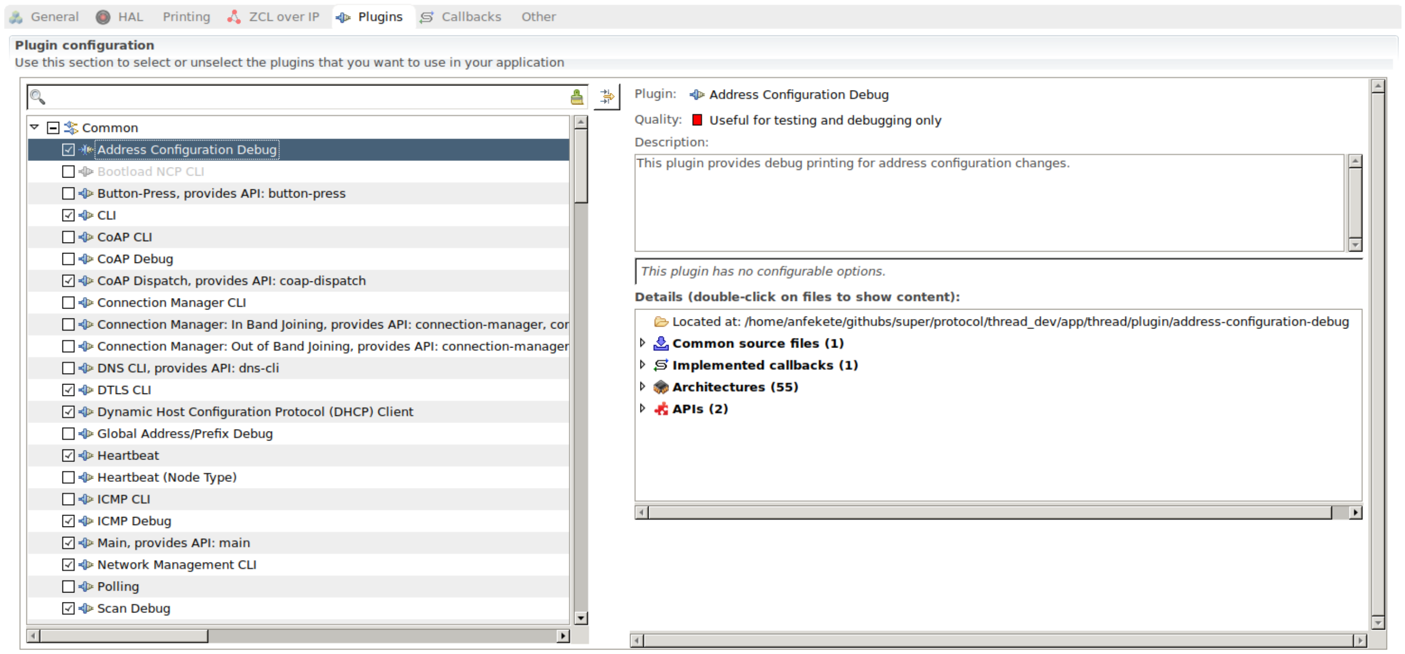
   This creates a .c file named translated-certificates.c.
4. (Optional) Rename translated-certificates.c.
5. Use translated-certificates.c as an option to the DTLS Authentication Parameters (dtls-auth-params) plugin.

5. Using the OTA Upgrade Solution in ZCL/ IP

The dotdot OTA (Over-The-Air) Upgrade (OTA) solution is based on the analogous solution in zigbee PRO. The dotdot OTA solution takes the concepts specific to zigbee PRO from the OTA Upgrading cluster (as specified in chapter 11 of the zigbee Cluster Library (ZCL) specification), and generalizes them so that they can be used in the dotdot transport-agnostic application layer. In addition to this information, please see *AN716: Image Builder Instructions* and *AN728: Over-the-Air Bootloader Server and Client Setup* for more information about the zigbee PRO OTA solution.

5.1 OTA Functionality in Example Applications

The Silicon Labs Thread 2.3 and higher release contains three example applications that demonstrate dotdot OTA functionality. The Light and Switch examples function as OTA Bootload Client devices, and the Border Router Management Application functions as an OTA Bootload Server device. The dotdot OTA functionality can be configured through the OTA Bootload Client (shown below) and OTA Bootload Server plugins in the ZCL plugins group.



Plugin configuration
Use this section to select or unselect the plugins that you want to use in your application

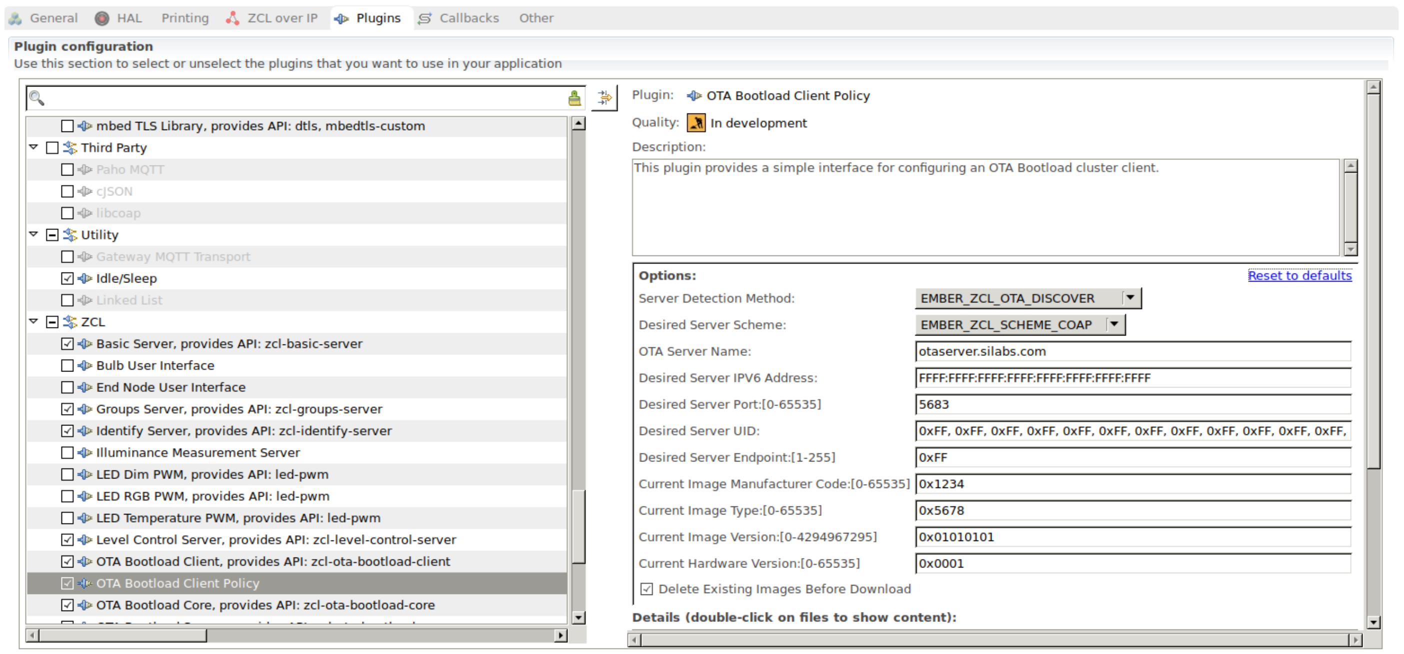
- Common
  - ☑ Address Configuration Debug
  - ☐ Bootload NCP CLI
  - ☐ Button-Press, provides API: button-press
  - ☑ CLI
  - ☐ CoAP CLI
  - ☐ CoAP Debug
  - ☑ CoAP Dispatch, provides API: coap-dispatch
  - ☐ Connection Manager CLI
  - ☐ Connection Manager: In Band Joining, provides API: connection-manager, cor
  - ☐ Connection Manager: Out of Band Joining, provides API: connection-manager
  - ☐ DNS CLI, provides API: dns-cli
  - ☑ DTLS CLI
  - ☑ Dynamic Host Configuration Protocol (DHCP) Client
  - ☐ Global Address/Prefix Debug
  - ☑ Heartbeat
  - ☐ Heartbeat (Node Type)
  - ☐ ICMP CLI
  - ☑ ICMP Debug
  - ☑ Main, provides API: main
  - ☑ Network Management CLI
  - ☐ Polling
  - ☑ Scan Debug

Plugin: Address Configuration Debug
Quality: 🟥 Useful for testing and debugging only
Description:
This plugin provides debug printing for address configuration changes.

This plugin has no configurable options.

Details (double-click on files to show content):
Located at: /home/anfekete/githubs/super/protocol/thread_dev/app/thread/plugin/address-configuration-debug
Common source files (1)
Implemented callbacks (1)
Architectures (55)
APIs (2)

In addition to these plugins, the OTA Bootload Client Policy (shown below) and OTA Bootload Server Policy plugins provide a helpful abstraction for configuring an application to run as an OTA device.



Plugin configuration
Use this section to select or unselect the plugins that you want to use in your application

- [ ] mbed TLS Library, provides API: dtls, mbedtls-custom
- [ ] Third Party
  - [ ] Paho MQTT
  - [ ] cJSON
  - [ ] libcoap
- [ ] Utility
  - [ ] Gateway MQTT Transport
  - [✓] Idle/Sleep
  - [ ] Linked List
- [ ] ZCL
  - [✓] Basic Server, provides API: zcl-basic-server
  - [ ] Bulb User Interface
  - [ ] End Node User Interface
  - [✓] Groups Server, provides API: zcl-groups-server
  - [✓] Identify Server, provides API: zcl-identify-server
  - [ ] Illuminance Measurement Server
  - [ ] LED Dim PWM, provides API: led-pwm
  - [ ] LED RGB PWM, provides API: led-pwm
  - [ ] LED Temperature PWM, provides API: led-pwm
  - [✓] Level Control Server, provides API: zcl-level-control-server
  - [✓] OTA Bootload Client, provides API: zcl-ota-bootload-client
  - [✓] OTA Bootload Client Policy
  - [✓] OTA Bootload Core, provides API: zcl-ota-bootload-core

Plugin: OTA Bootload Client Policy
Quality: In development
Description:
This plugin provides a simple interface for configuring an OTA Bootload cluster client.

Options: Reset to defaults

Server Detection Method:	EMBER_ZCL_OTA_DISCOVER
Desired Server Scheme:	EMBER_ZCL_SCHEME_COAP
OTA Server Name:	otaserver.silabs.com
Desired Server IPV6 Address:	FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
Desired Server Port:[0-65535]	5683
Desired Server UID:	0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
Desired Server Endpoint:[1-255]	0xFF
Current Image Manufacturer Code:[0-65535]	0x1234
Current Image Type:[0-65535]	0x5678
Current Image Version:[0-4294967295]	0x01010101
Current Hardware Version:[0-65535]	0x0001

[✓] Delete Existing Images Before Download

Details (double-click on files to show content):

There are three types of server detection methods: Discover, Static IP, and DNS. Each method uses the subsequent parameters in different ways.

- **Discover**

  This method tries to discover an update server that has announced itself on the mesh. If multiple servers are on the mesh, the `IPV6 Address' parameter can be set to that of the desired server. This acts as a filter to discover only the server that matches the given address. The same function can be performed with the `Port', `UID', and `Endpoint' parameters. Note, this is not a mask value, but rather an explicit match. A wildcard (all 0xFF) will match anything. The `OTA Server Name' is unused.

- **Static IP**

  All server parameters (with the exception of 'OTA Server Name') are used exactly as given to communicate with an OTA server. No parameter may be a wildcard (all 0xFF).

- **DNS**

  A DNS server is necessary for this setting. It is typically implemented via a Border Router that is on the mesh. The Border Router declares itself as having Network Discovery services. Using the `OTA Server Name' field, a query for the IPv6 address is initiated. The `IPV6 Address' field is ignored. The `Port', `UID', and `Endpoint' parameters are used as they appear in AppBuilder. These parameters cannot be wildcards (all 0xFF).

Depending on the method selected, the device will seek out the update server. An on-mesh upgrade server example is located in the Light (HOST) sample application. For upgrades in the cloud, the zclip Javascript sample server is offered. This server may be accessed through a Border Router.

5.2 Using the dotdot OTA Image Builder Utility

The Silicon Labs Thread 2.3 release contains a simple utility to generate an OTA file for a provided image file (such as an EBL or GBL file). This utility is similar to, but simpler than, the Image Builder utility provided in the EmberZNet SDK.

OTA files are a wrapper around binary data, such as upgrade images. The OTA file wrapper enables the data to be transferred securely over the air from an OTA Bootload Server device to an OTA Bootload Client device. An OTA file starts with a header that describes the data contained in the file. The header contains fields such as, but not limited to, the manufacturer code, the type, and the version describing the binary data. The primary purpose of the Image Builder utility is to wrap a provided image file with this OTA header so that the file can then be transferred over the air.

The interface for the Silicon Labs Thread Image Builder utility has five required command line arguments:

- The input image file name
- The output OTA file name
- The manufacturer code of the input image file (16-bit)
- Tthe type of the input image file (16-bit)
- The version of the input image file (32 bit).

Here is an example invocation of the utility.

```
image-builder-windows.exe light.gbl 1234-ABCD-10001000-light.ota 0x1234 0xABCD 0x10001000
```

The preceding command generates an OTA file named 1234-ABCD-10001000-light.ota, which contains the image file light.gbl, and has a manufacturer code of 0x1234, an image type of 0xABCD, and a file version of 0x10001000. This 1234-ABCD-10001000-light.ota file can then be added to an OTA Bootload Server and downloaded to an OTA Bootload Client for upgrade.

The Image Builder executable is located in <studio installation>\developer\sdks\gecko_sdk_suite\v1.1\protocol\thread_n.n\tool.

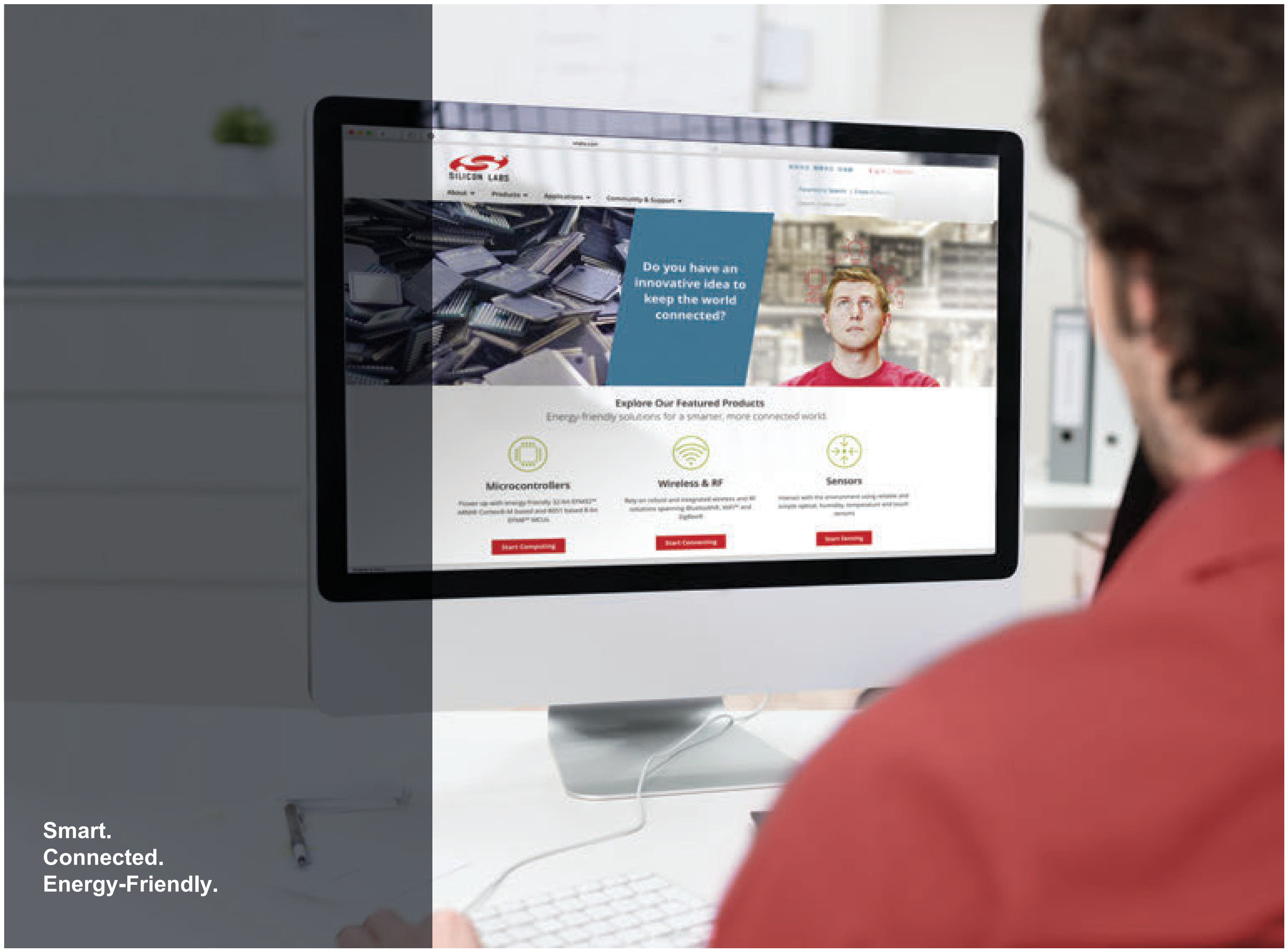
Note: The Image Builder utility found in a parallel location in the EmberZNet release should not be used to create dotdot OTA files for use in ZCL/IP.

5.3 Comparison with zigbee PRO OTA

The basic structure of the OTA file format has stayed the same. For example, an OTA file can still be uniquely identified by the combination of a manufacturer code, a file type, and a file version. The only changes made to the OTA file format are simple generalizations. For example, instead of specifying which version of the zigbee networking stack the OTA file applies to, the file specifies the type of general communication stack for which it is meant. In addition, the concept of an IEEE address has been replaced with a Unique Identifier (UID). A UID is now used anywhere that an IEEE address was used in the zigbee PRO OTA solution. Using a UID provides more authenticity and applicability to a general communication stack. Finally, it is a mandate that these dotdot OTA files are secured using IP security credentials, to ensure a secure transport of OTA files.

There are two main differences between the zigbee PRO and dotdot OTA designs. The first main difference is the image transfer mechanism. In dotdot OTA, the communication protocol with which the client and server devices transfer the image is left up to the application. A manufacturer is free to decide the method that they would prefer to use when sending an OTA file to one or more client devices. In the current dotdot OTA solution, the transfer mechanism of choice is Constrained Application Protocol (CoAP) block transfer, as specified in RFC7959. CoAP functionality is already required in dotdot implementations, so this method of file transfer should fit in easily for applications, while still maintaining manageable code size. Other examples of image transfer mechanisms are multicast block transfer and server "push."

The second main difference between zigbee PRO and dotdot OTA designs is server discovery. While it is possible that a client device exists on the same mesh network as its designated upgrade server, in reality, many manufacturer will wish to store their upgrade images in an off-mesh location, perhaps in the cloud. The dotdot OTA solution provides the ability for client devices to communicate with and download an image from a server device that exists off-mesh. A manufacturer can write the domain name of a server device to a client device, and the client device can then use Domain Name Service (DNS) to resolve the domain name into an IPV6 address to use to communicate with the server device. This feature enables manufacturers to cut out a large step in their upgrade process, solving many issues raised against the zigbee PRO OTA solution.


Smart.
Connected.
Energy-Friendly.
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Energy-friendly solutions for a smarter, more connected world
Microcontrollers
Wireless & RF
Sensors

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com