# UG129: Zigbee® Gateway Reference Design User's Guide (RD-0001-0201, RD-0002-0201)

*RD-0001-0201: Zigbee Wi-Fi/Ethernet Gateway Reference Design* and *RD-0002-0201: Zigbee USB Virtual Gateway Reference Design* are designed to demonstrate Zigbee coordinator functionality with Silicon Labs Zigbee reference designs:

- *RD-0039-0201: Capacitive Sense Dimmer Switch Reference Design*
- *RD-0030-0201: Contact Sensor Reference Design*
- *RD-0051-0201: Smart Outlet Reference Design*
- *RD-0100-0201: Smart Outlet Reference Design*
- *RD-0078-0201: Occupancy Sensor Reference Design*
- *RD-0099-0201: Occupancy Sensor Reference Design*
- *RD-0020-0601: Lighting Reference Design*
- *RD-0035-0601: Lighting Reference Design*
- *RD-0085-0401: Lighting Reference Design*
- *RD-0098-0401: Lighting Reference Design*

This user's guide refers to Silicon Labs Gateway software release version 2.5.0.

**KEY POINTS**

- Describes Zigbee gateway reference designs.
- Provides step-by-step instructions for the installation and configuration process.
- Explains the gateway functionality.
- Details the software architecture of the gateway.
- Offers troubleshooting solutions and references for common issues.

# 1. Introduction

This section introduces *RD-0001-0201: Zigbee Wi-Fi/Ethernet Gateway Reference Design* and *RD-0002-0201: Zigbee USB Virtual Gateway Reference Design*.

The software is distributed as three software components:

- Z3Gateway Application
- NodeJS Server Application
- ReactJS Front-End Application

The Z3Gateway Application offers three transport options for inter-process and off-gateway communication:

- Telnet
- CoAP
- MQTT

The MQTT transport is used for inter-process communication with the NodeJS Server Application.

The software binaries are available using the Linux apt package manager, as described in section 2. Installation and Configuration for the Zigbee Wi-Fi/Ethernet Gateway (RD-0001-0201) and 3. Installation and Configuration for the Zigbee USB Virtual Gateway (RD-0002-0201).

The software source code for the Z3Gateway application is available as a host example application distributed with the EmberZNet PRO 6.2.0 Zigbee stack. For more information on how access the Zigbee stack please see http://www.silabs.com/products/wireless/mesh-networking/Pages/getting-started-with-mighty-gecko-zigbee.aspx.

The software source code for the NodeJS Server Application and ReactJS Front-End Application is available on github at https://github.com/SiliconLabs/gateway-management-ui and is also installed on the target system at /opt/siliconlabs/zigbeegateway/gateway_management_ui.

The kits RD-0001-0201 and RD-0002-0201 are no longer available for purchase from Silicon Labs. Most users found it more convenient to acquire the Raspberry Pi hardware from third-party sources. Throughout this document the part numbers RD-0001-0201 and RD-0002-0201 are used to refer to the kit acquired previously from Silicon Labs or built with components from a third-party.

The kits RD-0001-0201 and RD-0002-0201 shipped from Silicon Labs used a CEL USB dongle as the NCP. To improve ease-of-use, the CEL USB dongle is replaced with one of the three wireless starter kit mainboards available with the EFR32 Mighty Gecko Wireless SoC Starter Kit (SLWSTK6000B). Notes are made throughout this document to guide users wishing to continue using their CEL NCP USB dongle and EM35x-DEV end-devices.

The WiFi / Ethernet Gateway runs on a Raspberry Pi 2 Model B or Raspberry Pi 3 Model B computer with Raspbian Linux and Zigbee NCP (network co-processor). A Silicon Labs EFR32 Mighty Gecko Wireless Starter Kit such as SLWSTK6000B is required as the NCP. The gateway includes a Wi-Fi soft access point and a web server that presents a user interface to a desktop or mobile web browser. The web browser can run on a device connected via Wi-Fi or wired local area network (LAN). A typical Zigbee system configuration with the Zigbee Wi-Fi/Ethernet Gateway is shown in the following figure.

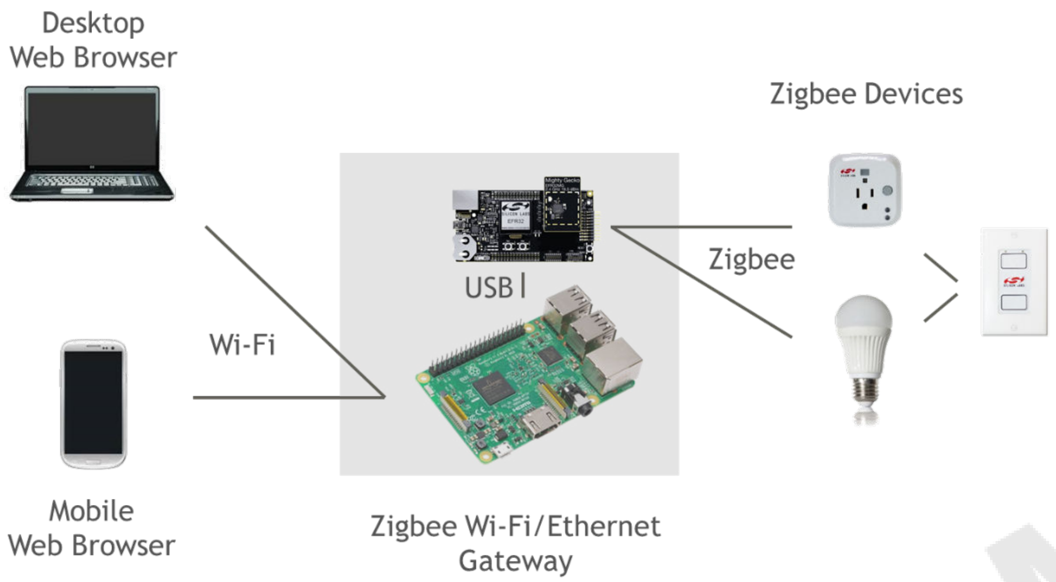


**Figure 1.1. Typical Zigbee Wi-Fi/Ethernet Gateway Configuration**

The USB Virtual Gateway runs on Ubuntu Linux 16.04 and Zigbee NCP, and is available for Windows and OSX host operating systems within a Virtualbox virtual machine. A Silicon Labs EFR32 Mighty Gecko Wireless Starter Kit such as SLWSTK6000B is required as the NCP. The gateway uses the host computer Wi-Fi client and includes a web server that presents a user interface to a desktop or mobile web browser. The web browser can run on the Virtualbox virtual machine or on a device connected via the LAN. A typical Zigbee system configuration with the Virtual Gateway is shown in the following figure.

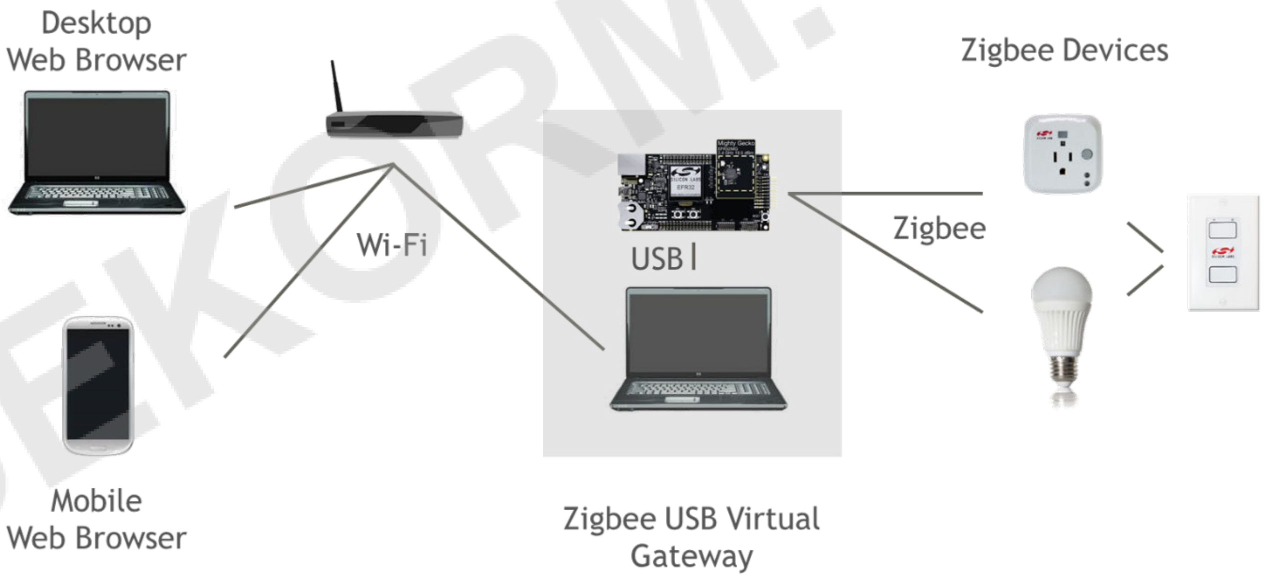


**Figure 1.2. Typical Zigbee Virtual Gateway Configuration**

# 2. Installation and Configuration for the Zigbee Wi-Fi/Ethernet Gateway (RD-0001-0201)

The Zigbee Wi-Fi/Ethernet Gateway (RD-0001-0201) runs on a Raspberry Pi computer. The following instructions describe how to order, set up, and install the Raspbian operating system and gateway software.

## 2.1 Order a Raspberry Pi

Refer to https://www.raspberrypi.org for recommended vendors for each component.

- Raspberry Pi 2 Model B or Raspberry Pi 3 Model B
- 16 GB MicroSD card
- Edimax EW-7811UN USB 2.0 Wireless Adapter (required for Raspberry Pi 2 Model B only)
- 5V 2A Power supply

## 2.2 Set Up the Raspberry Pi

1. Connect the USB Wi-Fi Adapter to one of the Raspberry Pi 2 Model B's USB ports (Raspberry Pi 3 Model B already has built-in Wi-Fi).
2. Connect the Raspberry Pi's Ethernet port to the Internet with an Ethernet cable.
3. Connect one of the Wireless Starter Kit mainboards to the Raspberry Pi with a USB cable.

   This will become the network co-processor as shown in Figure 1.1 Typical Zigbee Wi-Fi/Ethernet Gateway Configuration on page 3
4. Connect a monitor to the HDMI port and a keyboard to a free USB port.
5. Plug in RaspberryPi's power supply.

In the following steps you will power the Zigbee Gateway on and off by connecting and disconnecting its power supply. The red power LED will illuminate on the Zigbee Gateway and the green activity LED will blink until the boot process has completed. The proper procedure for power down is to issue the following command before removing power:

```
$ sudo shutdown –h now
```

## 2.3 Install the Raspbian OS and Gateway Software

1. Use a host computer to install the Raspbian Stretch Lite operating system on the SD card as described here:

   https://www.raspberrypi.org/downloads/raspbian/

   **Note:** Latest Silicon Labs Gateway 2.5.0 was verified on the 2017-11-29 Raspbian Stretch Lite image. (This archived version can be download here: http://downloads.raspberrypi.org/raspbian_lite/images/raspbian_lite-2017-12-01/)

2. Install the SD card in the Raspberry Pi and power it on.
3. Login and configure the keyboard layout.

   The default username is "pi" and password is "raspberry". On the first reboot you are prompted to change the password. Configure the default keyboard layout and optionally enable the ssh server with:

```
$ sudo raspi-conf
```

   The keyboard configuration is set with “Localization Options” and the ssh server is enabled with “Interfacing Options.”
4. Run the following commands to install the Zigbee Gateway and Wi-Fi soft access point packages:

```
$ sudo apt-get update
$ sudo apt-get install dirmngr
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 90CE4F77
$ sudo chmod 666 /etc/apt/sources.list
$ sudo echo deb http://devtools.silabs.com/solutions/apt stretch main >> /etc/apt/sources.list
$ sudo apt-get update
$ sudo apt-get install silabs-zigbee-gateway
$ sudo apt-get install silabs-networking
$ sudo reboot
```

   **Note:** If the keyserver’s port is blocked by firewall, use the hkp:// port 80 as below instead:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 90CE4F77
```

# 3. Installation and Configuration for the Zigbee USB Virtual Gateway (RD-0002-0201)

The Zigbee USB Virtual Gateway (RD-0002-0201) runs on Ubuntu Linux 16.04 operating system (OS) natively, or runs Ubuntu Linux 16.04 as a guest OS within a VirtualBox virtual machine for Windows or OSX host operating systems. The following instructions describe how to install a VirtualBox virtual machine, Linux 16.04 and the gateway software.

## 3.1 Preparing to Install

1. Download Ubuntu Linux 16.04 ISO image here: http://www.ubuntu.com/download
2. In a free USB port install the Silicon Labs EFR32 Mighty Gecko Wireless Starter Kit (SLWSTK6000B).

   If you prefer to run the Zigbee Virtual Gateway natively for Ubuntu Linux 16.04 you may skip to ahead to 3.10 Install the Gateway on Ubuntu Linux 16.04 OS.
3. Download and install VirtualBox here: https://www.virtualbox.org/wiki/Downloads. Accept requests to install drivers.

## 3.2 Create a Virtual Machine

1. Launch VirtualBox.
2. On the VirtualBox Manager menu, click [**New**].
3. Create a virtual machine with the following settings:
   a. Select Name: Ubuntu Linux 16.04
   b. Select Type: Linux.
   c. Select Version: Ubuntu (64-bit) or Ubuntu (32-bit) depending on your system.
   d. Select memory size to at least 1024 MB and click [**Create**].
4. Create a virtual hard disk.
   a. Select **Create a virtual hard disk now** and click [**Next**].
   b. Select **VDI** and click [**Next**].
   c. Select **Dynamically allocated** and click [**Next**].
   d. Select **25 GB or greater** size and click [**Create**].

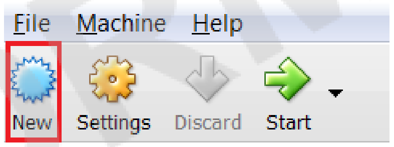


**Figure 3.1. VirtualBox Manager Menu**

### 3.3 Configure the Network

1. Select the virtual machine.
2. On the VirtualBox Manager menu, click [**Settings**].
3. In the Settings window, select **Network**.
4. Check **Enable Network Adapter**.
5. For **Attached to:** select **Bridged Adapter**.
6. For **Name:** select the host adapter.
7. Click [**OK**].

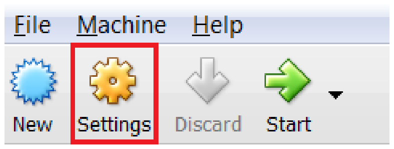
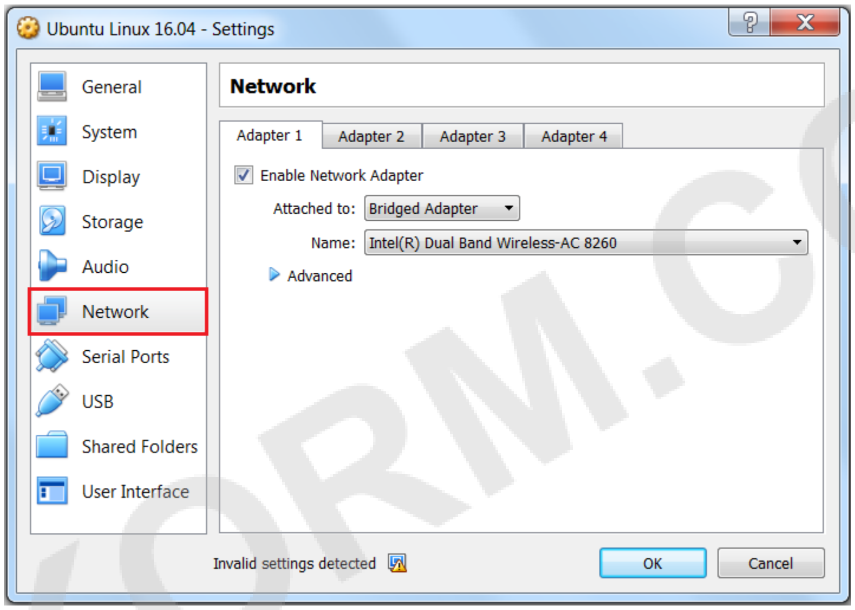


**Figure 3.2. VirtualBox Manager Settings**

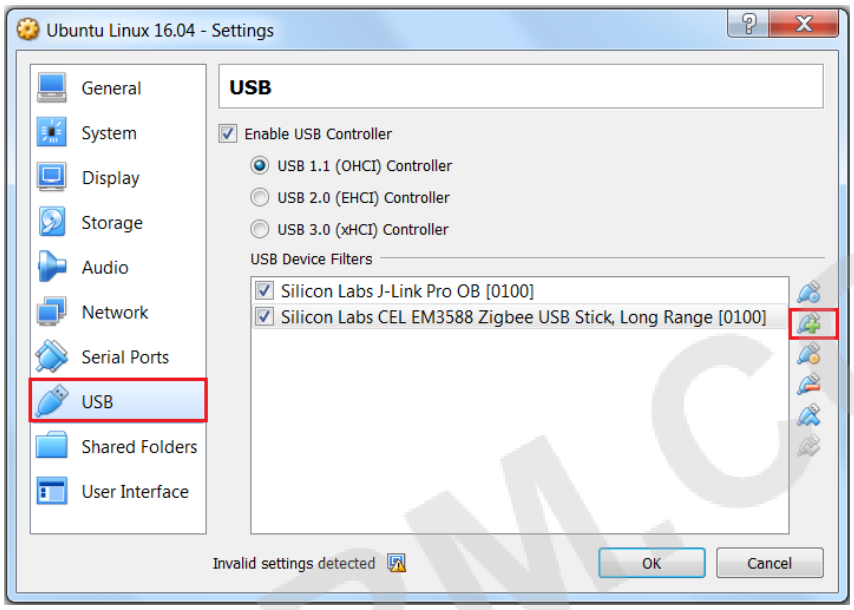


☑ Enable Network Adapter
Attached to: Bridged Adapter
Name: Intel(R) Dual Band Wireless-AC 8260

**Figure 3.3. Network Settings**

### 3.4 Configure USB Devices

1. Select the virtual machine.
2. On the VirtualBox Manager menu, click [**Settings**].
3. In the Settings window, select **USB**.
4. Insert the **Silicon Labs J-Link PRO OB** of the EFR32 Mighty Gecko Wireless Starter Kit (SLWSTK6000B) or the **Silicon Labs CEL EM3588 Zigbee USB stick** into the USB port of the virtual machine host PC.
5. Click the **Add new filter** icon on the right of the dialog box.
6. Add and check the corresponding NCP to let the virtual machine access that USB NCP device. Note that both NCPs are shown in the following image for demonstration purposes. Your system should only have one of these devices selected.
7. Click [**OK**].



[✓] Enable USB Controller
[● USB 1.1 (OHCI) Controller]
○ USB 2.0 (EHCI) Controller
○ USB 3.0 (xHCI) Controller

USB Device Filters
[✓] Silicon Labs J-Link Pro OB [0100]
[✓] Silicon Labs CEL EM3588 Zigbee USB Stick, Long Range [0100]

Invalid settings detected

Figure 3.4. USB Settings

### 3.5 Install Ubuntu 16.04

1. Select the virtual machine.
2. On the VirtualBox Manager menu, click [**Settings**].
3. In the Settings window, select **Storage**.
4. Select **Controller IDE**.
5. Click the **Add Optical Drive** icon.
6. Select the Ubuntu Linux 16.04 ISO image.
7. Click [**OK**].
8. On the VirtualBox Manager menu, click [**Start**].
9. Select **Install Ubuntu** and follow installation instructions.

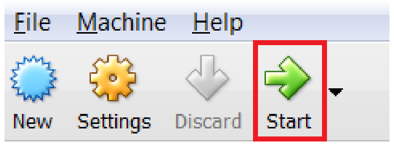
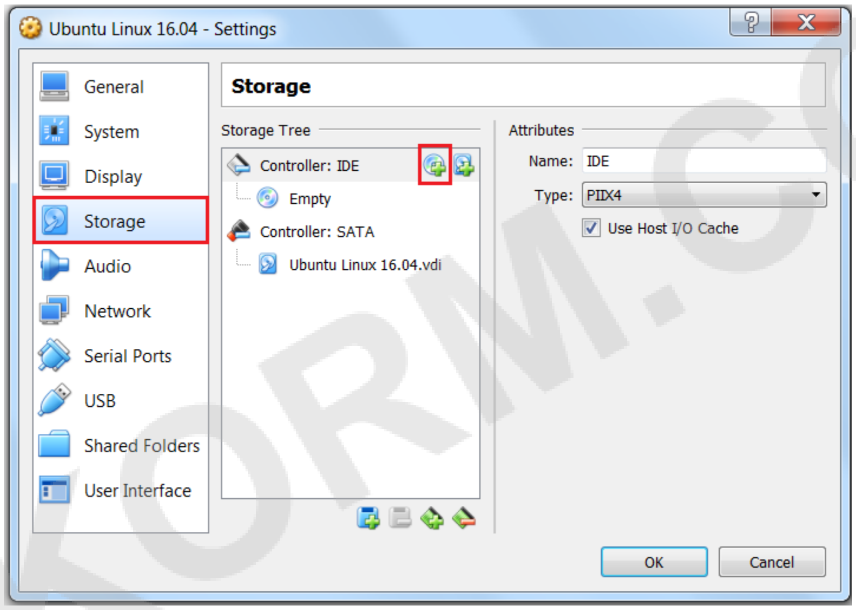


**Figure 3.5. VirtualBox Manager Start**



Name: IDE
Type: PIIX4
Use Host I/O Cache ✓

**Figure 3.6. Storage Settings**

### 3.6 Configure Guest Additions (Optional)

1. From the virtual machine menu, select **Devices >> Insert Guest Additions** and follow the installation instructions.
2. Eject the guest additions CD.

### 3.7 Configure Host / Guest File Sharing (Optional)

1. Select the virtual machine.
2. On the VirtualBox Manager menu, click [**Settings**].
3. Click **Shared Folders** and point to a shared folder on the host OS drive.
4. Click [**Automount**].
5. Click [**Make Permanent**].
6. Click [**OK**] twice.
7. From the virtual machine open a terminal and enter the following:

```
$ sudo usermod -a -G vboxsf <username>
$ sudo reboot
```

The shared drive will be found in the /media directory in the virtual machine.

### 3.8 Verify the Network is Connected

1. Confirm the network connection icon is present as shown in the following figure.

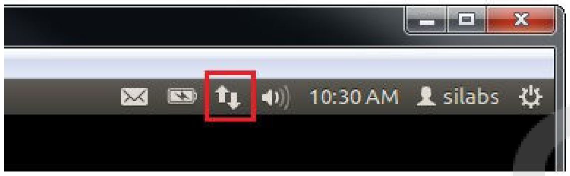


**Figure 3.7. Verify Network Connection**

### 3.9 Verify the USB NCP is Captured

1. From the VirtualBox Manager menu, select **Devices >> USB**.
2. Confirm the desired USB device is captured (captured devices are indicated with a check) as shown in the following figure. Note that both NCP options are shown for demonstration purposes. Your system should only have one of the options selected.

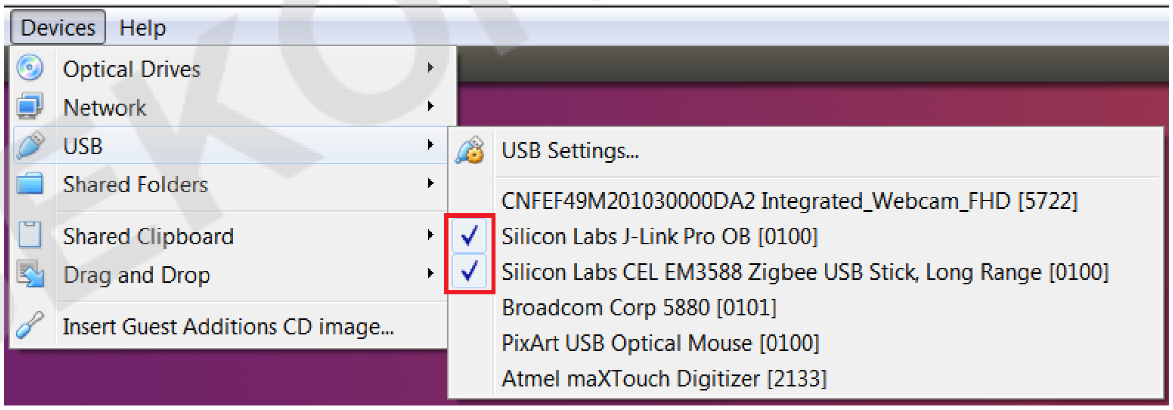

Devices
Help
Optical Drives
Network
USB
Shared Folders
Shared Clipboard
Drag and Drop
Insert Guest Additions CD image...
USB Settings...
CNFEF49M201030000DA2 Integrated_Webcam_FHD [5722]
✓ Silicon Labs J-Link Pro OB [0100]
✓ Silicon Labs CEL EM3588 Zigbee USB Stick, Long Range [0100]
Broadcom Corp 5880 [0101]
PixArt USB Optical Mouse [0100]
Atmel maXTouch Digitizer [2133]

**Figure 3.8. Verify USB NCP Device is Captured**

## 3.10 Install the Gateway on Ubuntu Linux 16.04 OS

Follow these instructions if you have purchased the Zigbee Virtual Gateway RD-0002-0201.

1. Run the following commands to install the Zigbee Gateway.

```
$ sudo add-apt-repository http://devtools.silabs.com/solutions/apt
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 90CE4F77
$ sudo apt-get update
$ sudo apt-get install silabs-zigbee-gateway
$ sudo reboot
```

**Note:** If the `silabs-zigbee-gateway` installation fails due to dependency on nodejs >= 5.12.0, follow this procedure to install a higher version, such as nodejs 8.10.0 LTS, and then re-run `sudo apt-get install silabs-zigbee-gateway`.

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install nodejs
```

# 4. Install the USB Zigbee Adapter NCP Software

The NCP software can be installed or updated in two ways. The first method is to update the NCP with Simplicity Studio. The second method is to update the NCP from the console if a bootloader has been installed.

## 4.1 Flashing from Simplicity Studio

Pre-compiled Zigbee NCP firmware and bootloader files are distributed with the Zigbee Gateway file system. These files may be transferred from the gateway to a host with a utility such as WinSCP or scp. The NCP programming files are available in the Zigbee Gateway file system here:

| Device | NCP Programming File |
|---|---|
| EFR32 | /opt/siliconlabs/zigbeegateway/firmware/ncp-uart/efr32mg12p432f1024gl125/ncp-uart-hw.s37 |

The bootloader files are available in the Zigbee Gateway file system here:

| Device | First and Second Stage Bootloader Files |
|---|---|
| EFR32 | /opt/siliconlabs/zigbeegateway/firmware/ncp-uart/efr32mg12p432f1024gl125/first_stage_btl_efx32xg12.s37<br><br>/opt/siliconlabs/zigbeegateway/firmware/ncp-uart/efr32mg12p432f1024gl125/bootloader-uart-xmodem-efr32mg12p432f1024gl125-brd4161a.s37 |

Information on how to flash the NCP firmware onto the appropriate chipset can be found in *QSG106: Getting Started with EmberZNet PRO*. **IMPORTANT:** The device should be erased before programming.

## 4.2 Flashing from the Console

The NCP software may be updated from the console if a bootloader is installed.

Stop the Zigbee Gateway applications, scan to determine the NCP port, and flash the NCP software. Be certain to specify the correct port for the flash operation (/dev/ttyACM0 is used as an example below) as returned by the scan function. There will be no visible indication while firmware is updated.

```
$ sudo service siliconlabsgateway stop
$ cd /opt/siliconlabs/zigbeegateway/
$ sudo python tools/ncp-updater/ncp.py scan
$ sudo python tools/ncp-updater/ncp.py flash -p /dev/ttyACM0 –f firmware/ncp-uart/efr32mg12p432f1024gl125/*.gbl
$ sudo reboot
```

## 5. Run the Gateway

If using the Zigbee Wi-Fi/Ethernet Gateway (RD-0001-0201), power-up the gateway hardware, use a mobile phone or laptop and connect to the Wi-Fi SSID "Silicon Labs xxxx" using the password "solutions" as shown in the following figure. Then, open a web browser and browse to "192.168.42.1" to access the gateway user interface. If using the Zigbee Virtual USB Gateway (RD-0002-0201), open Virtual Box and start the Ubuntu Virtual Machine. Then, open the web browser and browse to "localhost" to access the gateway user interface.

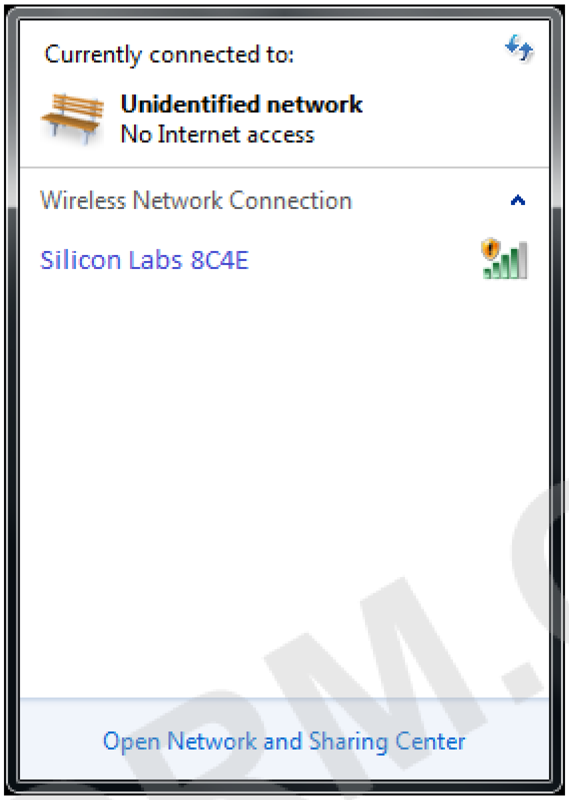


**Figure 5.1. Wireless Network Connections (Windows)**

The following sections describe the gateway user interface tabs and their functions.

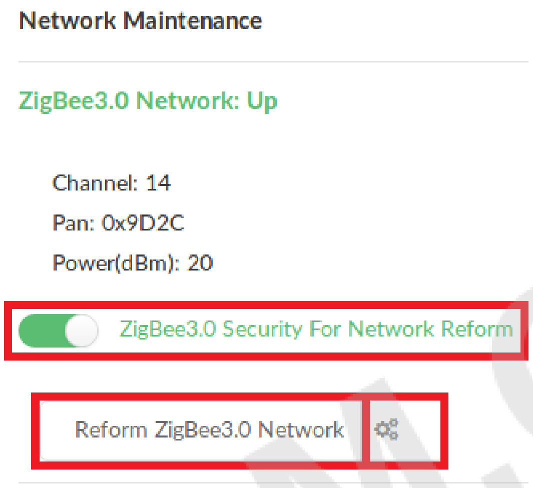
### 5.1 Setup

In the Network Maintenance section of the Setup tab, confirm that “ZigBee 3.0 Network: Up” is shown, and if not, refer to Section 8. Troubleshooting for possible solutions. On first boot by default the PAN ID is randomly assigned, a clear channel is auto-selected after channel scanning, and the power is set to 20 dBm. All settings are saved and restored on the next boot. All settings are saved and restored on the next boot.

The PAN ID is a 16-bit number expressed in hexadecimal format, the channel can be set to any valid Zigbee channel (11-26), and the valid power level range is –20 dBm to 20 dBm. Note that range checking is enforced.

Tapping the **Reform Zigbee 3.0 Network** button as shown in the following figure will re-initialize the Zigbee 3.0 network similar to the first boot. Note that all existing joined devices will be cleared.

Tapping the **Extended Network Form Settings** icon next to **Reform ZigBee 3.0 Network** allows you to set specific PAN ID, channel, and power levels explicitly, then reinitialize the Zigbee 3.0 network. All existing joined devices will be cleared.


Network Maintenance

ZigBee3.0 Network: Up

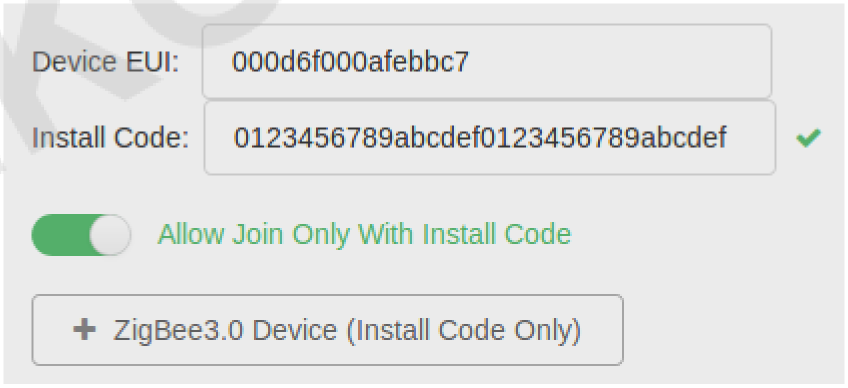
Channel: 14

Pan: 0x9D2C

Power(dBm): 20

**Figure 5.2. Network Maintenance**

Zigbee 3.0 devices should join the Zigbee 3.0 network using install codes, or optionally using the default global Trust Center link key. To enable a Zigbee 3.0 device joining only with an install code, enter the corresponding 16-hexadecimal-digit device EUI and the 16-byte (32-hexadecimal-digit) install code in the input boxes shown in the following figure. Turn on the **Allow Join Only With Install Code** option and then tap [**+ ZigBee 3.0 Device (Install Code Only)**] to open the network for joining the desired device.


Device EUI: 000d6f000afebbc7

Install Code: 0123456789abcdef0123456789abcdef

Allow Join Only With Install Code
+ ZigBee3.0 Device (Install Code Only)

**Figure 5.3. Install Code**

To join Zigbee devices using the default global Trust Center link key, simply leave the Device EUI and Install Code input boxes blank and tap [**+ ZigBee 3.0 Device**] to open the network for the active joining devices.

When a Zigbee HA device is joined on a Zigbee 3.0 network, the device will be allowed to join for 180 seconds before being sent a leave request.

When using Zigbee 3.0 install codes the device EUI and install code entered must match the EUI and install code of the desired device. The EUI is an 8-byte (16-digit number) expressed in hexadecimal format. The install code is 16-byte (32-digit number) expressed in hexadecimal format. Refer to the following for additional information on creating and flashing Silicon Labs reference designs with Zigbee 3.0 installation codes:

- http://community.silabs.com/t5/Mesh-Knowledge-Base/Z3-network-join-with-install-code-derived-link-key/ta-p/191924
- http://www.silabs.com/documents/public/application-notes/AN714-SmartEnergyECCEnabledDeviceSetupProcess.pdf

To enter join mode for the Silicon Labs lighting reference design press S1 ten times rapidly, for the contact sensor press S1 for more than one second, for the dimmer switch press S3 for more than one second, for the occupancy sensor press S1 for more than one second, and for the outlet press the front button for more than three seconds. Additional information can be found in the user's guide for each device. Devices will appear in the list with their name, endpoint, unique node ID, and state. The name shown is determined by the Zigbee device ID of the endpoint(s) reported by each device and the unique node ID is assigned each time the device joins a Zigbee network. For a device supporting multiple-endpoints, each endpoint will appear as an individual entry in the device list with corresponding name and endpoint.

If a device is on a network and communicating with the gateway, its state will be labeled as “joined”. A device failing to respond will be labeled “unresponsive”. To request a device to leave the network, select the “X” next to the device. The device will be labeled "leave sent" if there is no response from the device. Devices may become unresponsive or indicate leave sent because they are asleep, turned off, or out of range. When the device wakes, turns on, or comes back into range, the unresponsive device will be labeled as “joined” and a device labeled “leave sent” will be removed from the device list.

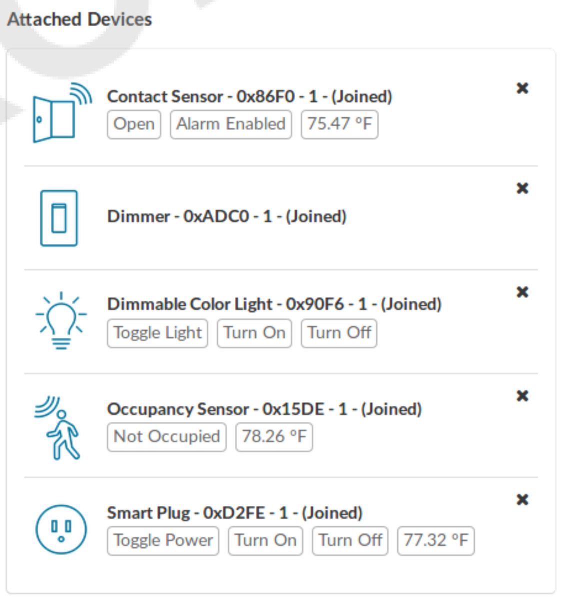
*RD-0030-0201: Contact Sensor Reference Design* will indicate open/close state, active/alarm state, temperature, and the join/leave-sent/unresponsive state. The open/close state is sent by the contact sensor immediately upon change of state to indicate whether the magnet is away (open) or near (closed) the reed switch. The alarm state is sent by the contact sensor immediately upon change of state when the tamper alarm is activated by pressing button S1 for more than four seconds and then releasing.

*RD-0020-0601, RD-0035-0601, RD-0085-0401, and RD-0098-0401: Lighting Demo Board Reference Designs* will present the Toggle Light, Turn On and Turn Off buttons to change the on/off state of the light and indicate the join/leave-sent/unresponsive state. The Toggle Light button sends the ZCL on-off toggle command while the Turn On/Off buttons send the ZCL on-off on and ZCL on-off off command respectively.

*RD-0039-0201: Capacitive Sense Dimmer Switch Reference Design* will show the joined/leave sent/unresponsive state.

*RD-0078-0201 and RD-0099-0201: Occupancy Sensor Reference Design* will indicate occupied or not occupied state, temperature, and the join/leave-sent/unresponsive state.

*RD-0051-0201 and RD-0100-0201: Smart Outlet Reference Design* will present Toggle Power, Turn On, and Turn Off buttons to change the on/off state of the outlet and indicate the join/leave-sent/unresponsive state. The Toggle Power button sends the ZCL on-off toggle command while the Turn On/Off buttons send the ZCL on-off on and ZCL on-off off command respectively.


Attached Devices

Contact Sensor - 0x86F0 - 1 - (Joined)
Open | Alarm Enabled | 75.47 °F

Dimmer - 0xADC0 - 1 - (Joined)

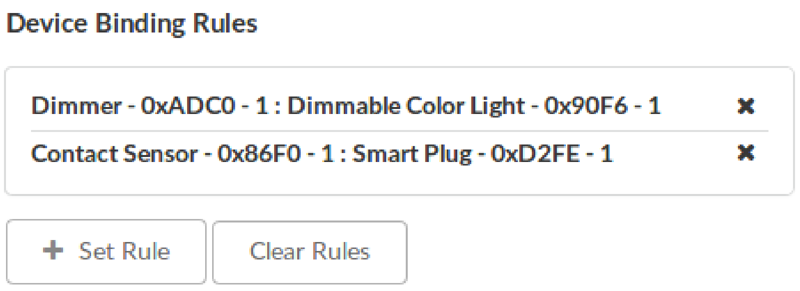
Dimmable Color Light - 0x90F6 - 1 - (Joined)
Toggle Light | Turn On | Turn Off

Occupancy Sensor - 0x15DE - 1 - (Joined)
Not Occupied | 78.26 °F

Smart Plug - 0xD2FE - 1 - (Joined)
Toggle Power | Turn On | Turn Off | 77.32 °F

**Figure 5.4. Attached Devices**

The gateway allows the user to create rules to bind one device to another. To create a rule, click [**+ Set Rule**], choose the desired input node and output node, and click [**Bind**]. Multiple rules can be set for both input nodes and output nodes. If two input nodes send a command to an output node, the commands are executed in the order received. Rules can be either individually removed with the "X" next to the device, or all rules can be removed by clicking [**Clear Rules**].



Dimmer - 0xADC0 - 1 : Dimmable Color Light - 0x90F6 - 1

Contact Sensor - 0x86F0 - 1 : Smart Plug - 0xD2FE - 1

**Figure 5.5. Device Binding Rules**

## 5.2 Home

The home tab duplicates the setup information and offers extended information with the [**Enter Identify Mode**] and the [**Show Extended Info**] buttons. A dimmable color light adds brightness, on/off, color temperature, and hue/saturation controls, an occupancy sensor adds light intensity and relative humidity, and an outlet adds on/off, power used, light intensity, relative humidity, temperature, RMS voltage, RMS current, and active power.

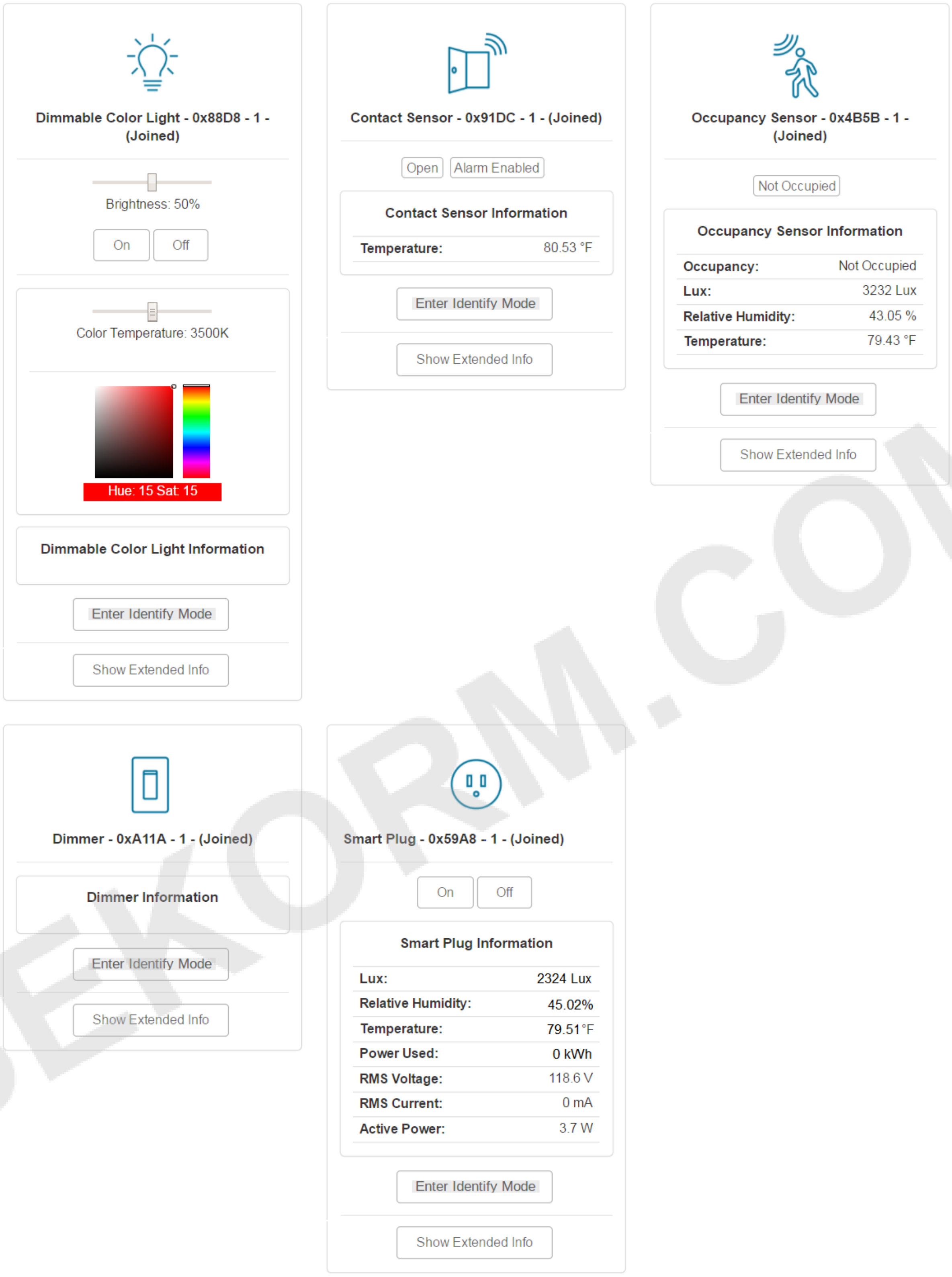
Tap the [**Enter Identify Mode**] button to command the device to enter identify mode with device-specific visual indication for 3 minutes. The button label will be toggled to [**Exit Identify Mode** ]. Tap the button again to exit the identify mode. The consequent behavior may not be triggered immediately during sleeping of a sleepy end-device.

The extended information includes:

- Node EUI
- Endpoint
- Gateway EUI
- Node State (joined, leave sent, unresponsive)
- Firmware Version
- Firmware Image type
- Manufacturer ID
- Device Type
- OTA Bytes Sent
- Updating Indicator (via OTA)
- Available OTA images list

Available OTA update images are located here: `/opt/siliconlabs/zigbeegateway/ota_staging`

**Note:** The OTA update process takes approximately ten minutes for non-sleepy devices and up to several hours for sleepy devices. Only one device should be in the "Attached Device" list before beginning the OTA update process.



Contact Sensor Information

| | |
|---|---|
| Temperature: | 80.53 °F |

Occupancy Sensor Information

| | |
|---|---|
| Occupancy: | Not Occupied |
| Lux: | 3232 Lux |
| Relative Humidity: | 43.05 % |
| Temperature: | 79.43 °F |

Smart Plug Information

| | |
|---|---|
| Lux: | 2324 Lux |
| Relative Humidity: | 45.02% |
| Temperature: | 79.51°F |
| Power Used: | 0 kWh |
| RMS Voltage: | 118.6 V |
| RMS Current: | 0 mA |
| Active Power: | 3.7 W |

**Figure 5.6. Home Tab**

### 5.3 Diagnostics

The diagnostics tab offers advanced logging options.

The server log tab displays all web server command routing. The gateway output tab shows all Zigbee gateway commands and data.

The “Console Log Streaming” option enables corresponding log updates to the server log tab and gateway log tab in additional to log file background save functionality.

In typical use, displaying this logging information is not necessary, and disabling this option reduces gateway traffic and overhead. The **Input CLI Command…** inbox box below the gateway output tab can also be used to send command line interface (CLI) commands.

The server log file is located here: `/opt/siliconlabs/zigbeegateway/logs/server.log`

The gateway output log file is located here: `/opt/siliconlabs/zigbeegateway/logs/gateway.log`

### 5.4 About

The About tab shows all versions and displays the web server IPv4 address for the purpose of connecting a mobile handset, tablet, or another computer to the gateway. It also displays the version of the installed gateway application as well as the NCP firmware version.

**Note:** The “Running on IP” address is updated when refreshing the browser window.

### 5.5 Shutdown

To properly close the gateway and virtual gateway, issue the following command at the terminal:

```
$ sudo shutdown –h now
```

# 6. Software Components

This section contains:

- An overview of the Zigbee gateway software architecture
- How to obtain, compile and run the software components of the gateway
- Details of the application interfaces (APIs) between key software components

The gateway reference design developed by Silicon Labs is designed to demonstrate Zigbee gateway functionality with several Silicon Labs Zigbee reference designs. The software platform is designed in a manner that customers can leverage to develop their own gateway applications with minimal customization.

The gateway software has the following architecture:

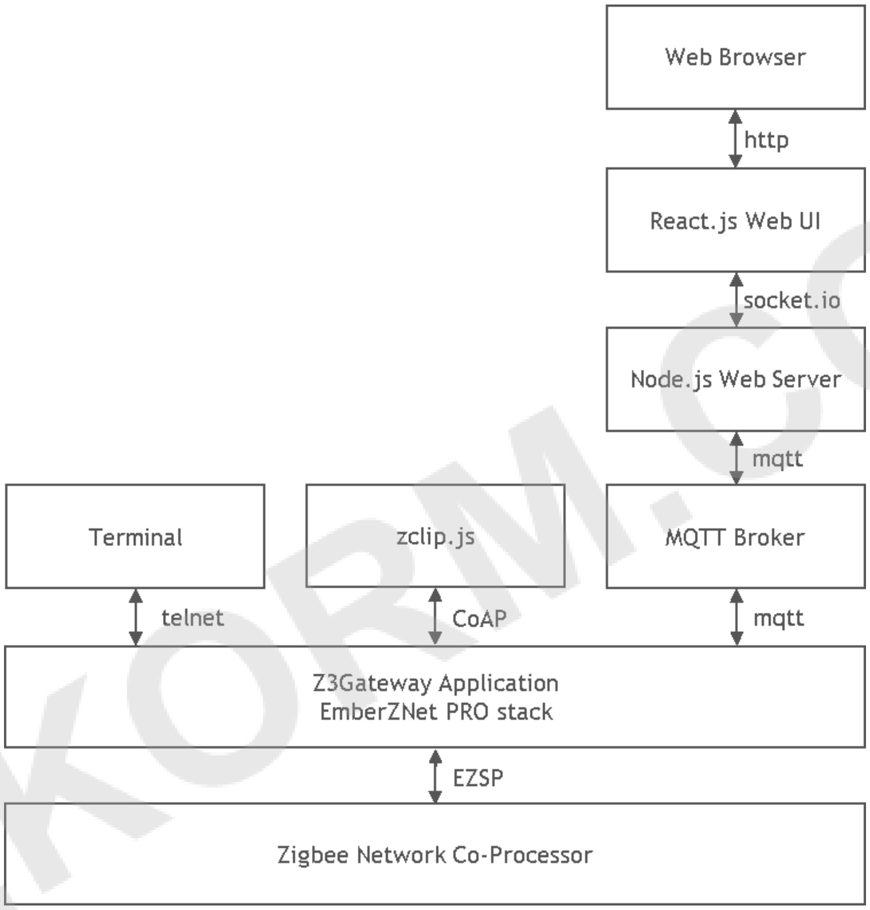


**Figure 6.1. Software Application Diagram**

## 6.1 Getting Started

The software is distributed as three software components:

- Z3Gateway Application
- Node Server Application
- ReactJS Front-End Application

The software binaries are available using the Linux apt package manager, as described in 2. Installation and Configuration for the Zigbee Wi-Fi/Ethernet Gateway (RD-0001-0201) and 3. Installation and Configuration for the Zigbee USB Virtual Gateway (RD-0002-0201).

The software source code for the Z3Gateway application is available as a host example application distributed with the EmberZNet PRO Zigbee stack. For more information on how access the Zigbee stack please see http://www.silabs.com/products/wireless/mesh-networking/Pages/getting-started-with-mighty-gecko-zigbee.aspx.

The software source code for the Node Server Application and ReactJS Front-End Application is available on github at https://github.com/SiliconLabs/gateway-management-ui and is also installed on the target system at /opt/siliconlabs/zigbeegateway/gateway-management-ui.

### 6.1.1 Set Up the Z3Gateway Application and MQTT Broker

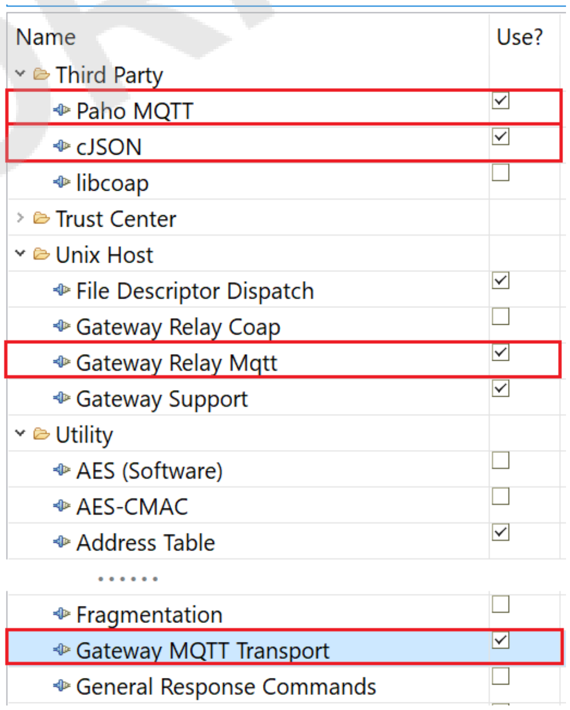
The following describes how to build and launch the Z3Gateway application in a Linux build and execution environment.

These setup instructions assume:

- Simplicity Studio Version 4.1.6 or later has been installed
- Within Simplicity Studio, Gecko SDK Suite 2.2 and EmberZNet 6.2.0.0 (or above) is installed

Perform the following steps:

1. Use Simplicity Studio's Simplicity IDE to generate a gateway project to build.
   a. Open Simplicity Studio and verify the SDK installation:
      i. Click the gear icon along the top left corner, or Preferences in the menu tab.
      ii. In the settings dialog box, select Simplicity Studio → SDKs.
      iii. Verify that Gecko SDK Suite: EmberZNet 6.2.0.0 is installed. If not, refer to *QSG106: Getting Started with EmberZNet PRO* for installation instructions. Note: Depending on your installation you may see other SDKs along with EmberZNet.
   b. From the Tools menu, open the Simplicity IDE.
   c. Select File menu → New → Project.
   d. In the New Project dialog, select **Silicon Labs AppBuilder Project** and click [**Next**].
   e. Select **ZCL Application Framework V2**, and click [**Next**].
   f. Select **EmberZNet 6.2.0.0 GA Host 6.2.0.0**, and click [**Next**].
   g. Select **Z3Gateway**, and click [**Next**].
   h. Rename your project (if desired) and click [**Next**].
   i. Ensure **Part** input box **None** has been selected. If not, click its right down-arrow button to select "Node (Compatible)" from the drop-down menu. Click [**Finish**].
   j. Under the General tab, ensure the directory for generated files directory is pointed at this location: C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.2.
   k. The gateway could be executed with or without communication to an MQTT broker. If MQTT is required, some additional plugins (shown in the following figure) need to be enabled on top of the default. Under the Plugins tab, click the checkboxes of "Paho MQTT", "cJSON", "Gateway Relay Mqtt", and "Gateway MQTT transport" to enable these additional plugins. Two plugins, "device-table" and "command-relay," will be enabled automatically when "Gateway Relay Mqtt" is enabled.



| Name | Use? |
|---|---|
| Third Party | |
| Paho MQTT | ☑ |
| cJSON | ☑ |
| libcoap | ☐ |
| Trust Center | |
| Unix Host | |
| File Descriptor Dispatch | ☑ |
| Gateway Relay Coap | ☐ |
| Gateway Relay Mqtt | ☑ |
| Gateway Support | ☑ |
| Utility | |
| AES (Software) | ☐ |
| AES-CMAC | ☐ |
| Address Table | ☑ |
| ...... | |
| Fragmentation | ☐ |
| Gateway MQTT Transport | ☑ |
| General Response Commands | ☐ |

**Figure 6.2. Additional Plugins for an MQTT Connection**

   l. Click [**Generate**].
2. Run make to build the executable binary image:
   a. In a terminal shell window browse to the `sdks/gecko_sdk_suite/v2.2/app/builder/Z3GatewayHost/` directory.

   b. Type `make` to build the gateway.

3. Launch the Z3Gateway application.
   a. In terminal shell window browse to the `sdks/gecko_sdk_suite/v2.2/app/builder/Z3GatewayHost/build/exe` directory
   b. For the EFR32 Mighty Gecko Wireless Starter Kit:

```
./Z3GatewayHost -n 0 -p /dev/ttyACM0
```

   For the CEL MeshConnect NCP:

```
 ./Z3GatewayHost -n 1 -p /dev/ttyUSB0
```

   c. The Z3Gateway application will publish and subscribe to MQTT messages.

### 6.1.2 Set Up the Node Server Application

The following describes how to install, build and launch the node server application. The source for the Node Server Application is located on the target system at `/opt/siliconlabs/zigbeegateway/gateway-management-ui/nodeserver`.

1. Open a terminal window in the source directory.
2. Build the project:

```
$ sudo npm install
```

3. Launch the application. If running this application on the same platform as the Z3Gateway application:

```
$ sudo npm run local
```

   Otherwise if the Z3Gateway and this application are on different platforms:

```
$ sudo npm run cloud
```

### 6.1.3 Set Up the React Front-end Application

The following describes how to build and launch the ReactJS front-end application. The source for the Node Server Application is located on the target system at /opt/siliconlabs/zigbeegateway/gateway-management-ui/reactui.

1. Open a terminal window in the source directory.
2. Build the project:

```
$ sudo npm install gulp
$ sudo npm install
$ sudo npm run build
```

3. Launch the static server. Note this is only required if the Node Server is launched with "cloud" option. A static server is launched when the Node Server is launched with the "local" option. This application should reside on the same platform as the Node Server.

```
$ sudo npm run server
```

### 6.1.4 Set Up the MQTT Broker

This reference design uses Mosquitto MQTT broker. Visit http://mosquitto.org and install and launch the broker that corresponds to the target platform (ex: Ubuntu).

### 6.1.5 Final Steps

1. Open a web browser on the same machine, browse to this address: `http://localhost`.
2. The gateway application will render in the browser.

## 6.2 Z3 Gateway Application with CoAP

### 6.2.1 Overview

Three plugins are required to support the ZCL/IP (Zigbee Cluster Library over IP) protocol over CoAP from a Zigbee gateway. These plugins collectively expose Zigbee devices connected to the Z3 gateway as CoAP ports for two-way communication with a ZCL/IP client in the cloud, and provide translation between the ZCL/IP protocol over CoAP and the ZCL protocol on Zigbee. The goal is to provide parity with our current Silicon Labs Thread offering.

### 6.2.2 Architecture Overview

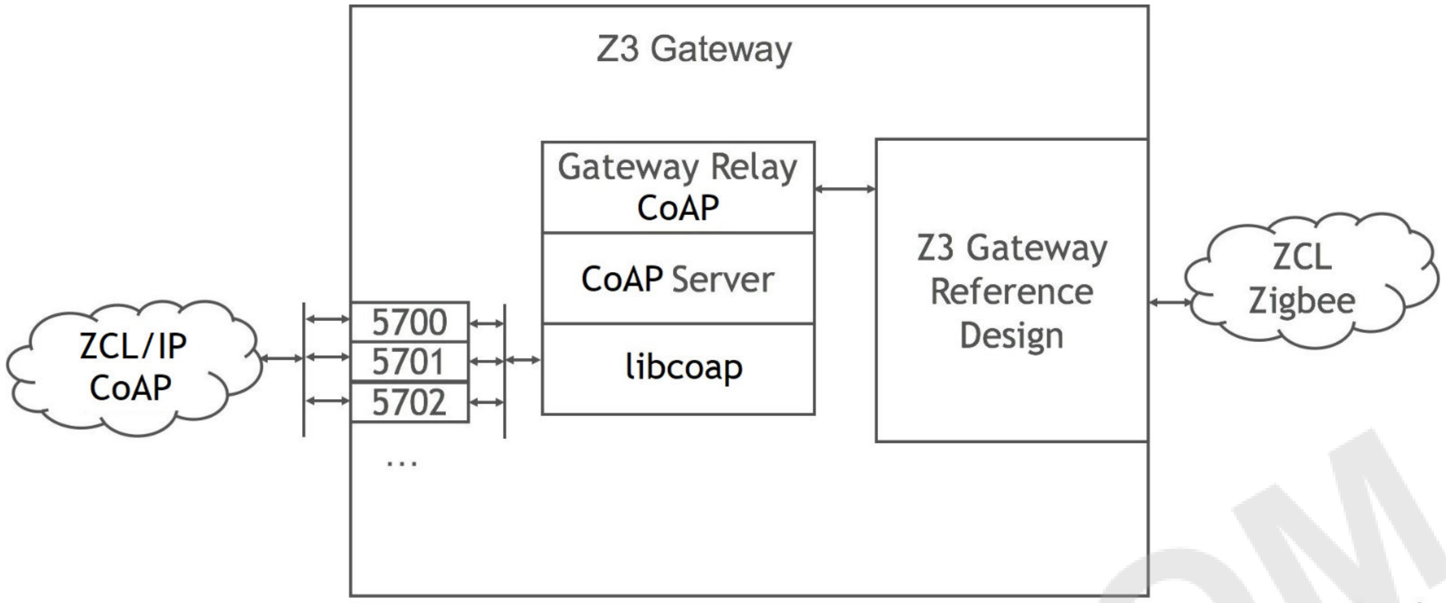


**Figure 6.3. CoAP Architecture**

Currently, the Z3 Gateway Reference Design provides a Zigbee host application that allows multiple Zigbee devices to be connected to it and be controlled by it. However, the Z3 Gateway Reference Design does not include any connectivity off network by default.

One option for off network connectivity is to leverage the ZCL/IP application layer with the CoAP transport. Three different plugins are required to enable this functionality in the Z3 Gateway: libcoap, CoAP Server, and Gateway Relay CoAP.

**Libcoap** is a snapshot of the publically available libcoap project configured for use with a Linux or similar operating system, with one modification. This modification allows for partial mapping of URI before calling the method callbacks. This facilitates its use with the ZCL/IP protocol where the URI structures are the protocol itself.

**CoAP Server** is a plugin that sits between libcoap and the ZCL/IP translation layers. CoAP Server opens one CoAP channel for each Zigbee device and map this port to the respective Zigbee device on the network. In this way, each device in the Zigbee network has its own unique connection to the outside world. This mirrors what we would expect to see with a Thread network, where each Thread device is individually addressable from the outside world.

Incoming messages from the CoAP network are sent to the **Gateway Relay CoAP** plugin for translation and eventual forwarding to the correct Zigbee device. Incoming messages and responses from the Zigbee network can be forwarded to a client device specified by the Gateway Relay Coap plugin.

**Note:** The CoAP Server code is provided as an alpha test tool. Because it does not implement any security layer, it is not recommended for use in a final product in its current form, as that would be an obvious security risk. Future versions of the plugin will employ security.

The Gateway Relay CoAP plugin provides translation between the ZCL/IP and ZCL application layers. It leverages the underlying network stacks to provide connectivity. By design, ZCL/IP messages have a natural translation to ZCL messages, and the translation is straightforward. In a few notable situations the ZCL/IP translation does not behave exactly as a Thread device would. This includes any situation where the ZCL/IP translation plugin requires communicating with a Zigbee device to obtain data (such as a read attribute request). In these cases, the ZCL/IP plugin responds to the initial message with an empty ack, and it then forwards the response as a separate, translated message.

The Gateway Relay CoAP plugin includes a simple CBOR encoder/decoder that is limited to the functionality required for the ZCL/IP communication currently implemented.

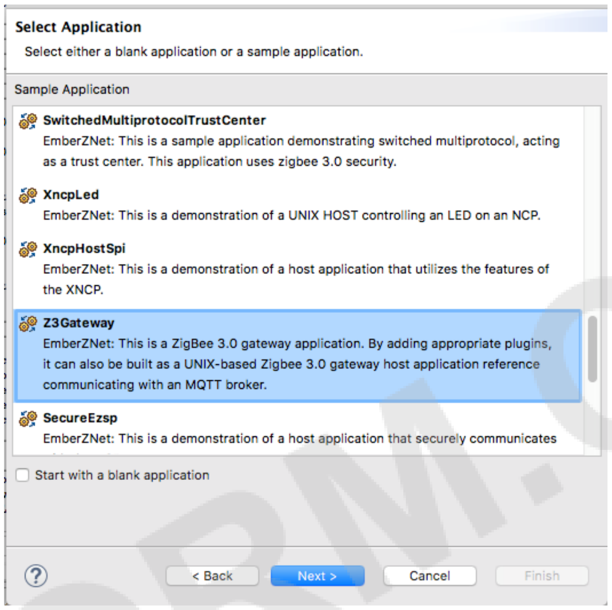
The Gateway Relay CoAP plugin includes a simple CLI interface for setting the client IP address, as well as starting and stopping the heartbeat messages.

| CLI Command | Description |
|---|---|
| plugin gateway-relay-coap set-server <ip addr> | Set the IP address of the client to which the gateway will relay all outgoing messages. <ip addr> is an IP address string of the format “192.168.2.1” |

| CLI Command | Description |
|---|---|
| plugin gateway-relay-coap hbstart <period> | Start the heartbeat. <period> is a two-byte integer and is in seconds. |
| plugin gateway-relay-coap hbstop | Stop the heartbeat. |

### 6.2.3 Building CoAP into a Gateway Application

three plugins are designed to be added to the Z3 Gateway and compiled under a Linux environment. To build a gateway with these CoAP plugins, start by building a Z3 Gateway (File->New->Project, then select Silicon Labs AppBuilder Project). After selecting ZCL Application Framework (V2), and selecting Host applications, you will see the sample application dialog as shown in the following figure.



**SwitchedMultiprotocolTrustCenter**
EmberZNet: This is a sample application demonstrating switched multiprotocol, acting as a trust center. This application uses zigbee 3.0 security.

**XncpLed**
EmberZNet: This is a demonstration of a UNIX HOST controlling an LED on an NCP.

**XncpHostSpi**
EmberZNet: This is a demonstration of a host application that utilizes the features of the XNCP.

**Z3Gateway**
EmberZNet: This is a ZigBee 3.0 gateway application. By adding appropriate plugins, it can also be built as a UNIX-based Zigbee 3.0 gateway host application reference communicating with an MQTT broker.

**SecureEzsp**
EmberZNet: This is a demonstration of a host application that securely communicates

Figure 6.4. Building a Gateway with CoAP

Select the Z3 Gateway and click [**Finish**] to create the project. To enable the CoAP functionality, select the Plugins tab and search for the keyword “coap” to find the three CoAP-related plugins.

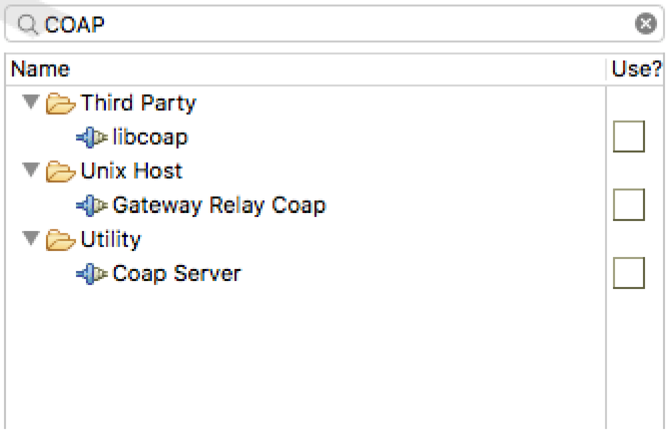


Figure 6.5. Adding CoAP Plugins

You must select all three of these plugins in order to enable the CoAP related functionality in your Zigbee gateway.

### 6.2.4 Controlling Zigbee Devices with zclip.js

Zclip.js is a ZCL-over-IP implementation written in JavaScript that is available at https://github.com/SiliconLabs/zclip.js. It provides a library useful for interacting with devices, scripting, and off-mesh development platforms like cloud and mobile.

Zclip-cli adds a command line wrapper around zclip.js. It comes with a CLI interface, facilitating the following examples. More information about this library can be found at *QSG102: Thread Border Router Add-On Kit Quick Start Guide*. This CLI utility is available at https://github.com/SiliconLabs/zclip-cli.

These examples assume you are connected to the Z3 gateway enabled with the CoAP functionality and a device containing an On-Off Cluster and Level Control Cluster such as the Z3ColorControlLight has been added to its Zigbee network.

1. Turn the device on and off.

```
$ zcl onOffCluster on 192.168.3.40 –port 5700
$ zcl onOffCluster off 192.168.3.40 –port 5700
```

2. Set the device level.

```
$ zcl levelControlCluster moveToLevel 192.168.3.40 –port 5700 --level 0 --transitionTime 0
$ zcl levelControlCluster moveToLevel 192.168.3.40 –port 5700 --level 255 --transitionTime 0
```

## 6.3 Gateway Application with MQTT

### 6.3.1 Overview

**Table 6.1. Plugins Supporting the Z3Gateway Application**

| Plugin Name | Description |
|---|---|
| Device Table | Tracks devices as they join the network |
| Command Relay | Forwards incoming Zigbee messages to outgoing Zigbee messages |
| cJSON | Open source JSON command parser (https://github.com/DaveGamble/cJSON) |
| Paho MQTT | Open source MQTT client (https://github.com/eclipse/paho.mqtt.c) |

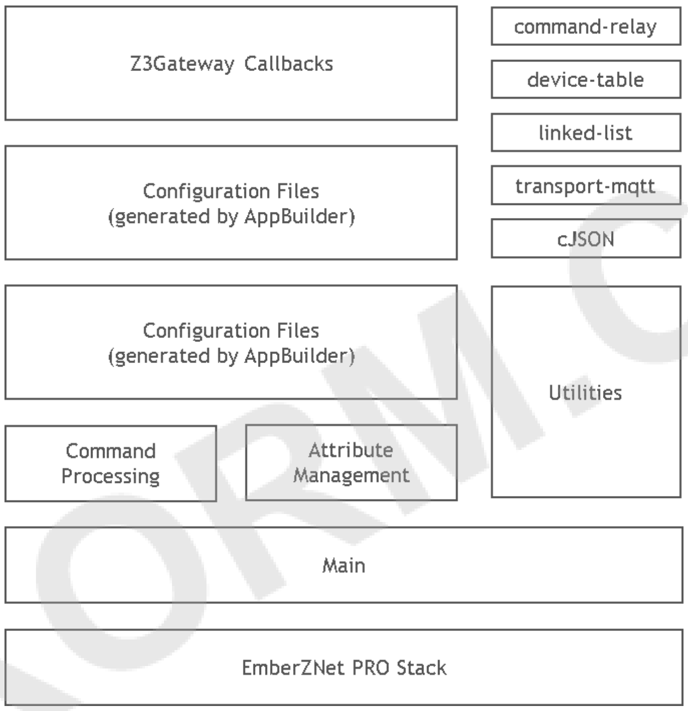


**Figure 6.6. Z3Gateway Application Diagram**

### 6.3.2 MQTT Pub/Sub Structure

The Z3Gateway application and Node server use MQTT topics and a Mosquitto MQTT broker for inter-process communication. Topics are also split so that information sent to the Z3Gateway application is never published on the same topic as information sent from the Z3Gateway application. All topics associated with the Z3Gateway application have the topic prefix = "`gw/<eui64_id>/`" where `<eui64_id>` is equal to the Z3Gateway application's EUI 64 identification.

Each MQTT topic contains a JSON payload. See each topic for more details.

**Table 6.2. MQTT Topic from the Z3Gateway Application**

| MQTT topic | Description |
|---|---|
| `gw/<eui64_id>/heartbeat` | An active ZB3 Gateway sends heartbeats containing its network status, short pan ID, transmit power and transmit channel |
| `gw/<eui64_id>/devices` | Responds to a request for the currently connected devices |
| `gw/<eui64_id>/relays` | Communicates current status of the command relay list |
| `gw/<eui64_id>/settings` | Gateway setting information containing NCP stack version, network status, short pan ID, transmit power, and channel number. |
| `gw/<eui64_id>/devicejoined` | Published when a node joins |
| `gw/<eui64_id>/deviceleft` | Published when a node leaves |
| `gw/<eui64_id>/devicestatechange` | Published when a device changes state |
| `gw/<eui64_id>/otaevent` | OTA event messages |
| `gw/<eui64_id>/executed` | Published when a command sent has been executed |
| `gw/<eui64_id>/zclresponse` | Generic ZCL level response for attribute report responses |
| `gw/<eui64_id>/zdoresponse` | Generic ZDO level response for binding and reporting table responses |
| `gw/<eui64_id>/apsresponse` | APS layer message transmission responses |

**Table 6.3. MQTT Topic to the Z3Gateway Application**

| MQTT topic | Description |
|---|---|
| `gw/<eui64_id>/commands` | Commands from the node server to process at the Z3Gateway application |
| `gw/<eui64_id>/publishstate` | Forces a publish of:<br>`gw/<eui64_id>/devices`<br>`gw/<eui64_id>/relays`<br>`gw/<eui64_id>/settings` |
| `gw/<eui64_id>/updatesettings` | Supported settings: trafficReporting: true/false. This enables advanced traffic diagnostics. |

### 6.3.3 MQTT Topic / Message Attributes

All MQTT topics:

- Use JSON format
- Publish to TCP port 1883
- Use QoS = 2

### 6.3.4 MQTT Topics Published to by the Z3Gateway Application

**Note:** This is information sent *from* the Z3Gateway application.

Table 6.4. Z3Gateway Application Heartbeat

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/heartbeat` | |
| **Trigger Event:** | Published by Z3Gateway application every five seconds | |
| **Description:** | An active Z3Gateway application sends heartbeats containing its network status, short pan ID, transmit power and transmit channel. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{`<br>` "networkUp":<Bool>,`<br>` "networkPanId": <String>,`<br>` "radioTxPower": <Int>,`<br>` "radioChannel": <Int>`<br>`}` | **networkUp** True or False<br><br>**networkPanId** formatted “0xXXXX”<br><br>**radioTxPower** transmission power setting<br><br>**radioChannel** number |

Table 6.5. Z3Gateway Application Device State

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/devices` | |
| **Trigger Event:** | Response to publish on MQTT channel: `gw/<eui64_id>/publishstate` | |
| **Description:** | This message is published in response to a request for the currently connected Z3Gateway application devices, and after each new joined device. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{`<br>`  "devices":[{`<br>`     "nodeId":<String>,`<br>`     "deviceState": <Int>,`<br>`     "deviceType":<String>,`<br>`     "timeSinceLastMessage":<Int>,`<br>`     "deviceEndpoint":{`<br>`       "eui64": <String>,`<br>`       "endpoint": <Int>,`<br>`       "clusterInfo":[{`<br>`         "clusterId":<String>,`<br>`         "clusterType":<String>`<br>`         }]`<br>`     },`<br>`  }]`<br>`}` | **devices** array containing all connected nodes<br><br>**nodeId** the node's 16-bit network ID formatted “0xXXXX”<br><br>**deviceState** current status of the node<br><br>**deviceType** Zigbee device ID number formatted “0xXXXX” (see ZCLDataTypes.js)<br><br>**timeSinceLastMessage** elapsed seconds since last message<br><br>**deviceEndpoint** array containing active endpoints<br><br>**eui64** EUI64 string formatted “0xXXXXXXXXXXXXXXXX”<br><br>**endpoint** Zigbee endpoint number<br><br>**clusterInfo** array containing clusters of the device<br><br>**clusterId** Zigbee cluster ID number formatted “0xXXXX” (see Constants.js)<br><br>**clusterType** In (server cluster) or Out (client cluster) |
| **Payload if no devices exist:** | `{`<br>`  "devices":[`<br>`  ]`<br>`}` | |

**Table 6.6. Z3Gateway Application Relays List**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/relays` | |
| **Trigger Event:** | Response to publish on MQTT channel: `gw/<eui64_id>/publishstate` | |
| **Description:** | This message is published in response to a request for the Z3Gateway application relay list contents. | |
| **Direction:** | From Z3Gateway application | |
| **Payload response when relays are present:** | `{ "relays":[{ "inDeviceEndpoint": { "eui64": <String>, "endpoint": <Int>, "clusterId": <String> }, "outDeviceEndpoint":{ "eui64": <String>, "endpoint": <Int>, "clusterId": <String> } }] }` | **relays** array contains all the relay connections between device endpoints<br>**inDeviceEndpoint** structure that contains device endpoint information for the relay source<br>**outDeviceEndpoint** structure that contains device endpoint information for the relay destination<br>**eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br>**endpoint** zigbee endpoint number<br>**clusterId** Zigbee cluster ID number formatted "0xXXXX" (see Constants.js) |
| **Payload response when no relays are present:** | `{ "relays":[ ] }` | |

**Table 6.7. Z3Gateway Application Gateway Info**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/settings` | |
| **Trigger Event:** | Response to publish on MQTT channel: `gw/<eui64_id>/publishstate` | |
| **Description:** | This is published in request for logging state from the Z3Gateway application. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{ "ncpStackVersion": <String>, "networkUp": <Bool>, "networkPanId": <String>, "radioTxPower": <Int>, "radioChannel": <Int> }` | **ncpStackVersion** stack version string<br>**networkUp** true or false<br>**networkPanId** network PAN formatted "0xXXXX"<br>**radioTxPower** Zigbee TX power in decimal<br>**radioChannel** Zigbee transmit channel in decimal |

**Table 6.8. Z3Gateway Application State Change**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/devicestatechange` | |
| **Trigger Event:** | Automatic response to a device changing state. | |
| **Description:** | This is published when the Z3Gateway application becomes aware of a change in the network state of a device on the Zigbee network. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{`<br>` "eui64": <String>,`<br>` "deviceState": <Int>`<br>`}` | **eui64** EUI64 string formatted “0xXXXXXXXXXXXXXXXX”<br><br>**deviceState** new state code for node<br><br>ND_JUST_JOINED: 0x00<br><br>ND_HAVE_ACTIVE: 0x01<br><br>ND_HAVE_EP_DESC: 0x02<br><br>ND_JOINED: 0x10<br><br>ND_UNRESPONSIVE: 0x11<br><br>ND_LEAVE_SENT: 0x20<br><br>ND_LEFT: 0x30<br><br>ND_UNKNOWN: 0xff |

**Table 6.9. Z3Gateway Application Node Join**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/devicejoined` | |
| **Trigger Event:** | Automatic response to a node joining the Zigbee network. | |
| **Description:** | This is published when a node joins the Z3Gateway application. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{`<br>`  "nodeId": <String>,`<br>`  "deviceState": <Int>,`<br>`  "deviceType": <String>,`<br>`  "timeSinceLastMessage": <Int>,`<br>`  "deviceEndpoint": {`<br>`    "eui64": <String>,`<br>`    "endpoint": <Int>`<br>`    "clusterInfo":[{`<br>`      "clusterId":<String>,`<br>`      "clusterType":<String>`<br>`    }]`<br>`  },`<br>`}` | **nodeId** the node's 16-bit network ID formatted “0xXXXX”<br><br>**deviceState** current status of the node<br><br>**deviceType** Zigbee device ID number formatted<br><br>**timeSinceLastMessage** elapsed seconds since last<br><br>**eui64** EUI64 string formatted “0xXXXXXXXXXXXXXXXX”<br><br>**endpoints** Zigbee endpoint number<br><br>**clusterInfo** array containing clusters of the device<br><br>**clusterId** Zigbee cluster ID number formatted “0xXXXX” (see Constants.js)<br><br>**clusterType** In (server cluster) or Out (client cluster) |

**Table 6.10. Z3Gateway Application Node Left**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/deviceleft` | |
| **Trigger Event:** | Automatic response to a node leaving the network, to tell a node to leave, MQTT publish to `message` the CLI command `zdo leave <nodeId> 1 0` | |
| **Description:** | This is published when a node leaves the Z3Gateway application. | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{`<br>`  "eui64": <String>`<br>`}` | **eui64** EUI64 string formatted “0xXXXXXXXXXXXXXXXX” |

**Table 6.11. Z3Gateway Application OTA Messages**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/otaevent` | |
| **Trigger Event:** | Automatically generated when an OTA event is occurring on the Zigbee network. For more advanced usage on starting an OTA event, see 6.3.9 CLI Command Structure. | |
| **Description:** | These messages are published by the Z3Gateway application when an OTA event occurs. | |
| **Direction:** | From Z3Gateway application | |
| **Payload <cmd> = finished:** | `{`<br>`  "messageType": "otaFinished",`<br>`  "eui64": <String>,`<br>`  "manufacturerId": <String>,`<br>`  "imageTypeId": <String>,`<br>`  "firmwareVersion": <String>`<br>`}` | **eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**manufacturerId** 16-bit manufacturer ID loaded<br><br>**imageTypeId** 16-bit image type ID loaded<br><br>**firmwareVersion** 32-bit firmware ID loaded |
| **Payload <cmd> = blocksent:** | `{`<br>`  "messageType": "otaBlockSent",`<br>`  "eui64": <String>,`<br>`  "bytesSent": <Int>,`<br>`  "manufacturerId":<String>,`<br>`  "imageTypeId": <String>,`<br>`  "firmwareVersion": <String>`<br>`}` | **eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**bytesSent** total number bytes that have been sent<br><br>**manufacturerId** 16-bit manufacturer ID loaded<br><br>**imageTypeId** 16-bit image type ID loaded<br><br>**firmwareVersion** 32-bit firmware ID loaded |
| **Payload <cmd> = started:** | `{`<br>` "messageType": "otaStarted",`<br>` "nodeEui": <String>,`<br>` "manufacturerId": < String>,`<br>` "imageTypeId": <String>,`<br>` "firmwareVersion": <String>`<br>`}` | **nodeEui** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**manufacturerId** 16-bit manufacturer ID loaded<br><br>**imageTypeId** 16-bit image type ID loaded<br><br>**firmwareVersion** 32-bit firmware ID loaded |
| **Payload <cmd> = failed:** | `{`<br>`  "messageType": "otaFailed"`<br>`  "eui64": <String>`<br>`  "manufacturerId": <String>`<br>`  "imageTypeId": <String>`<br>`  "firmwareVersion": <String>`<br>`}` | **eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**manufacturerId** 16-bit manufacturer ID loaded<br><br>**imageTypeId** 16-bit image type ID loaded<br><br>**firmwareVersion** 32-bit firmware ID loaded |

### 6.3.5 Zigbee Cluster Layer Response

Table 6.12. Zigbee Cluster Layer Response

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/zclresponse` | |
| **Trigger Event:** | Receipt of any attribute read response, attribute report, IAS zone (security) report or ZCL layer command. If a ZCL layer command was initiated the response payload will follow the command payload format. IAS zone events will follow the IAS zone payload format. All other `zclresponses` will utilize the attribute payload format. | |
| **Description:** | This topic will relay any ZCL layer attribute report, attribute read response, or cluster command from the Z3Gateway application. Note: An extra parsed response for the case of incoming security state update commands is also provided. This is in addition to the generic command update that will also be received. | |
| **Direction:** | From Z3Gateway application | |
| **Command Payload:** | `{`<br>`  "clusterId": <String>,`<br>`  "commandId": <String>,`<br>`  "commandData": <String>,`<br>`  "clusterSpecific": <Bool>,`<br>`  "mfgCode": <String>,`<br>`  "deviceEndpoint": {`<br>`    "eui64": <String>,`<br>`    "endpoint": <Int>`<br>`  }`<br>`}` | **clusterId** Cluster Id, formatted “0xXXXX”<br><br>**commandId** Command Id formatted “0xXX”<br><br>**commandData** Varying length data formatted “XXXX”<br><br>**clusterSpecific** “true” or “false”<br><br>**mfgCode** Optional if this is manufacturer specific command. Otherwise, omitted<br><br>**eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**endpoint** Zigbee endpoint number |
| **Attribute Payload:** | `{`<br>`  "clusterId": <String>,`<br>`  "attributeId": <String>,`<br>`  "attributeBuffer": <String>,`<br>`  "attributeDataType": <String>,`<br>`  "deviceEndpoint": {`<br>`    "eui64": <String>,`<br>`    "endpoint": <Int>`<br>`  }`<br>`}` | **clusterId** Zigbee reported cluster 16-bit ID<br><br>**attributeId** Zigbee reported attribute 16-bit ID<br><br>**attributeBuffer** Zigbee global read buffer response in hex string. This is parsed by the node server.<br><br>**attributeDataType** data type for the attribute buffer<br><br>**eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**endpoint** Zigbee endpoint number |

### 6.3.6 Zigbee Device Object Response

**Table 6.13. Zigbee Device Object Response**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/zdoresponse` | |
| **Trigger Event:** | Receipt of binding table or report table responses | |
| **Description:** | The gateway will use this topic to report binding table or report table activity, such as update success or table entry values. | |
| **Direction:** | From Z3Gateway application | |
| **Bind Response or Configure Report Response:** | `{ "zdoType": "bindResponse", "eui64": <String>, "status": <String> }` | **zdoType** "bindResponse" or "configureReportResponse" **eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX" **status** 0 = success, error code otherwise |
| **Bind Table Entry Response":** | `{ "zdoType": <String>, "status": <String>, "eui64": <String>, "bindTable" [{ "sourceDeviceEndpoint": <String>, "addressMode": <String>, "clusterId": <String>, "destDeviceEndpoint": <String> }] }` | **zdoType** "bindTableResponse" **status** 0x00 = success, error code otherwise **eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX" **bindTable** list if binding table entries **sourceDeviceEndpoint** endpoint of the source **addressMode** short or long address (can be ignored) **clusterId** cluster ID formatted "0xXXXX" **destDeviceEndpoint** endpoint of the destination |
| **Report Table Entry Response:** | `{ "zclType": <String>, "deviceEndpoint": <String>, "status": <String>, "direction": <Int>, "clusterId": <String>, "attributeId": <String>, "datatype": <String>, "minInterval": <String>, "maxInterval": <String>, "data": <String> }` | **zclType** "reportTableEntry" **deviceEndpoint** endpoint of the device **status** 0x00 = success, error code otherwise **direction** 0 = server-to-client, 1 = client-to-server **clusterId** cluster ID formatted "0xXXXX" **attributeId** attribute ID formatted "0xXXXX" **datatype** Zigbee data type formatted "0xXX". See ZCL spec for details. **minInterval** seconds, formatted "0xXXXX" **maxInterval** seconds, formatted "0xXXXX" **data** variable length data, formatted "XXXX" |

### 6.3.7 Zigbee APS Layer Response

Table 6.14. Zigbee APS Layer Response

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/apsResponse` | |
| **Trigger Event:** | Receipt of an APS layer response | |
| **Description:** | APS layer responses inform the Z3Gateway application whether a message was received (apsAck), and if it was received whether it was understood (defaultResponse). | |
| **Direction:** | From Z3Gateway application | |
| **Payload:** | `{ "statusType": <String>, "eui64": <String>, "status": <String>, "clusterId": <String>, "commandId": <String> }` | **statusType** "apsAck" or "defaultResponse"<br><br>**eui64** EUI64 string formatted "0xXXXXXXXXXXXXXXXX"<br><br>**status** 0x00 = success, error code otherwise<br><br>**clusterId** cluster Id formatted "0xXXXX"<br><br>**commandId** command Id formatted "0xXX" |

### 6.3.8 MQTT Topics Subscribed to by the Z3Gateway Application

**Note:** This is information sent *to* the Z3Gateway application.

**Table 6.15. ZCL Commands to Z3Gateway Application**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/commands` | |
| **Description:** | The Z3Gateway application can receive commands with this MQTT topic. The Z3Gateway application will acknowledge completion of the command with an executed MQTT topic message. | |
| **Direction:** | To Z3Gateway application | |
| **Payload:** | `{`<br>`  "commands":[{`<br>`    "command": <String>,`<br>`    "postDelayMs": <Int>`<br>`  }]`<br>`}` | CLI message list sent to Gateway<br><br>See section 6.3.9 CLI Command Structure to manually populate this list with commands. Each command is executed sequentially. postDelay is added after a command is run before the next command is processed.<br>Note: If postDelay is omitted, no post delay is added |

**Table 6.16. Message Commands to Z3Gateway Application**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/updatesettings` | |
| **Description:** | This topic updates settings of the Z3Gateway application. Supported settings: measureStatistics. | |
| **Direction:** | To Z3Gateway application | |
| **Payload:** | `{`<br>`"measureStatistics": true`<br>`}` | This command changes measureStatistics to true. |

**Table 6.17. Message Commands to Z3Gateway Application**

| | | |
|---|---|---|
| **MQTT Topic:** | `gw/<eui64_id>/publishstate` | |
| **Description:** | This topic forces a state update. | |
| **Direction:** | To Z3Gateway application | |
| **Payload:** | `{`<br>`}`<br>`or`<br>`NULL` | |

### 6.3.9 CLI Command Structure

The API for the Z3Gateway application exposes the CLI within the EmberZNet PRO stack. Commands such as `form` and `pjoin` (permit joining), as well as methods for building and sending ZCL commands to specific devices on the network, are used within the Z3Gateway application.

#### Sending a Standard ZCL Command

To send a ZCL command to a device on the network, one must follow a two-step process.

1. Build up the ZCL command in the outgoing command buffer.
2. Issue the `send` command which will send the contents of the outgoing command buffer to the intended recipient.

Example: Build and send a Zigbee command using standard ZCL commands:

```
zcl on-off on
send <shortID> <source-endpoint> <dest-endpoint>
```

Example: Build and send a Zigbee command using the device-table plugin:

```
zcl on-off on
plugin device-table send <deviceEui> <deviceEndpoint>
```

See the documentation in the EmberZNet PRO stack install for the full CLI description:
$STACK_ROOT/documentation/120-3023-000_AF_V2_API/group__cli.html

**Table 6.18. Common Zigbee CLI Commands Used**

| Command Description | Command | Parameters | Parameter Description |
|---|---|---|---|
| Forming a network | `network form <channel> <tx power> <pan id>` | `<channel>`<br>`<tx power>`<br>`<pan id>` | Channel used to form the network<br>TX power used to form the network<br>Short PAN ID to be used |
| Enable permit joining | `network pjoin <time>` | `<time>` | Integer value from 0 to 255 seconds |
| Enable network-wide joining | `network broad pjoin <time>` | `<time>` | Integer value from 0 to 255 seconds |
| Send a ZCL on-off command | `zcl on-off <command>` | `<command>` | Options are on, off, and toggle |

### 6.3.10 Plugin Command Structure

Several of the plugins used for the Z3Gateway application, such as device-table and command-relay, possess specific CLI commands. Each of the plugin commands within this document follow this plugin CLI command structure:

```
plugin <plugin-name> <plugin-cmd> <param-1> <param-2> ...
```

This CLI command structure uses the following parameters:

**Table 6.19. CLI Command Structure Parameters**

| Parameter | Description |
|---|---|
| `<plugin-name>` | Name of the plugin (ex: device-table, command-relay) |
| `<plugin-cmd>` | Command being called within the plugin |
| `<param-x>` | Each command within the plugin may have one to many parameters |

This example command is used to save the device-table:

```
plugin device-table save
```

### 6.3.11 Network Reform Example

Two ways to reform a network are presented in the following table.

**Table 6.20. Network Reform Example Topics and Payloads**

| Description | Prototype | Example |
|---|---|---|
| Form a ZB3 network with automatic clean channel searching | Publish Topic:<br>gw/<eui64>/commands<br>Publish Payload:<br>`{`<br>`   "commands":[{`<br>`   "command":`<br>`   "plugin network-creator start 1",`<br>`   "postDelayMs":<time>`<br>`   }]`<br>`}` | Publish Topic:<br>gw/0022A30000115EDA/commands<br>Publish Payload:<br>`{`<br>`   "commands":[{`<br>`   "command":`<br>`   "plugin network-creator start 1",`<br>`   "postDelayMs":0`<br>`   }]`<br>`}` |
| Form a ZB3 network with parameters | Publish Topic:<br>gw/<eui64>/commands<br>Publish Payload:<br>`{`<br>`   "commands":[{`<br>`   "command":`<br>`   "plugin network-creator form`<br>`   <channel>`<br>`   <power>`<br>`   <panId>",`<br>`   "postDelayMs":<time>`<br>`   }]`<br>`}` | Publish Topic:<br>gw/0022A30000115EDA/commands<br>Publish Payload:<br>`{`<br>`   "commands":[{`<br>`   "command":`<br>`   "plugin network-creator form`<br>`   11`<br>`   8`<br>`   0x1234",`<br>`   "postDelayMs":0`<br>`   }]`<br>`}` |

### 6.3.12 Permit Joining / Stop Joining

The following table presents the methods to permit joining and to stop joining in a ZB3 network.

**Table 6.21. Permit Joining / Stop Joining Example Topics and Payloads**

| Description | Prototype | Example |
|---|---|---|
| Permit devices to join a ZB3 network with or without install code | Publish Topic:<br><br>gw/<eui64>/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"network broad-pjoin 0"`<br>`<deviceEui>`<br>`<link-key derived from install code>", "postDela`<br>`yMs":<time>"},{`<br>`"command":"plugin network-creator-security open-`<br>`network",`<br>`"postDelayMs":<time>`<br>`}]`<br>`}` | Publish Topic:<br><br>gw/0022A30000115EDA/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"network-creator-security set-joining-link-key`<br>`{00 0D A3 00 00 11 5E DA}`<br>`{90 EF 8B D1 78 32 6C 2A 3E 8F DF 61 DF 1B CC 4B`<br>`}",`<br>`"postDelayMs":0`<br>`},`<br>`{"command":"plugin network-creator-security open`<br>`-network","postDelayMs:100"}]`<br>`}` |
| Permit devices to join a ZB3.0 network with install code only | Publish Topic:<br><br>gw/<eui64>/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security clear-joining-l`<br>`ink-keys"},{`<br>`"command":`<br>`"plugin network-creator-security open-with-key`<br>`<deviceEui>`<br>`<link-key derived from install code>", "postDela`<br>`yMs":<time>`<br>`}]`<br>`}` | Publish Topic:<br><br>gw/0022A30000115EDA/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security clear-joining-l`<br>`ink-keys"},{`<br>`"command":`<br>`"plugin network-creator-security open-with-key`<br>`{00 0D A3 00 00 11 5E DA}`<br>`{90 EF 8B D1 78 32 6C 2A 3E 8F DF 61 DF 1B CC 4B`<br>`}",`<br>`"postDelayMs":100`<br>`}]`<br>`}` |
| Permit devices to join a ZB3.0 network without install code | Publish Topic:<br><br>gw/<eui64>/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security open-network",`<br>`"postDelayMs":<time>`<br>`}]`<br>`}` | Publish Topic:<br><br>gw/0022A30000115EDA/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security open-network",`<br>`"postDelayMs":100`<br>`}]`<br>`}` |
| Stop devices from joining a ZB3.0 network | Publish Topic:<br><br>gw/<eui64>/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security close-network",`<br>`"postDelayMs":<time>`<br>`}]`<br>`}` | Publish Topic:<br><br>gw/0022A30000115EDA/commands<br><br>Publish Payload:<br><br>`{`<br>`"commands":[{ "command":`<br>`"plugin network-creator-security close-network",`<br>`"postDelayMs":100`<br>`}]`<br>`}` |

### 6.3.13 Lighting Configuration Example

This section demonstrates the steps required by the Z3Gateway application to control a light. The following table describes each step with prototype and example. After the table is a condensed version with publish payload more suitable for cutting and pasting. Note that in these examples the EUI64 for the Z3Gateway application is 0022A30000115EDA and the EUI64 for the light is 000B57FFFE097874.

**Table 6.22. Lighting Example Topics and Payloads**

| Step | Prototype | Example |
|---|---|---|
| Form the network | See section 6.3.11 Network Reform Example | See section 6.3.11 Network Reform Example |
| Permit devices to join and view devices | See section 6.3.12 Permit Joining / Stop Joining | See section 6.3.12 Permit Joining / Stop Joining |
| Send on command and view response | Subscribe Topic:<br>gw/<eui64>/apsresponse<br>gw/<eui64>/zclresponse<br><br>Publish Topic:<br>gw/<eui64>/commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>`  "command":"zcl on-off on",`<br>`  "postDelayMs":<time>`<br>` },{`<br>`  "command":"plugin`<br>`  device-table send`<br>`  <deviceEui>`<br>`  <deviceEndpoint>",`<br>`  "postDelayMs":<time>`<br>` }]`<br>`}` | Subscribe Topic:<br>gw/0022A30000115EDA/apsresponse<br>gw/0022A30000115EDA/zclresponse<br><br>Publish Topic:<br>gw/0022A30000115EDA /commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>`  "command":"zcl on-off on",`<br>`  "postDelayMs":0`<br>` },{`<br>`  "command":"plugin`<br>`  device-table send`<br>`  {000B57FFFE097874}`<br>`  1",`<br>`  "postDelayMs":0`<br>` }]`<br>`}` |
| Send move-to-level command and view response | Subscribe Topic:<br>gw/<eui64>/apsresponse<br>gw/<eui64>/zclresponse<br><br>Publish Topic:<br>gw/<eui64>/commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>`  "command" :"zcl level-control`<br>` mv-to-level <level> <time>",`<br>`  "postDelayMs":<time>`<br>` },{`<br>`  "command":"plugin`<br>`  device-table send`<br>`  <deviceEui>`<br>`  <deviceEndpoint>",`<br>`  "postDelayMs":<time>`<br>` }]`<br>`}` | Subscribe Topic:<br>gw/0022A30000115EDA /apsresponse<br>gw/0022A30000115EDA /zclresponse<br><br>Publish Topic:<br>gw/0022A30000115EDA /commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>`  "command" :"zcl level-control`<br>` mv-to-level 0xFE 0",`<br>`  "postDelayMs":0`<br>` },{`<br>`  "command":"plugin`<br>`  device-table send`<br>`  {000B57FFFE097874}`<br>`  1",`<br>`  "postDelayMs":0`<br>` }]`<br>`}` |

| Step | Prototype | Example |
|---|---|---|
| Send move-to-hue-saturation command and view response | Subscribe Topic:<br>gw/<eui64>/apsresponse<br>gw/<eui64>/zclresponse<br><br>Publish Topic:<br>gw/<eui64>/commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>` "command":"zcl color-control`<br>` movetohueandsat`<br>` <hue> <sat> <time>",`<br>` "postDelayMs":<time>`<br>` },{`<br>` "command":"plugin`<br>` device-table send`<br>` <deviceEui>`<br>` <deviceEndpoint>",`<br>` "postDelayMs":<time>`<br>` }]`<br>`}` | Subscribe Topic:<br>gw/0022A30000115EDA /apsresponse<br>gw/0022A30000115EDA /zclresponse<br><br>Publish Topic:<br>gw/0022A30000115EDA /commands<br><br>Publish Payload:<br>`{`<br>` "commands":[{`<br>` "command":"zcl color-control`<br>` movetohueandsat`<br>` 0x78 0xFE 0",`<br>` "postDelayMs":0`<br>` },{`<br>` "command":"plugin`<br>` device-table send`<br>` {000B57FFFE097874}`<br>` 1",`<br>` "postDelayMs":0`<br>` }]`<br>`}` |

1. Form the network (see section 6.3.11 Network Reform Example).
2. Permit devices to join and view devices (see section 6.3.12 Permit Joining / Stop Joining).
3. Send on command and view response
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl on-off on", "postDelayMs":0},{"command":"plugin device-table send {000B57
FFFE097874} 1","postDelayMs":0}]}
```

4. Send move-to-level command and view response
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl level-control mv-to-level 0xFE 0","postDelayMs":0},{"command":"plugin dev
ice-table send {000B57FFFE097874}  1","postDelayMs":0}]}
```

5. Send move-to-hue-saturation command and view response
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl color-control movetohueandsat 0x78 0xFE 0","postDelayMs":0},{"command":"p
lugin device-table send {000B57FFFE097874} 1","postDelayMs":0}]}
```

### 6.3.14 Occupancy Sensor Configuration Example

This section demonstrates the steps required by the Z3Gateway application to receive attributes' update from an occupancy sensor. The following table describes each step with a prototype and an example. After the table is a condensed version with publish payload more suitable for cutting and pasting. Note that in these examples the EUI64 for the Z3Gateway application is 0022A30000115EDA, the EUI64 for the occupancy sensor is 000B57FFFE097874 and the nodeId for the occupancy sensor is 0x434C.

**Table 6.23. Occupancy Sensor Example Topics and Payloads**

| Step | Prototype | Example |
|---|---|---|
| Form the network | See section 6.3.11 Network Reform Example | See section 6.3.11 Network Reform Example |
| Permit devices to join and view devices | See section 6.3.12 Permit Joining / Stop Joining | See section 6.3.12 Permit Joining / Stop Joining |
| Send attributes reading command and view response | Subscribe Topic: gw/<eui64>/apsresponse gw/<eui64>/zclresponse Publish Topic: gw/<eui64>/commands Publish Payload: `{ "commands":[{ "command":"zcl global direction 0" },{ "command":"zcl global read <clusterId> <attributeId>"},{ "command":"plugin device-table send <deviceEui> <deviceEndpoint>" }] }` | Subscribe Topic: gw/0022A30000115EDA/apsresponse gw/0022A30000115EDA/zclresponse Publish Topic: gw/0022A30000115EDA /commands Publish Payload: `{ "commands":[{ "command":"zcl global direction 0" },{ "command":"zcl global read 0x0400 0"},{ "command":"plugin device-table send {000B57FFFE097874} 1" }] }` |
| Send cluster binding command and view response | Subscribe Topic: gw/<eui64>/apsresponse gw/<eui64>/zclresponse Publish Topic: gw/<eui64>/commands Publish Payload: `{ "commands":[{ "command":"zdo bind <nodeId> <sourceEndpoint> <destinationEndpoint> <clusterId> <destinationEUI> <gatewayEUI>"},{ "command":"plugin device-table send <deviceEui> <deviceEndpoint>", "postDelayMs": <time> }] }` | Subscribe Topic: gw/0022A30000115EDA /apsresponse gw/0022A30000115EDA /zclresponse Publish Topic: gw/0022A30000115EDA /commands Publish Payload: `{ "commands":[{ "command":"zdo bind 0x434C 0x1 0x1 0x0400 {000B57FFFE097874} {0022A30000115EDA}"},{ "command":"plugin device-table send {000B57FFFE097874} 1", "postDelayMs":100 }] }` |

| Step | Prototype | Example |
|---|---|---|
| Set send-me-a-report command and view response | Subscribe Topic:<br>gw/<eui64>/apsresponse<br>gw/<eui64>/zclresponse<br><br>Publish Topic:<br>gw/<eui64>/commands<br><br>Publish Payload:<br>`{`<br>`  "commands":[{`<br>`  "command":"zcl global send-me-a-report <clusterId> <attributeId> <dataType> <minReportTime> <maxReportTime> <reportableChange>"},{`<br>`  "command":"plugin device-table send <deviceEui> <deviceEndpoint>",`<br>`  "postDelayMs": <time>`<br>` }]`<br>`}` | Subscribe Topic:<br>gw/0022A30000115EDA /apsresponse<br>gw/0022A30000115EDA /zclresponse<br><br>Publish Topic:<br>gw/0022A30000115EDA /commands<br><br>Publish Payload:<br>`{`<br>`  "commands":[{`<br>`  "command":"zcl global send-me-a-report 0x0402 0x0000 0x29 0x0001 0x0708 {32 00}"},{`<br>`  "command":"plugin device-table send {000B57FFFE097874} 1",`<br>`  "postDelayMs":100`<br>` }]`<br>`}` |

1. Form the network (see section 6.3.11 Network Reform Example).
2. Permit devices to join and view devices (see section 6.3.12 Permit Joining / Stop Joining).
3. Send illuminance measurement reading command and view response.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl global direction 0"},{"command":"zcl global read 0x0400 0"},{"command":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

   e. Read illuminance value from topic: gw/0022A30000115EDA/zclresponse
   f. Received Payload:

```
{"clusterId":"0x0400","attributeId":"0x0000","attributeBuffer":"0x30","attributeDataType":"0x21","deviceEndpoint":{"eui64":"000B57FFFE097874","endpoint":1}}
```

4. Send reading command of occupancy attribute and view response.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :" zcl global direction 0"},{"command":"zcl global read 0x0406 0"},{"command":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

   e. Read occupancy attribute from topic: gw/0022A30000115EDA/zclresponse.
   f. Received Payload:

```
{"clusterId":"0x0406","attributeId":"0x0000","attributeBuffer":"0x00","attributeDataType":"0x18","deviceEndpoint":{"eui64":"000B57FFFE097874","endpoint":1}}
```

5. Send reading command of humidity measured value attribute and view response.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse

c. Publish Topic: gw/0022A30000115EDA/commands

d. Publish Payload:

```
{"commands":[{"command" :"zcl global direction 0"},{"command":"zcl global read 0x405 0"},{"command":"pl
ugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

e. Read humidity measured value from topic: gw/0022A30000115EDA/zclresponse.

f. Received Payload:

```
{"clusterId":"0x0405","attributeId":"0x0000","attributeBuffer":"0x2919","attributeDataType":"0x21","dev
iceEndpoint":{"eui64":"000B57FFFE097874","endpoint":1}}
```

6. Send reading command of temperature measured value attribute and view response.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zcl global direction 0"},{"command":"zcl global read 0x0402 0"},{"command":"p
lugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

    e. Read temperature measured value from topic: gw/0022A30000115EDA/zclresponse.
    f. Received Payload:

```
{"clusterId":"0x0402","attributeId":"0x0000","attributeBuffer":"0x2409","attributeDataType":"0x29","dev
iceEndpoint":{"eui64":"000B57FFFE097874","endpoint":1}}
```

7. Send illuminance binding command.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zdo bind 0x434C 0x1 0x1 0x0400 {000B57FFFE097874} {0022A30000115EDA}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

8. Set send-me-a-report command for illuminance measured value.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0400 0x0000 0x21 0x0001 0x0708 {A6 33}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

9. Send humidity binding command.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zdo bind 0x434C 0x1 0x1 0x0405 {000B57FFFE097874} {0022A30000115EDA }"},{"com
mand":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

10. Set send-me-a-report command for humidity measured value.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands

d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0405 0x0000 0x21 0x0001 0x0708 {F4 01}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

11. Send temperature binding command.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zdo bind 0x434C 0x1 0x1 0x0402 {000B57FFFE097874} {0022A30000115EDA}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

12. Set send-me-a-report command for temperature measured value.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0402 0x0000 0x29 0x0001 0x0708 {32 00}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

13. Send occupancy binding command.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zdo bind 0x434C 0x1 0x1 0x0406 {000B57FFFE097874} {0022A30000115EDA}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

14. Set send-me-a-report command for occupancy state.
    a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
    b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
    c. Publish Topic: gw/0022A30000115EDA/commands
    d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0406 0x0000 0x18 0x0001 0x0708 {01}"},{"command
":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

### 6.3.15 Contact Sensor Configuration Example

Prototypes and examples for a contact sensor could refer to occupancy sensor. Also, temperature measured value reading, binding and send-me-a-report setting could refer to occupancy sensor.

### 6.3.16 Smart Outlet Configuration Example

This section demonstrates the steps required by the Z3Gateway application to receive attributes' update from a smart outlet. A condensed version of steps is presented with publish payloads. Note that in these examples the EUI64 for the Z3Gateway application is 0022A30000115EDA, the EUI64 for the smart outlet is 000B57FFFE097874, and the nodeId for the smart outlet is 0x434C. Prototypes and examples could refer to occupancy sensor. Temperature/humidity/illuminance measured value reading, binding and send-me-a-report setting could refer to occupancy sensor as well. Turning on/off or toggling main power on a smart outlet could refer to related controls in the lighting example. The following examples present the payloads that are exclusive to an outlet.

1. Send RMS current reading command and view response.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl global direction 0"},{"command":"zcl global read 0x0B04 0x0508"},{"comman
d":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

   e. Read RMS current value from topic: gw/0022A30000115EDA/zclresponse
   f. Received Payload

```
{"clusterId":"0x0B04","attributeId":"0x0508","attributeBuffer":"0x2123","attributeDataType":"0x21","dev
iceEndpoint":{"eui64":"000B57FFFE097874","endpoint":1}}
```

2. Send binding command for electrical measurement cluster.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zdo bind 0x434C 0x1 0x1 0x0B04 {000B57FFFE097874} {0022A30000115EDA }"},{"com
mand":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

3. Set send-me-a-report command for RMS current.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0B04 0x0508 0x21 0x0001 0x0708 {0A 00}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

4. Set send-me-a-report command for RMS voltage.
   a. Subscribe Topic: gw/0022A30000115EDA/apsresponse
   b. Subscribe Topic: gw/0022A30000115EDA/zclresponse
   c. Publish Topic: gw/0022A30000115EDA/commands
   d. Publish Payload:

```
{"commands":[{"command" :"zcl global send-me-a-report 0x0B04 0x0505 0x21 0x0001 0x0708 {0A 00}"},{"comm
and":"plugin device-table send {000B57FFFE097874} 1","postDelayMs":100}]}
```

### 6.3.17 Plugin: Device Table

The Device Table plugin is a mechanism for keeping track of devices as they join the network. It uses the trust center callback as notification of a new device joining the network. This starts an interrogation of the new device for its active endpoints, device types, and clusters supported. This can be modified to include additional information that the application requires.

**Table 6.24. Device Table CLI Command Summary**

| Command Description | Command | Parameters | Parameter Description |
|---|---|---|---|
| Discover a new device | `plugin device-table disc <nodeId>` | `<nodeId>` | 16-bit address of device to be discovered |
| Send message command | `plugin device-table send <deviceEui> <deviceEndpoint>` | `<deviceEui>` `<deviceEndpoint>` | n/a |
| Save the device table | `plugin device-table save` | n/a | n/a |
| Load the device table | `plugin device-table load` | n/a | n/a |
| Print the device table | `plugin device-table print` | n/a | n/a |

**CLI Command Detail**

**Table 6.25. Discover a New Device**

| | |
|---|---|
| **Command:** | `plugin device-table disc <nodeId>` |
| **Description:** | This command enables the device discovery process. |
| **Parameter(s):** | `<nodeId>` = 16-bit address of the device to be discovered (in 0x hex format) |

**Table 6.26. Send message command**

| | |
|---|---|
| **Command:** | `plugin device-table send <deviceEui> <deviceEndpoint>` |
| **Description:** | This command sends a built ZCL message to the device. |
| **Parameter(s):** | `<deviceEui>` = Device EUI64<br>`<deviceEndpoint>` = Device endpoint number |

**Table 6.27. Save the Device Table**

| | |
|---|---|
| **Command:** | `plugin device-table save` |
| **Description:** | This command saves the device table to the local disk. |
| **Parameter(s):** | n/a |

**Table 6.28. Load Device Table**

| | |
|---|---|
| **Command:** | `plugin device-table load` |
| **Description:** | This command loads the last saved device table from the local disk. |
| **Parameter(s):** | n/a |

**Table 6.29. Print Device Table**

| | |
|---|---|
| **Command:** | `plugin device-table print` |
| **Description:** | This command prints the current device table information. |
| **Parameter(s):** | n/a |

### 6.3.18 Plugin: Command Relay

#### Overview

The command relay engine intercepts incoming messages from devices on the network and passes them on to other devices as outgoing commands. This can be used as a very simple rules engine.

#### Add Relays

Adding relays is a simple way to take all of the input from a single device (such as a light switch) and forward the input to one or more devices (such as lights). There is no interpretation of the incoming messages. Incoming messages are forwarded exactly as they came in.

**Table 6.30. Add Relays CLI Command Detail**

| Command Description | Command | Parameters | Parameter Description |
|---|---|---|---|
| Add relay | `plugin command-relay add` `<inDeviceEui>` `<inDeviceEndpoint>` `<inDeviceClusterId>` `<outDeviceEui>` `<outDeviceEndpoint>` `<outDeviceClusterId>` | `<inDeviceEui>` `<inDeviceEndpoint>` `<inDeviceClusterId>` `<outDeviceEui>` `<outDeviceEndpoint>` `<outDeviceClusterId>` | Device #1 that will send commands<br><br>Device #2 that will receive commands |
| Save relays | `plugin command-relay save` | n/a | n/a |
| Load relays | `plugin command-relay load` | n/a | n/a |
| Print relays | `plugin command-relay print` | n/a | n/a |
| Clear relays | `plugin command-relay clear` | n/a | n/a |

**Add Relays CLI Command Detail**

**Table 6.31. Add a Relay**

| | |
|---|---|
| **Command:** | `plugin command-relay add <inDeviceEui> <inDeviceEndpoint> <inDeviceClusterId> <outDeviceEui> <outDeviceEndpoint> <outDeviceClusterId>` |
| **Description:** | This command forwards incoming messages from one device to another. This relay is one way (“in” device will be sent to the “out” device, but not vice versa). |
| **Parameter(s):** | <inDeviceEui> = eui64 of the device that will send commands<br><br><inDeviceEndpoint> = endpoint number of the device that will send commands<br><br><inDeviceClusterId> = cluster ID of the device that will send commands<br><br><outDeviceEui> = eui64 of the device that will receive commands<br><br><outDeviceEndpoint> = endpoint number of the device that will receive commands<br><br><outDeviceClusterId> = cluster ID of the device that will receive commands |

**Table 6.32. Save Relays**

| | |
|---|---|
| **Command:** | `plugin command-relay save` |
| **Description:** | This command persists the relay table from the command relay. |
| **Parameter(s):** | n/a |

**Table 6.33. Load Relays**

| | |
|---|---|
| **Command:** | `plugin command-relay load` |
| **Description:** | This command loads the relay table from the disk. |
| **Parameter(s):** | n/a |

**Table 6.34. Print Relays**

| | |
|---|---|
| **Command:** | `plugin command-relay print` |
| **Description:** | This command prints the relay table from the command relay. |
| **Parameter(s):** | n/a |

**Table 6.35. Clear Relay**

| | |
|---|---|
| **Command:** | `plugin command-relay clear` |
| **Description:** | This command clears the relay table in the command relay. |
| **Parameter(s):** | n/a |

### 6.3.19 How to Send CLI Commands via MQTT Clients

Users could connect their MQTT clients, and send CLI commands via such clients. Assume a newly created client is connected to a local MQTT broker, users could wrap CLI commands in the following payload template, and publish the payload through “commands” MQTT topic (Table 6.3 MQTT Topic to the Z3Gateway Application on page 28). In the template, “postDelayMs” is optional, and at least one command forms a valid payload.

```
{"commands":[{"commandcli":"<cli command>"},{"commandcli":"<cli command>","postDelayMs":100}]}
```

It is noteworthy in the template that a CLI command is wrapped using a tag “commandcli” rather than the aforementioned “command”. On the NodeJS server, although “commandcli” brings the same effect, a few commands utilize “command” to ensure backward compatibility. The following table lists all the commands that could be used with “command” and “commandcli” tags. Commands that are not included in the list should be used with “commandcli” as the tag.

**Table 6.36. CLI Commands with Both “command” and “commandcli” Tags**

| | |
|---|---|
| `plugin network-creator-security clear-joining-link-keys` | `network broad-pjoin` |
| `plugin network-creator-security set-joining-link-key` | `network leave` |
| `plugin network-creator-security open-network` | `zcl global direction` |
| `plugin network-creator-security close-network` | `zcl global write` |
| `plugin network-creator-security open-with-keys` | `zcl global read` |
| `plugin ias-zone-client clear-all` | `zcl level-control o-mv-to-level` |
| `plugin groups-server clear` | `zcl color-control movetocolortemp` |
| `plugin scenes clear` | `zcl color-control movetohueandsat` |
| `plugin network-creator start` | `zcl on-off toggle` |
| `plugin network-creator form` | `zcl on-off off` |
| `plugin device-table send` | `zcl on-off on` |
| `plugin command-relay add` | `zcl bind` |
| `plugin command-relay remove` | `zdo leave` |
| `plugin command-relay clear` | |
| `plugin device-table clear` | |
| `plugin ota-server policy query` | |
| `plugin ota-server notify` | |
| `plugin ota-storage-common reload` | |

Here is an example of publishing CLI commands (listed in the previous table) using mosquitto-clients (install by `apt-get install mosquitto-clients`) where the payload is transformed to a C-code string.

```
mosquitto-pub -t "gw/<eui64_id>/commands" -q 2 -m "{\"commands\":[{\"commandcli\":\"plugin network-creator star
t 1\", \"postDelayMs\":0}]}"
```

or

```
mosquitto-pub -t "gw/<eui64_id>/commands" -q 2 -m "{\"commands\":[{\"command\":\"plugin network-creator start 1
\", \"postDelayMs\":0}]}"
```

If the command is not listed in the previous table, the valid form of a payload is shown in the following example:

```
mosquitto-pub -t "gw/<eui64_id>/commands" -q 2 -m "{\"commands\":[{\"commandcli\":\"plugin command-relay print\
"}]}"
```

### 6.4 NodeJS Server Application

The node server is built with various node.js version 5.12.0 packages including:

**Table 6.37. node.js Packages**

| Package | Description |
|---|---|
| Npm | Node package manager |
| Mqtt | Used to interface to an MQTT broker |
| IP | IP address utilities |
| Socket.io | Real-time framework server |
| Express | Web application framework utilities |
| Winston | Used for asynchronous logging |

### 6.4.1 Node Server Architecture

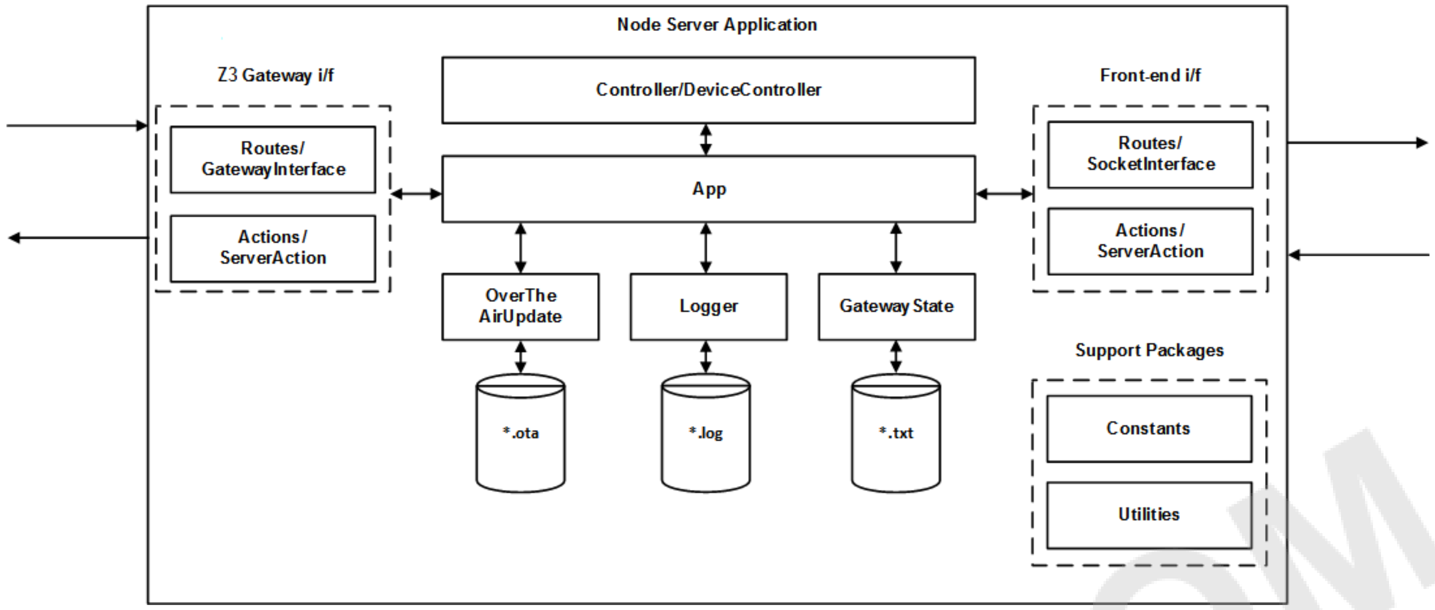


**Figure 6.7. Node Server Application Diagram**

**Table 6.38. File Structure**

| Package | Description |
|---|---|
| Local/App.js | Top file |
| Common/Constants.js | Contains constant values and state names |
| Common/CustomerTest.js | Supports the network test feature |
| Common/GatewayState.js | Stores persistent information |
| Common/Logger.js | Supports the log files |
| Controller/Sub-modules/OverTheAirUp-date.js | Supports over-the-air updates |
| Common/Utilities.js | Supports basic conversion formatting algorithms |
| Actions/ServerActions.js | Sends MQTT messages to the Z3Gateway application, sends socket IO messages to the front end. |
| Controller/DeviceController.js | Used to communicate JSON messaging to the front end app |
| Routes/GatewayInterface.js | Receives MQTT messages from the Z3Gateway application API |
| Routes/SocketInterface.js | Receives socket IO messages from the front end |

## 6.5 ReactJS Front-end Application

### 6.5.1 Overview

The front end application that supports web clients is built with various node.js version 5.12.0 tools including:

**Table 6.39. node.js Tools**

| Package | Description |
|---|---|
| Npm | Node package manager |
| React | Overall front-end view architecture |
| Flux | Application architecture |
| Fluxxor | Architecture tool support |
| Socket.io | Real-time framework server |
| Browserify | Supports require() calls and loads into bundle.js |
| Underscore | Adds additional JavaScript programming helpers |
| Semantic-ui | Package UI object library (set to default) |
| Gulp | Supports streaming builds |

### 6.5.2 Front-end App Architecture

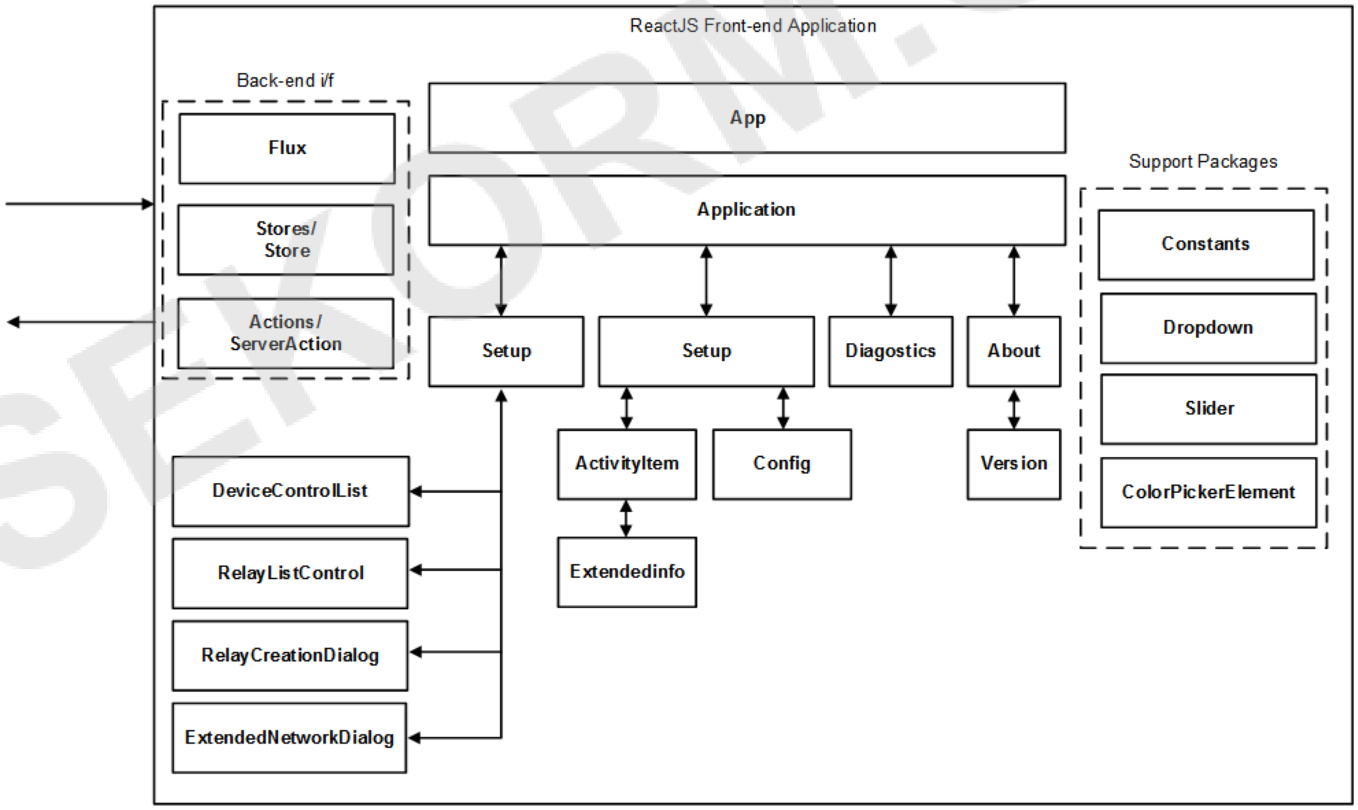


**Figure 6.8. React Front-end Application Diagram**

**Table 6.40. File Structure**

| Package | Description |
|---|---|
| About.react.js | Renders the about tab which includes version and support information |
| ActivityItem.react.js | Renders the control and monitor attributes of each connected device |
| App.js | Top file |
| Application.react.js | Supports top menu bar and renders the user-selected menu tab |
| ColorPickerElement.react.js | Supports color selection for lighting devices with color hue support |
| Config.js | Reads in the gateway address |
| Constants.react.js | Contains constant values and state names |
| ControlPanel.react.js | Supports the control panel menu for adding devices, managing rules and Zigbee network settings |
| DeviceControlList.react.js | Manages join/removal of devices along with simplified control and status features |
| Diagnostics.react.js | Supports the console logs and network test |
| Dropdown.react.js | Supports active dropdown menus |
| ExtendedInfo.react.js | Supports the extended info feature which includes OTA upgrades on each connected device |
| ExtendedNetworkDialog.react.js | Control and status of Zigbee network settings |
| Home.react.js | Supports the control and status of connected devices |
| RuleCreationDialog.react.js | Manages the creation of rules |
| RuleListControl.react.js | Enumerates the created rules |
| Slider.react.js | Supports user-controllable slide selectors |
| Version.js | Contains key package versions used by the application |
| Actions/ServerActions.js | Sends and receives socket IO messages to/from the Node server. |

## 7. Next Steps

The Zigbee Gateway Reference Designs are designed to demonstrate Zigbee gateway functionality with Silicon Labs Zigbee reference designs such as *RD-0020-0601, RD-0035-0601, and RD-0085-0401: Lighting Reference Designs*, *RD-0030-0201: Contact Sensor Reference Design*, *RD-0039-0201: Capacitive Sense Dimmer Switch Reference Design*, *RD-0051-0201: Smart Outlet Reference Design*, and *RD-0078-0201: Occupancy Sensor Reference Design*. For next steps, refer to the user's guides for each of these reference designs.

# 8. Troubleshooting

## 8.1 Zigbee Network Down

Confirm the CEL MeshConnect USB Stick or EFR32 Mighty Gecko Wireless Starter Kit is available, as shown in the figure below.

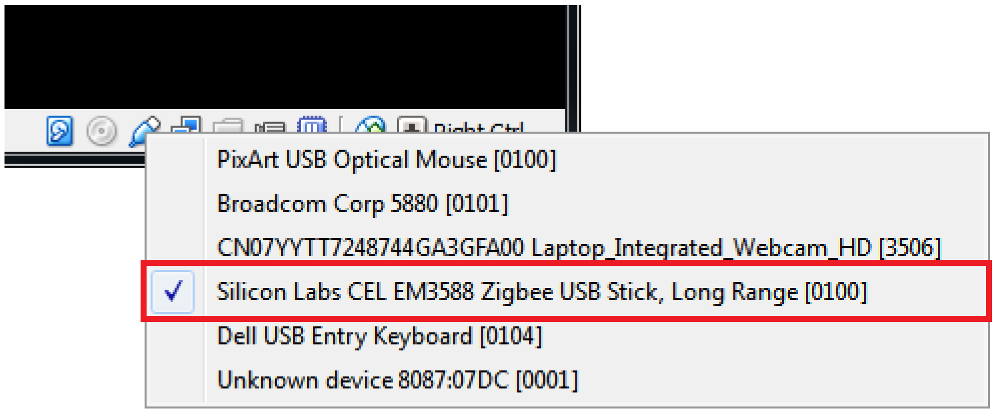
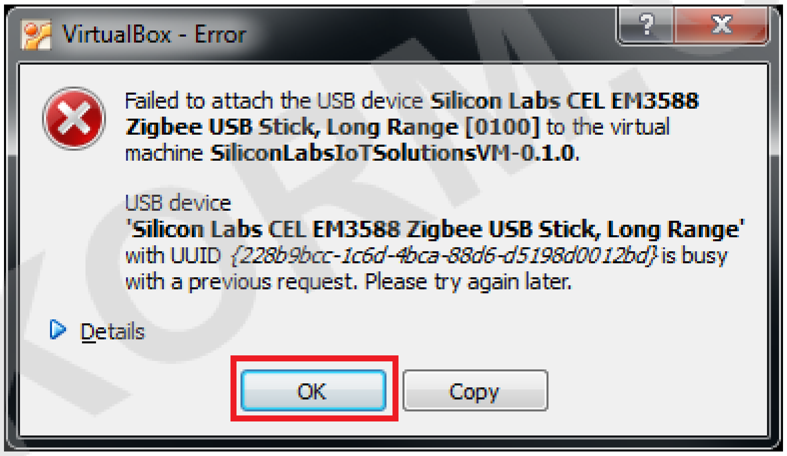

PixArt USB Optical Mouse [0100]
Broadcom Corp 5880 [0101]
CN07YYTT7248744GA3GFA00 Laptop_Integrated_Webcam_HD [3506]
✓ Silicon Labs CEL EM3588 Zigbee USB Stick, Long Range [0100]
Dell USB Entry Keyboard [0104]
Unknown device 8087:07DC [0001]

**Figure 8.1. Confirm Zigbee USB Virtual Gateway Hardware Availability**

If the CEL MeshConnect USB Stick or EFR32 Mighty Gecko Wireless Starter Kit does not appear as an option, or if an error occurs such as shown in the following figure, it may be in use by the Windows or OSX host operating system. Close the Ubuntu Virtual Machine and Oracle VM VirtualBox Manager, remove the CEL MeshConnect USB Stick or EFR32 Mighty Gecko Wireless Starter Kit, and confirm in Device Manager (Windows) or System Information (OSX) that the device is no longer present.



Failed to attach the USB device **Silicon Labs CEL EM3588 Zigbee USB Stick, Long Range [0100]** to the virtual machine **SiliconLabsIoTSolutionsVM-0.1.0**.

USB device **'Silicon Labs CEL EM3588 Zigbee USB Stick, Long Range'** with UUID *{228b9bcc-1c6d-4bca-88d6-d5198d0012bd}* is busy with a previous request. Please try again later.

**Figure 8.2. Error Condition**

## 8.2 Unable to Add Devices

The Zigbee end node may not be in the active network search state or security settings may be incorrect. Refer to the user's guide for each device and verify network search mode.

## 8.3 Unable to Remove Devices

When attempting to remove Zigbee end nodes by selecting the "X" next to the device name, the message "leave sent" may appear but the device remains in the device list. The device may be powered down or out of range, and unable to acknowledge the request. Once powered up and in range, the end node will acknowledge the request to leave and disappear from the device list.

## 8.4 Other Issues

If you are having an issue, visit http://community.silabs.com for Knowledge Base or discussions on these reference designs. Additionally, you may also request support at http://silabs.com/support.

## Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

**IoT Portfolio**
*www.silabs.com/IoT*

**SW/HW**
*www.silabs.com/simplicity*

**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**