IoT Developer Boot-camp: ZigBee 3.0

KEY FEATURES

Step-by-step guide to create, build and run ZigBee 3.0 applications based on EmberZNet 6.4.0

Use Simplicity Studio v4 as the development tool

ZigBee on EFR32MG

ZigBee 3.0 networks and security

In this worksheet we provide a step-by-step guide to create, build and run ZigBee 3.0 applications based on EmberZNet Stack 6.6.3. If you use a later release in the future, most of the instructions should be still applied, although there could be minor differences not foreseen at the time of this document.

These exercises help you get familiar with ZigBee 3.0 in the EmberZNet Stack, Simplicity Studio v4 development environment, and the Wireless Start Kit (WSTK) with EFR32MG12 SoC. We assume that you have a WSTK and the following software requirements:

* Simplicity Studio v4
* EmberZNet 6.6.3
* GNU ARM v7.2.1

# Introduction

## Purpose

This tutorial will give an overall knowledge about how to build a Light and Switch device from the scratch. For the end of the Lab, the user will be familiar with the Simplicity Studio, fundamental needs to make an SoC to work, SDK source architecture, the key points of EmberZNet stack, events, non-volatile memory using.

The network will consist of two devices by using board of BRD4162A (EFR32MG12).

One of them is the Light. Since the realized network is centralized, it will work as the Coordinator and Trust Center of the network. This device forms- and opens the network, permits other devices to join, and manages the security keys. It will turn Off- or On its LED according to the received command by the Switch.

The other device is the Switch. It joins to the opened network and send On-Off commands to the Light. It has a periodic event to execute any user code, moreover it can store any custom data into the flash by using non-volatile memory.

## Application features

To realize the above detailed functionalities, the application development is split into four steps to show how an application should be built up from the beginning.

In the 1st phase, a basic network forming by the Light-, and a joining process by the Switch will be realized.

The 2nd part will prepare the devices to transmit-, receive-, and process the On-Off commands by using APIs.

At the 3rd step the Switch will have a periodic event to execute any custom code, which will be a LED blinking in our case.

The 4th thing to do is to make the Switch to be able to store any custom data in its flash by using Non-volatile memory.

Before all the individual steps would be performed, it’s necessary to check some basics to avoid unwanted issues during the development.

# Fundamental steps

Regardless of the created application or device type, there are some general steps that must be done prior to start the development.

## Check Tools

Make sure you have installed the EmberZNet 6.6.3 SDK and GCC toolchain on your PC.

### Check EmberZNet SDK

1. Launch Simplicity Studio v4.
2. *Window*🡪*Preference*🡪*Simplicity Studio*🡪*SDK*s, make sure “EmberZNet 6.6.3” is installed.

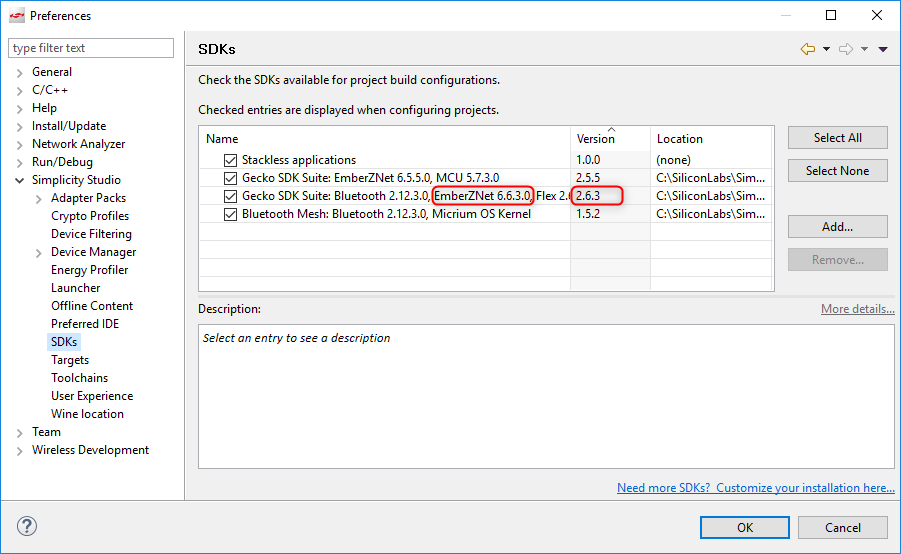
It is part of the Gecko SDK Suite 2.6.3, therefore it doesn’t appear itself. See Figure 2‑1.

Figure 2‑1  
Check installed EmberZNet SDK

### Check Toolchains

1. *Windows*🡪*Preference*🡪*Simplicity Studio*🡪*Toolchains*, make sure GCC toolchain is installed.

It is important to use the same toolchain version when building your project that was used to build the libraries supplied as part of the SDK. The list of the proper toolchain-SDK pairing can be found [here](https://www.silabs.com/community/software/simplicity-studio/knowledge-base.entry.html/2018/08/22/gecko_sdk_suite_tool-qlc4). See Figure 2‑2.

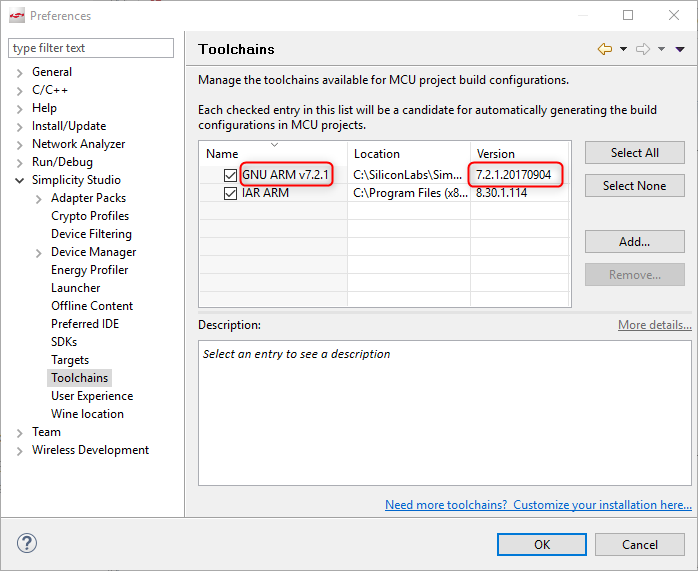


Figure 2‑2  
Check the Toolchain

## Using Gecko Bootloader

A bootloader is a program stored in reserved flash memory that can initialize a device, update firmware images, and possibly perform some integrity checks. If the application seems to do not running, always check the bootloader, because lack of it causes program crash.

There are two possible way to have a bootloader application.

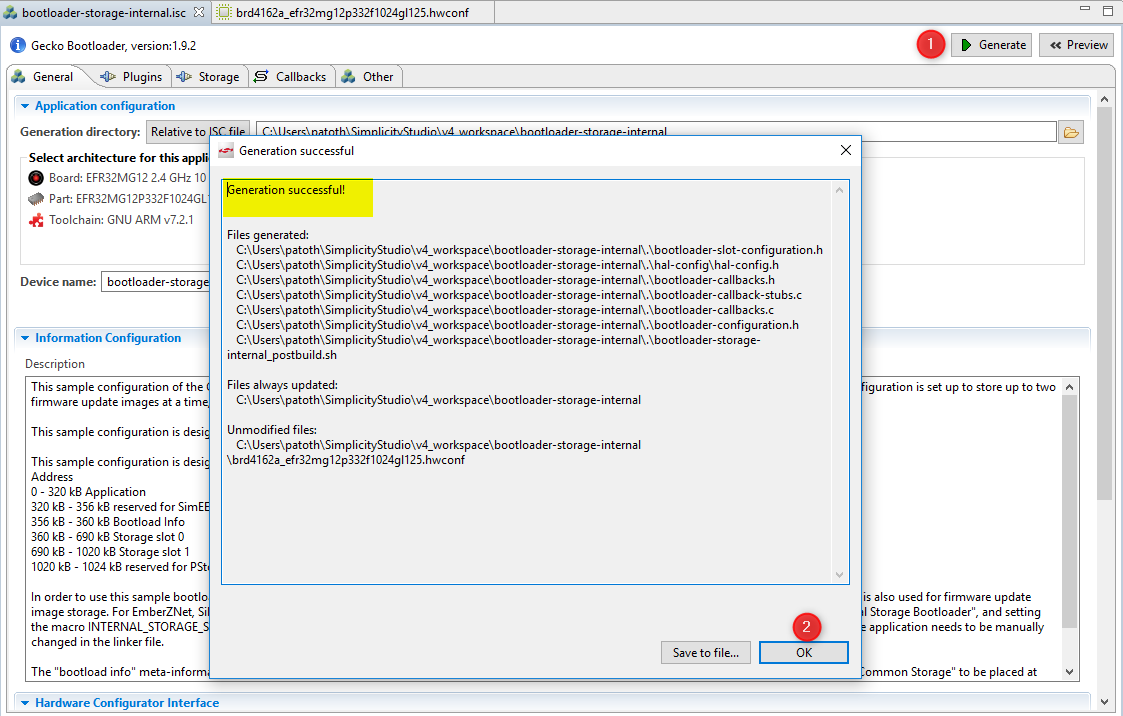
1. Use Silicon Labs pre-built image (not for all boards)
2. Create your own bootloader project

The 1st option is the easiest solution. Each Gecko SDK contain pre-built bootloader images for different boards. The most suitable for a Zigbee application is the “Internal Storage Bootloader (single image on 1MB devices)”. It fits for the flash size of the device, furthermore it can contribute in OTA firmware update. If there is no any special requirement regarding the bootloader, I recommend using this.

It can be found at “c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.6\platform\bootloader\sample-apps\bootloader-storage-internal-single\efr32mg12p332f1024gl125-brd4162a\”.

The 2nd way is to use the AppBuilder to create-, generate-, and build your own application. It is possible to customize and add new features to it, but the current lab doesn’t detail these possibilities.

1. Go to File -> New -> Project. This will bring up the New Project Wizard
2. Select “Silicon Labs AppBuilder Project”. Click Next.
3. Select “Gecko Bootloader”. Click Next.
4. Select the latest version. (Gecko Bootloader 1.9.2). Click Next.
5. Select “Internal Storage Bootloader (single image on 1MB devices)”. Click Next.
6. Name your project (Whatever name you want). Click Next.
7. **Select board** and compiler. Then finish.
8. The new project should have been created now, with the project configuration file (an .isc file) open.
9. Click “**Generate**”. Notice the project files appearing in Project Explorer. A window saying Generation successful will appear. See Figure2‑3.

Figure 2‑3  
Generating the project sources

1. Select the project in Project Explorer window and compile your project by clicking on the Build icon. Ensure that the build completes without any errors.

The output “bootloader-storage-internal-single-combined.s37” file should be downloaded to **both** target device. I recommend to use the .s37 or .hex, since these files already contain the address where to flash.

Use the Simplicity Commander or Flash Programmer to download the output file. See Figure 2‑4.

The gecko bootloader has 2 stages. Since the <project\_name**-combined**>.s37/.hex includes both stage, I recommend to use this. See Figure 2‑4.

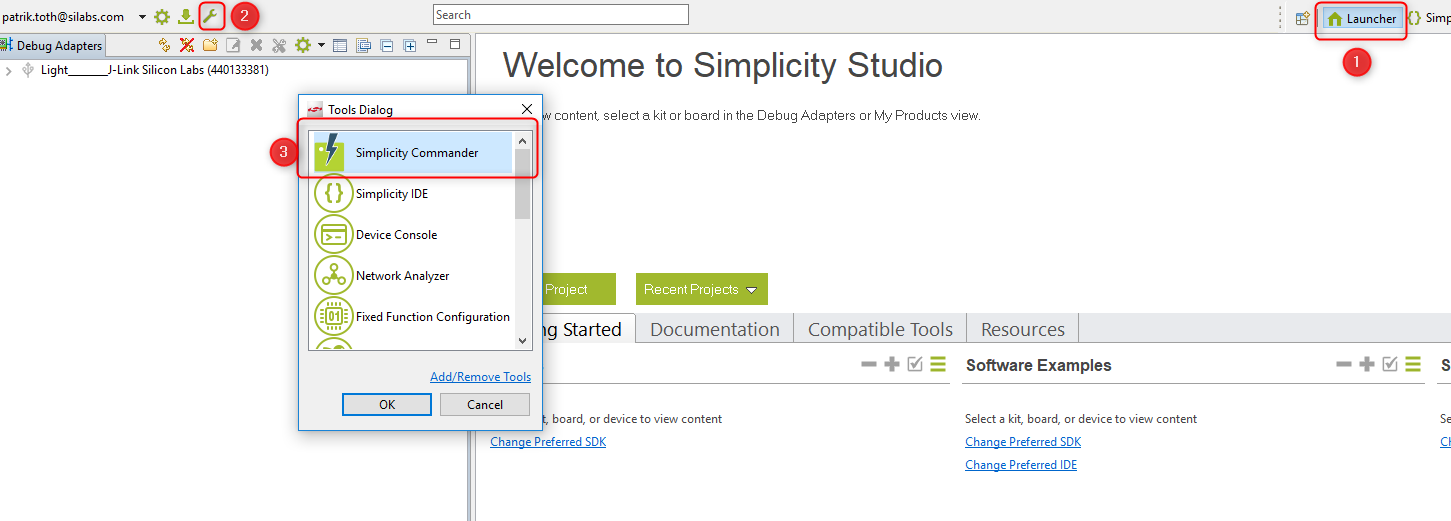


Figure 2‑4  
Open Simplicity Commander

To flash the image, connect to the Adapter, then to the Target. Browse the image at the “Flash” tab and press “Flash” button.

For more information about Gecko Bootloader, please find  
[UG266: Silicon Labs Gecko Bootloader User’s Guide](https://www.silabs.com/documents/public/user-guides/ug266-gecko-bootloader-user-guide.pdf)  
[UG103.6: Bootloader Fundamentals](https://www.silabs.com/documents/public/user-guides/ug103-06-fundamentals-bootloading.pdf)  
[AN1084: Using the Gecko Bootloader with EmberZNet and Silicon Labs Thread](https://www.silabs.com/documents/public/application-notes/an1084-gecko-bootloader-emberznet-silicon-labs-thread.pdf).

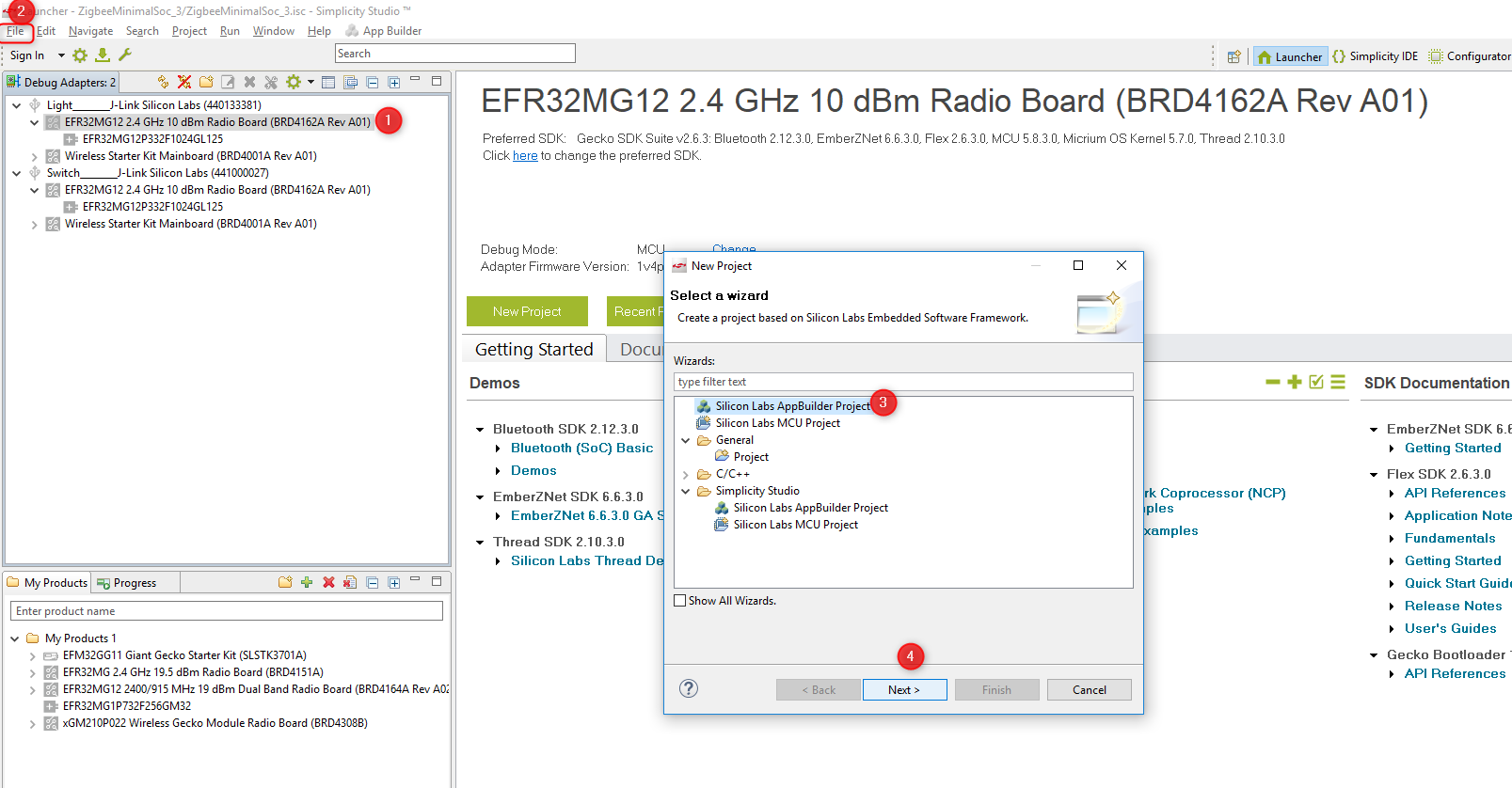
# Create Light application

After the previous steps have been done, it’s time to realize the 1st feature of the Light device. As discussed before, the Light should be able to form-, and open the network.

Similarly to the bootloader project, the AppBuilder is needed to create the application. Before the builder would be opened, I recommend to select the target board on the left side of the Launcher view. It helps to the AppBuilder to recognize the target device, thus the proper board related configurations (peripherals, pins) are automatically applied.

To open it,

1. Go to File -> New -> Project. This will bring up the New Project Wizard. See Figure 3‑1.

Figure 3‑1  
Open AppBuilder

1. Select “Silicon Labs Zigbee”. Click Next. See Figure 3‑2.

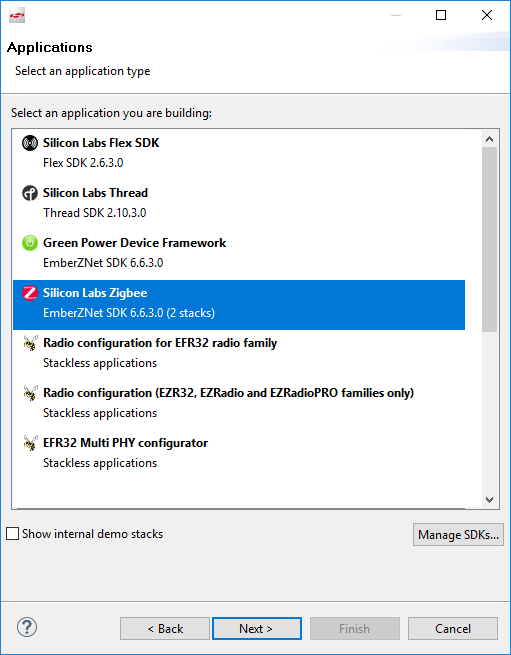


Figure 3‑2  
Select the stack type.

1. Select “EmberZNet 6.6.3 GA SoC 6.6.3.0”. Click Next. See Figure 3‑3.

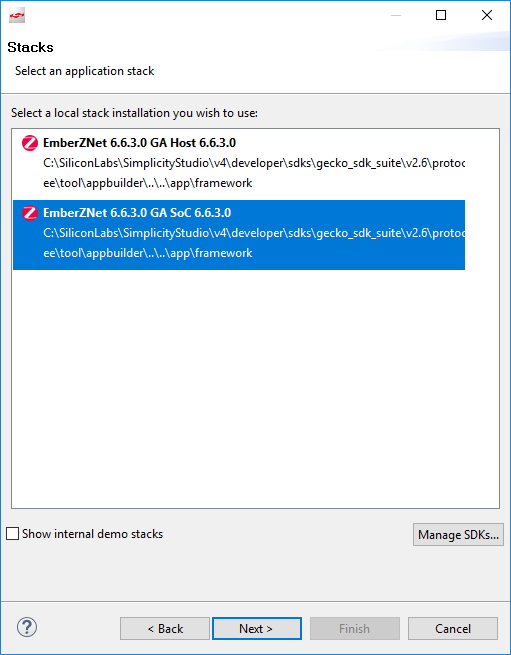


Figure 3‑3  
Select stack version and SoC application type.

1. Choose the “ZigbeeMinimal” sample application. Click Next. See Figure 3‑4.

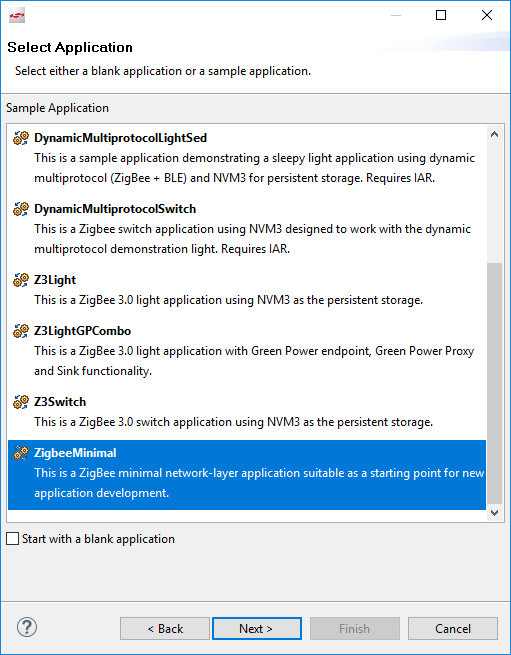


Figure 3‑4  
Select ZigbeeMinimal sample application.

1. Name your project to “Zigbee\_Light\_ZC”. Click Next. See Figure 3‑5.

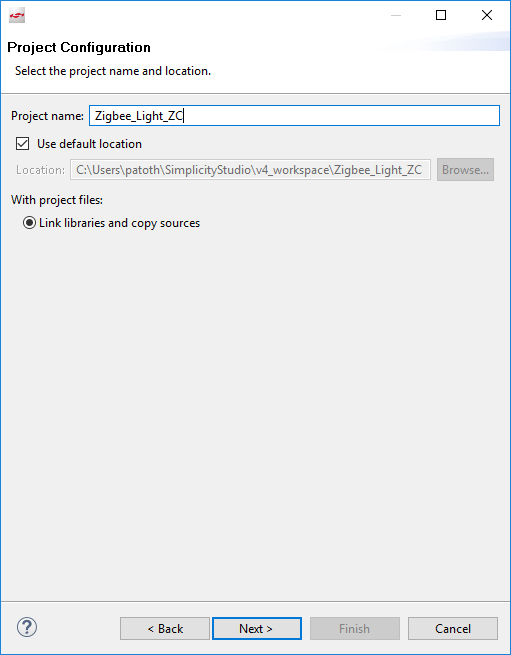


Figure 3‑5  
Name the project.

1. In next window (Project Setup), double check the board is BRD4162A, and “GNU ARM v7.2.1” compiler. Click Finish. See Figure 3‑6.

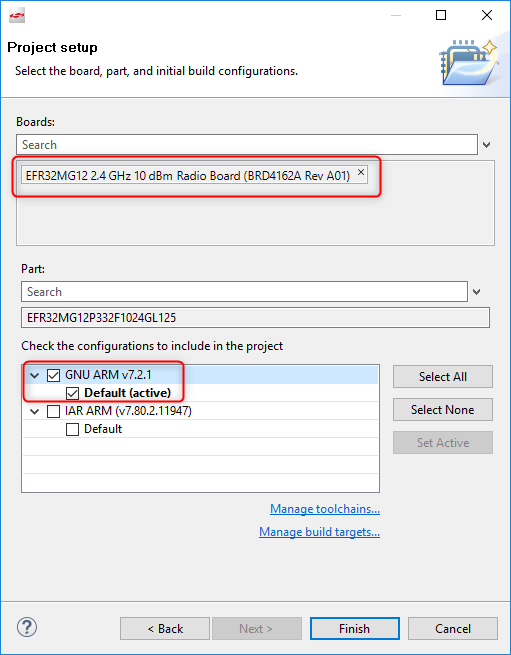


Figure 3‑6  
Check the board and compiler.

At this point the project is placed into the default workspace directory, but most of the source files are missing. These files will be later linked or generated according to the AppBuilder settings. To open the AppBuilder, click double to the “Zigbee\_Light\_ZC.isc” file.

There are multiple tabs in the file. Have a closer look at each tab.

## General

This page gives information about the current project configuration, its path, furthermore, shows the selected toolchain and board. Nothing to do with this tab. It’s important to mention that in case of changing the toolchain or the board, always create a new project rather than modify the project settings.

## ZCL Clusters

One of the most important setting is the ZCL configurations. The type of the device is based on its clusters and attributes. The Silicon Labs pre-defined most of the available device types. In our tutorial it’s a “HA Light On/Off Light” kind of device. To enable all the mandatory clusters and attribute for a Light, click on the “ZCL device type” dropdown menu, then select “HA Light On/Off Light” template. See Figure 3‑7.

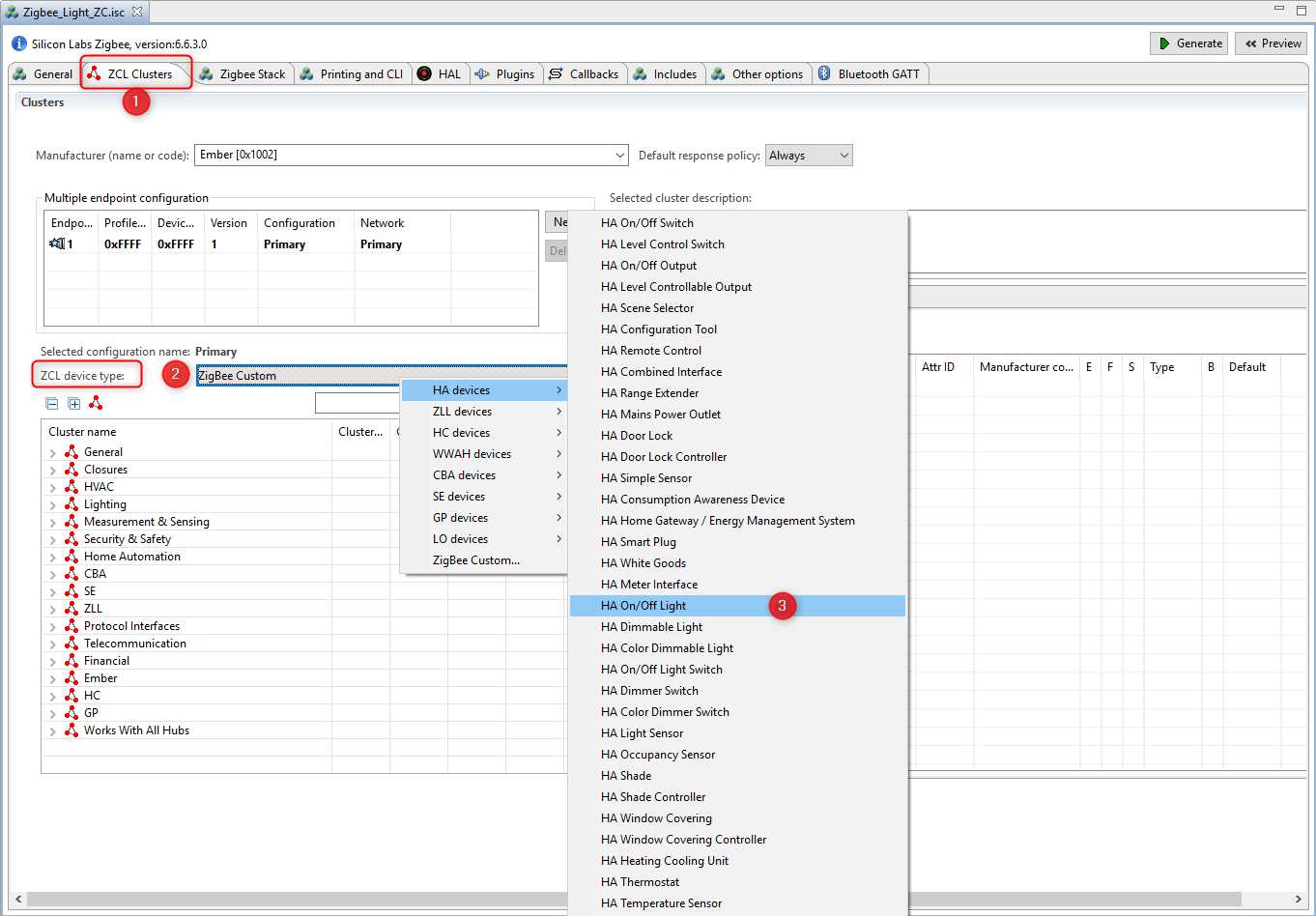


Figure 3‑7  
Select ZCL device type.

After selecting the template, new enabled clusters and attributes are appeared in the list, moreover, the endpoint configuration is changed. These settings are applied based on the Zigbee Specification.

**It’s important to mention that the ZCL selection in not strictly mandatory for network creating and opening!  
This step prepares the device to be able to receive and process the On-Off commands in the 2nd step.**

Note: It’s not possible to modify these templates, therefore the “ZigBee Custom..” should be used if there is need to add any additional cluster.

## Zigbee Stack

This tab lets to change the device type in network aspect. Since the router device cannot form centralized network, the “Coordinator and Router” type must be selected. The default Zigbee 3.0 Security is appropriate. See Figure 3‑8.

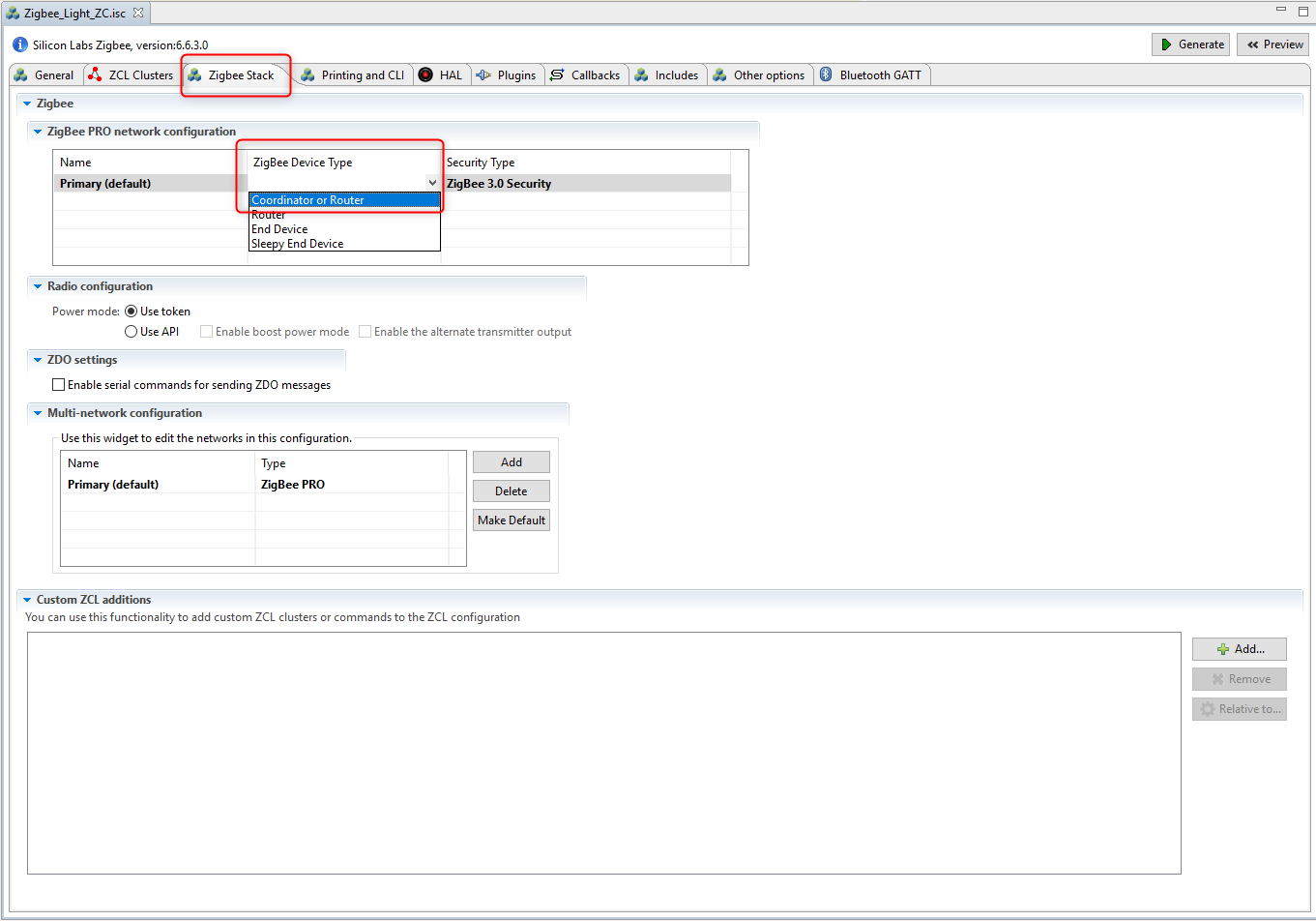


Figure 3‑8  
Change device type to Coordinator.

The rest of the settings should not be modified, because the device operates on Single network with basic clusters.

## Printing and CLI

Usually the default setting is enough in this Lab. The only thing to do is verify the “Enable debug printing” box is enabled, and check-in the “On off cluster” debug prints to get more information later. See Figure 3‑9.

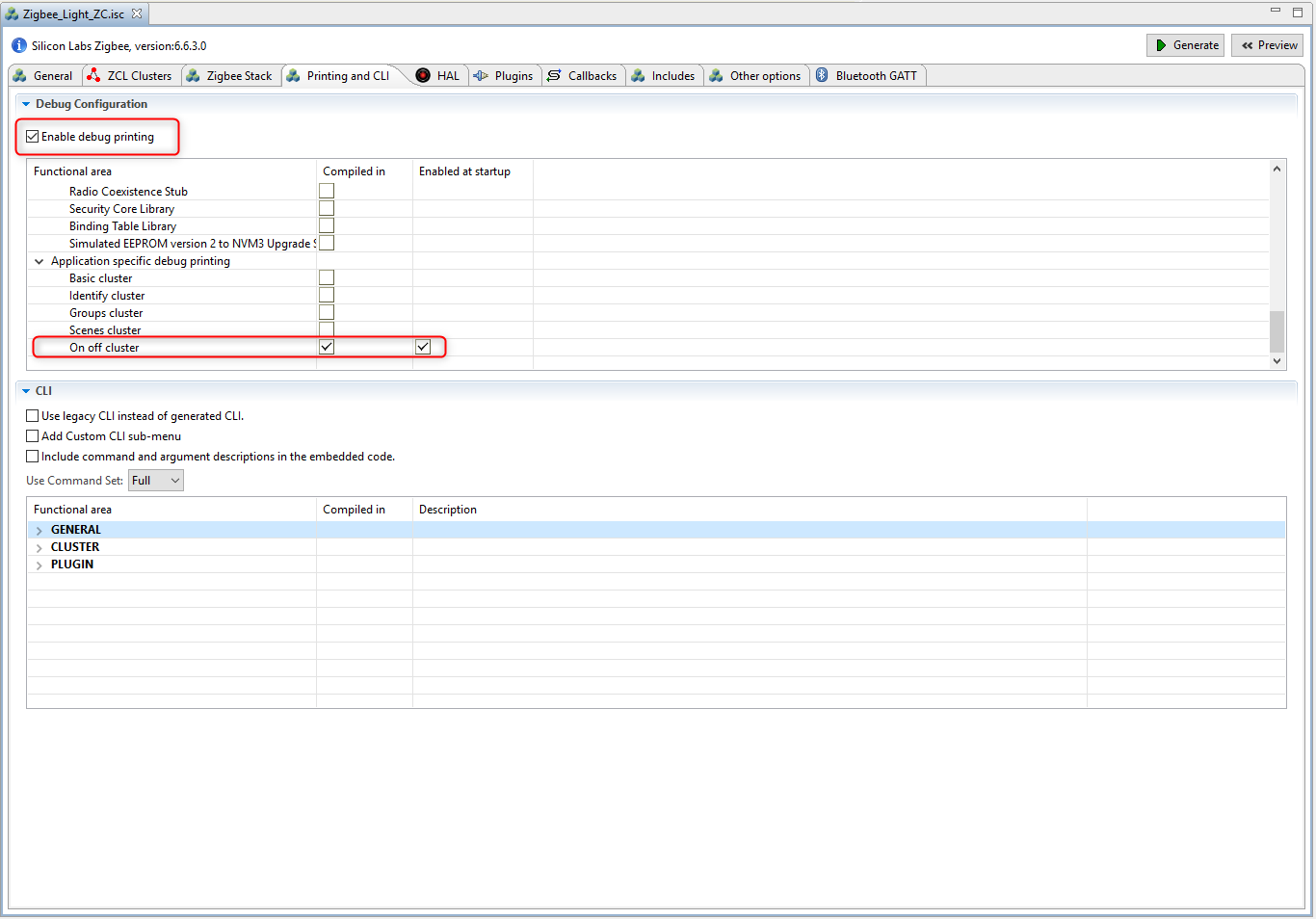


Figure 3‑9  
Debug printing.

Note: Like in **3.2 ZCL Clusters** part, the “On off cluster” debug print also serves the later implemented features.

## HAL

This tab is modified quite rarely. It would be possible to use external hardware configurator and change bootloader type, but it’s rather exists for legacy purposes.

## Plugin

The plugins are individual software packages which implement a functionality. A plugin can consist of libraries and source files as well. These are collected on this tab, and the selection of device type doesn’t filter out the plugins that the device cannot use, thus it must be done manually. For example, this sample application doesn’t enable the necessary plugins for network forming/opening. The below plugins must be added or removed to get a device which can operate as a Coordinator.

The “Network Creator” and “Network Creator Security” plugins implement the network forming and opening functionality, therefore these are required to have.

The “Security Link Keys Library” provides the APS security key management which is key feature in Zigbee 3.0. This plugin serves the Trust Center functionality.

The “Network Steering” and “Update TC Link Key” can be removed, since the device doesn’t intend to joint to any network.

The “ZigBee PRO Stack Library” includes one of the most complex stack libraries. It contains the routing-, networking-, scanning-, neighbor-, child-handler and other functionalities. It’s mandatory for Coordinator and Router. The sample application uses this plugin by default.

The “Serial” establishes the Command Line Interface (CLI). This interface lets the user to communicate with the SoC. In case of selecting the correct board at project creation phase, the plugin settings should fit to the pinout of the device, but it is also important to double check the values. This application uses UART0 via USB Mini-B Connector. The WSTK Main board has a Board Controller which does the UART-USB conversion. This is the Virtual COM port, which must be enabled separately out of the plugin. It will be detailed later.

Summarized the above, the following table presents the affected plugins. See Table 1.

|  |  |  |
| --- | --- | --- |
| Plugin name | Add/Remove/Check | Importance |
| Network Creator | Add | Required |
| Network Creator Security | Add | Required |
| Security Link Keys Library | Add | Recommended |
| ZigBee PRO Stack Library | Check | Required |
| Serial | Check | Required |
| Network Steering | Remove | Recommended |
| Update TC Link Key | Remove | Recommended |

Table 1  
Plugins to check.

Before going ahead, it’s a good place to point how the users can find more information about the plugins. As mentioned above, some plugins have source files, not just pre-built libraries. These files can be examined to find some not detailed information about its internal working. The header-, and source files can be found at “C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.6\protocol\zigbee\app\framework”, under “plugin”, “plugin-host” and “plugin-soc” folders. This separation is used to identify the commonly used, SoC and Host specific plugins.

These files are available from the AppBuilder as well, but some extra information can be found, as the implemented-, defined callbacks and APIs by the plugin. See Figure 3‑10.

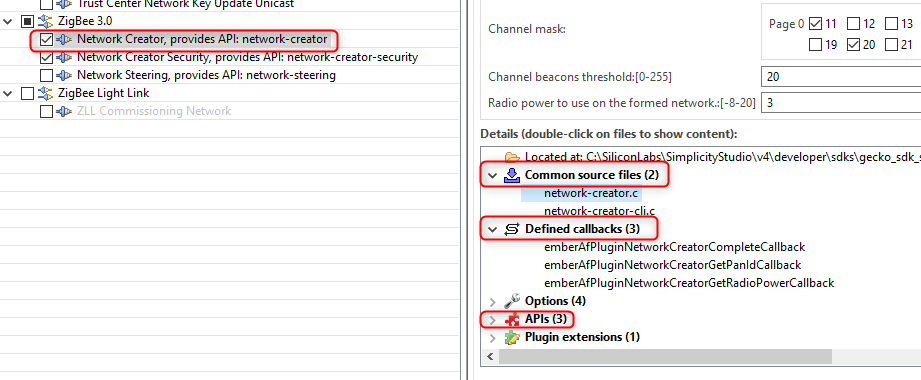


Figure 3‑10  
Plugin details.

## Callbacks

The callbacks are a set of functions for the implementation of the application level functionalities. Some of them are related to plugins, while others can be used without any limitation. This tab is dynamically changing based on the previous *Plugins* and *ZCL Clusters* tab. It means some callbacks are visible/usable only if the appropriate plugin or cluster has been enabled.

It’s not necessary to use any callback for the basic network forming and opening functionalities. It will be used later.

## Includes

Project specific macros and include paths are defined here. It should not be modified, unless the user would use any custom token, or event. It will be used later.

## Other options

Advance settings in case of using dual band functionalities. It’s not used in this project.

## Bluetooth GATT

The Zigbee-BLE Dynamic Multiprotocol bluetooth side configurator is resided into the AppBuilder, but it’s not used in this project.

Note: Some BLE related plugin make it editable.

After saving the modifications the .isc file ready to generate the project files and link the necessary SDK sources and libraries.

Press the Generate button. See Figure 3‑11.

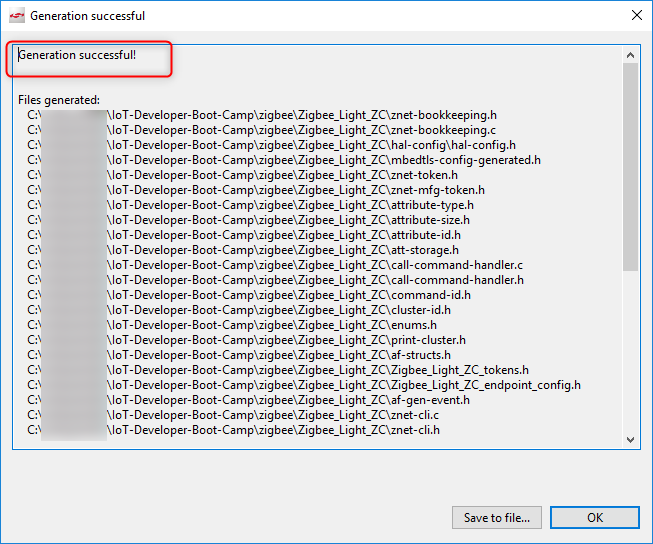


Figure 3‑11  
Generation result.

The “Generation successful” label signs all the required files are created.

## Hardware configurator

The hardware configurator is NOT part of the AppBuilder. It’s unique file in the project for generating the “hal-config/hal-config.h” file. This file contains includes, which will used by other source files.

It’s important to understand that the here applied settings don’t directly mean that the peripheral is initialized, it just provides macros for the proper pin and clock settings. It could happen that the UART peripheral is enabled in this configurator, but not in the Serial plugin. In this case the initializer functions will not be called, thus the UART won’t work. However, the other way around, if a plugin refers to a macro which is not defined by the hardware configurator, it causes compiler errors.

In our project, the VCOM enable pin must be enabled to make the UART-USB converter works. The serial port initializer sets the PA5 to high state. If the board was selected correctly at the begging, this already has been set. See Figure 3‑12.

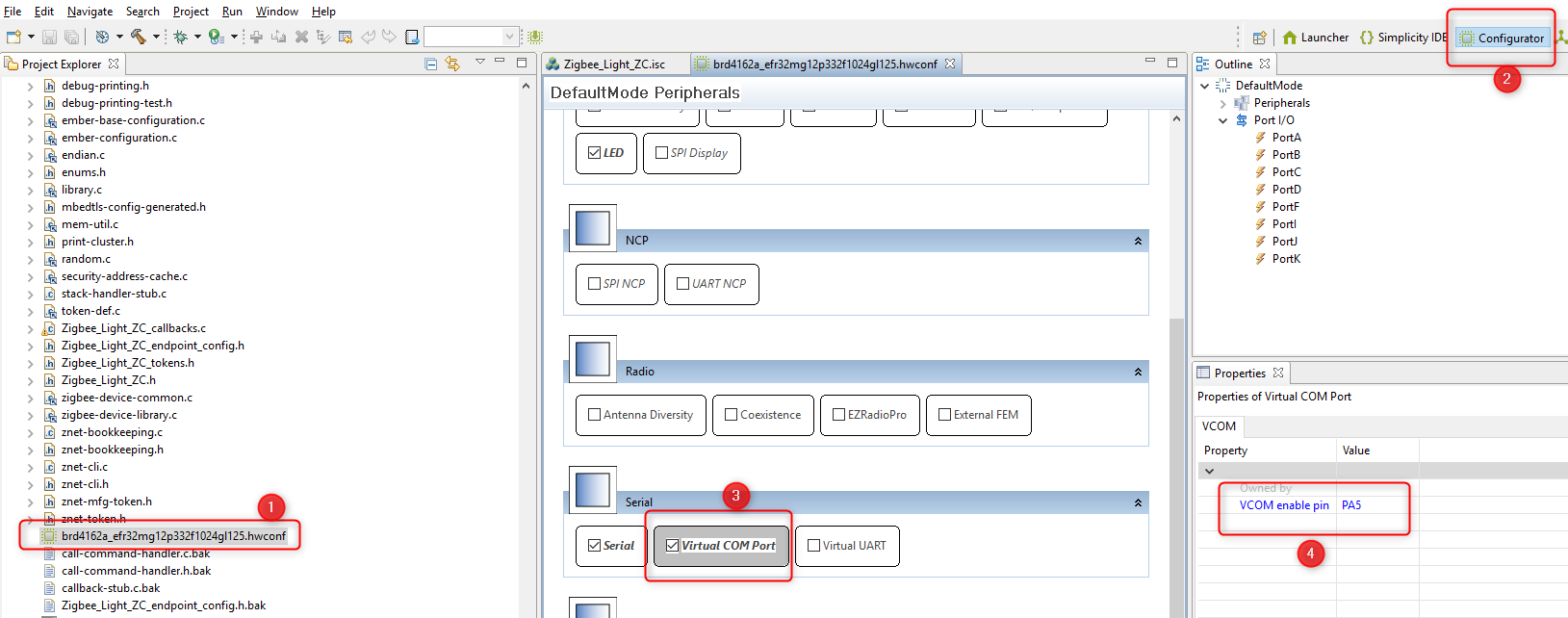


Figure 3‑12  
Hardware configurator.

The saving of this file re-generates the “hal-config.h” file according to the settings.

Press the Build button ( ). Upon a successful build, the binary files should be appeared in the “Binaries” directory.

# Download and test the Light application

Let’s download the *Zigbee\_Light\_ZC.s37* file as shown below. See Figure 4‑1 and Figure 4‑2.

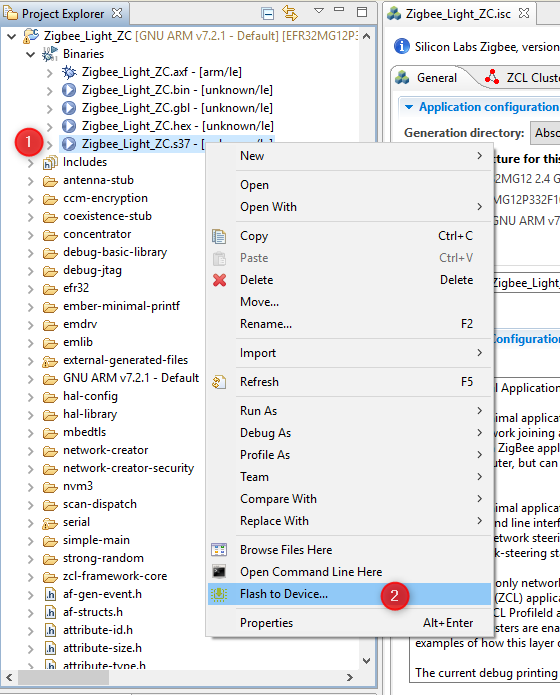


Figure 4‑1  
Open Flash Programmer.

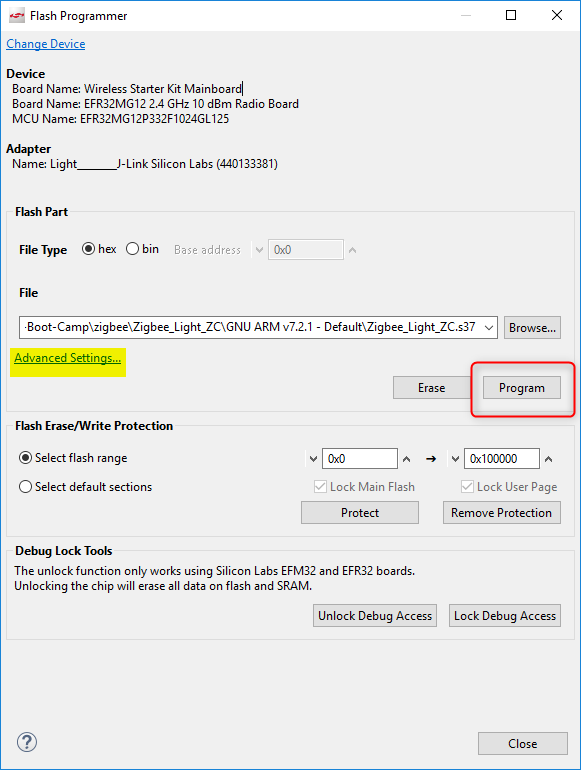


Figure 4‑2  
Download the image.

The highlighted “Advanced Settings..” provides possibility to decide how to flash the chip. Here the flash can be merged with new image (Merge Content), partially- (Page Erase) or completely (Full Erase) erased before downloading the file.

Keep in mind that neither erase type clean the bootloader section, but the Full erase deletes the region of the tokens.

After the image has been downloaded, it’s possible to communicate with the device. For this purpose, open the Launch console, which is a built-in serial port terminal in the Studio. See Figure 4‑3.

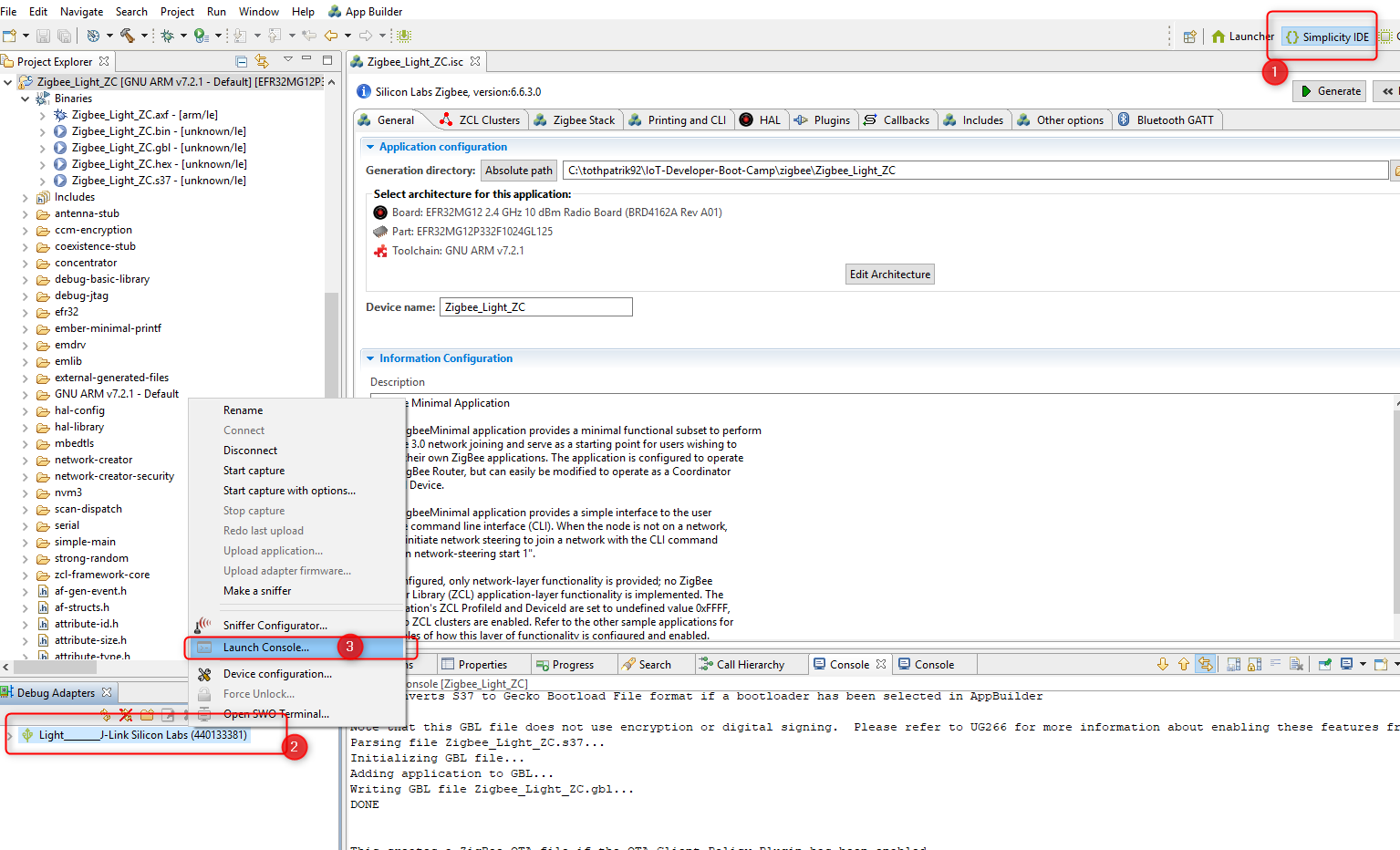


Figure 4‑3  
Open Serial console.

If the serial console is opened, switch to “Serial 1” and press “Enter”. See Figure 4‑4.

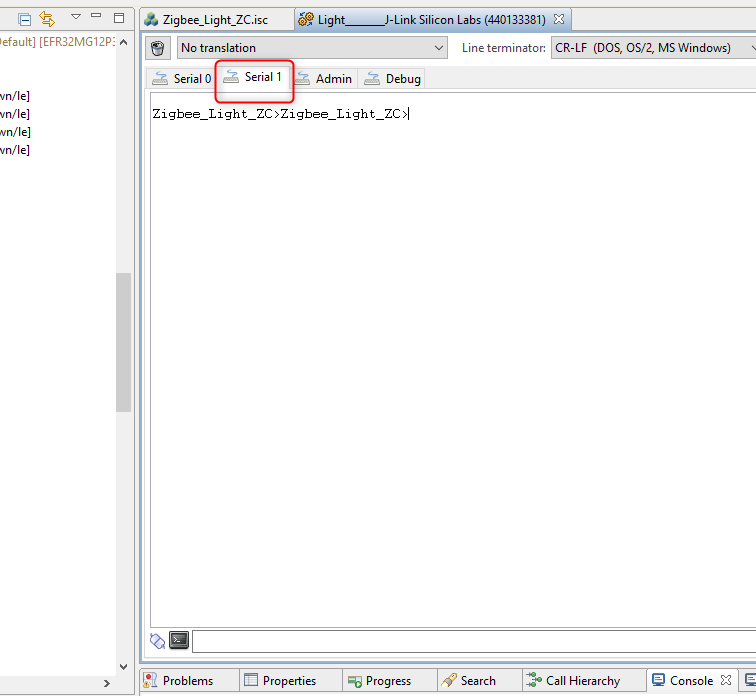


Figure 4‑4  
Select Serial 1 tab.

The “\n\r” characters triggers the project name printing. This basic test shows that the RX and TX of the CLI is working correctly.

If the same text is printed, put a bit away the Light application and start to create the Switch.

# Create Switch application

The 1st step of the Switch is to be able to join to the network what is opened by the Light.

The creating of the project and configuration way of the AppBuilder are the same as in case of the Light application, therefore this chapter will include a bit less figure than the Light.

The project also based on the “ZigBeeMinimal” sample application, so please

1. Repeat the step 1-6 of chapter **3 Create Light application**, except name the project to “Zigbee\_Switch\_ZR”.
2. Open the .isc file of the project.
   1. Go to *ZCL Clusters* tab and choose **“HA On/Off Switch”** device template.
   2. Go to *Zigbee Stack* tab and select the **“Router”** device type from the dropdown menu.
   3. Go to *Printing and CLI* tab and double check the “Enable debug printing” is turned on.
   4. Go to *Plugins* tab and double check the below plugins are enabled

* Serial
* Network Steering
* Update TC Link Key

Have a closer look at the enabled plugins.

The *Serial* has already been discussed at the Light. It’ required for the CLI.

The *Network Steering* plugin serves to discover the existing networks on the enabled channels. The device issues a Beacon Request message and listens the responses. If the Beacon response (from ZC or ZR) is received with set “permit association” flag, the device starts the joining process for the network, otherwise continue the scanning.

The *Update TC Link Key* is used to request new APS Link Key from the Trust Center. It should be enabled since the Light (Trust Center) has the Security Link Keys Library.

1. Press *Generate* button
2. Verify the VCOM enable is enabled in the Hardware Configurator (likewise to **3.11 Hardware configurator**)
3. Build the project

# Download and test the Switch application

Please repeat the steps from the chapter **4 Download and test the Light application** and test it. See Figure 6‑1.

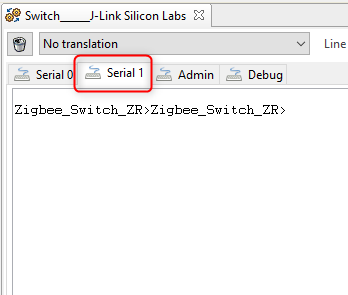


Figure 6‑1  
CLI testing.

# Establish connection between Light and Switch

The 1st phase.

This chapter presents how to form a network and join to this. The communication between the devices will be captured by Network Analyzer tool.

For the 0th step, open both Light and Switch serial console, and type the “echo 1” command. It helps to track the released command which makes the log more understandable.

Please perform the following operations:

## Create a centralized network

|  |
| --- |
| Zigbee\_Light\_ZC> plugin network-creator start **1** |

Result: 0x00 means “EMBER\_SUCCESS”. Find more status codes [here](https://docs.silabs.com/zigbee/latest/em35x/group-status-codes).

|  |
| --- |
| NWK Creator: Form: 0x00  Zigbee\_Light\_ZC>NWK Creator Security: Start: 0x00  EMBER\_NETWORK\_UP 0x0000  NWK Creator: Form. Channel: 25. Status: 0x00  NWK Creator: Stop. Status: 0x00. State: 0x00 |

## Find network and device information

|  |
| --- |
| Zigbee\_Light\_ZC>info |

Result:

|  |
| --- |
| MFG String:  AppBuilder MFG Code: 0x1002  node [(>)000B57FFFEDEA657] chan [25] pwr [3]  **panID [0xD216]** nodeID [0x0000] xpan [0x(>)FD7B0901B45A9C4E]  parentID [0xFFFF] parentRssi [0]  stack ver. [6.6.3 GA build 151]  nodeType [0x01]  Security level [05]  network state [02] Buffs: 75 / 75  Ep cnt: 1  ep 1 [endpoint enabled, device enabled] nwk [0] profile [0x0104] devId [0x0100] ver [0x01]  in (server) cluster: 0x0000 (Basic)  in (server) cluster: 0x0003 (Identify)  in (server) cluster: 0x0004 (Groups)  in (server) cluster: 0x0005 (Scenes)  in (server) cluster: 0x0006 (On/off)  Nwk cnt: 1  nwk 0 [Primary (pro)]  nodeType [0x01]  securityProfile [0x05] |

## Find the Network key for capturing

|  |
| --- |
| Zigbee\_Light\_ZC> keys print |

Result:

|  |
| --- |
| EMBER\_SECURITY\_LEVEL: 05  NWK Key out FC: 00000013  NWK Key seq num: 0x00  NWK Key: 3F B8 D4 09 4E AD 0A 83 89 A2 7F 1F C0 03 BF 87  Link Key out FC: 00000000  TC Link Key  Index IEEE Address In FC Type Auth Key  - (>)000B57FFFEDEA657 00000000 L y C0 1C 81 33 69 98 A3 21 D4 67 92 29 73 59 D0 8E  Link Key Table  Index IEEE Address In FC Type Auth Key  0/6 entries used.  Transient Key Table  Index IEEE Address In FC TTL(s) Flag Key  0 entry consuming 0 packet buffer. |

## Add network key to Network Analyzer

Copy yellow highlighted network key and add it to the Network Analyzer’s key storage to be able to decode the messages. See Figure 7‑1.

* 1. Open Window->Preferences

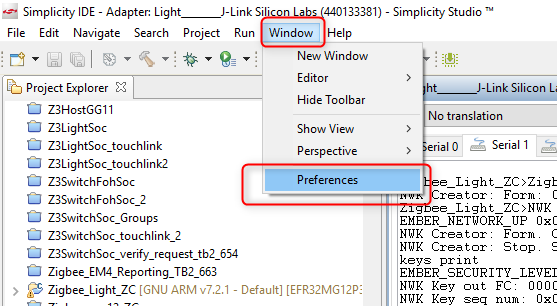


Figure 7‑1  
Open Security Keys tab.

* 1. Navigate to Network Analyzer->Decoding-> Security Keys and add the new key. See Figure 7‑2.

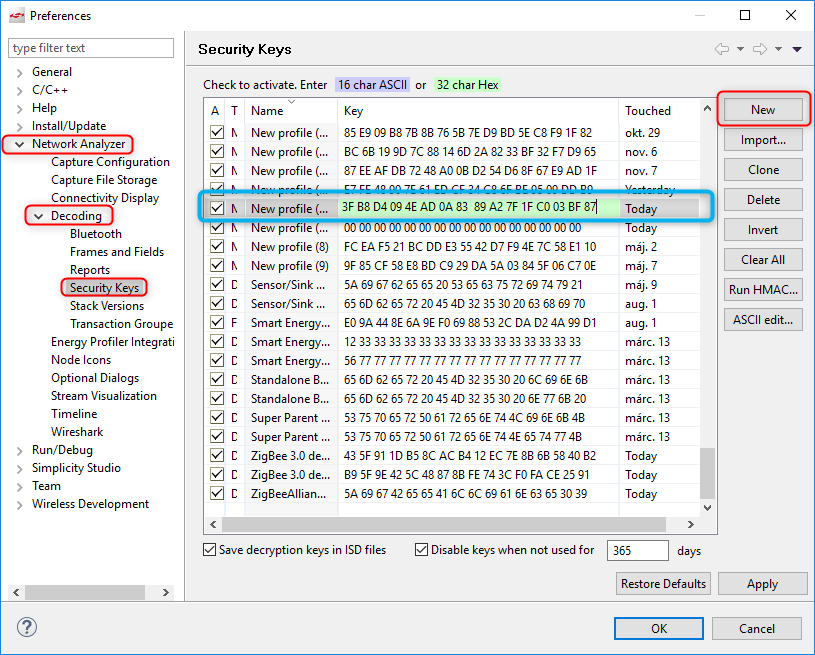


Figure 7‑2  
Add new Network Key.

## Start capturing on Light device

Right click on Adapter name of the Light-> *Connect* (if not connected yet)->*Start capture*. See Figure 7‑3.

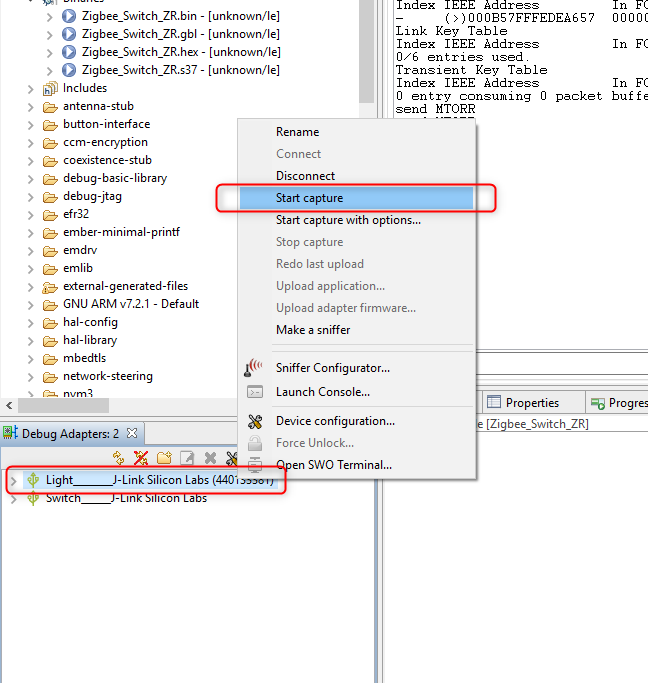


Figure 7‑3  
Start capturing.

It should change the perspective to *Network Analyzer* and immediately start capturing. See Figure 7‑4.

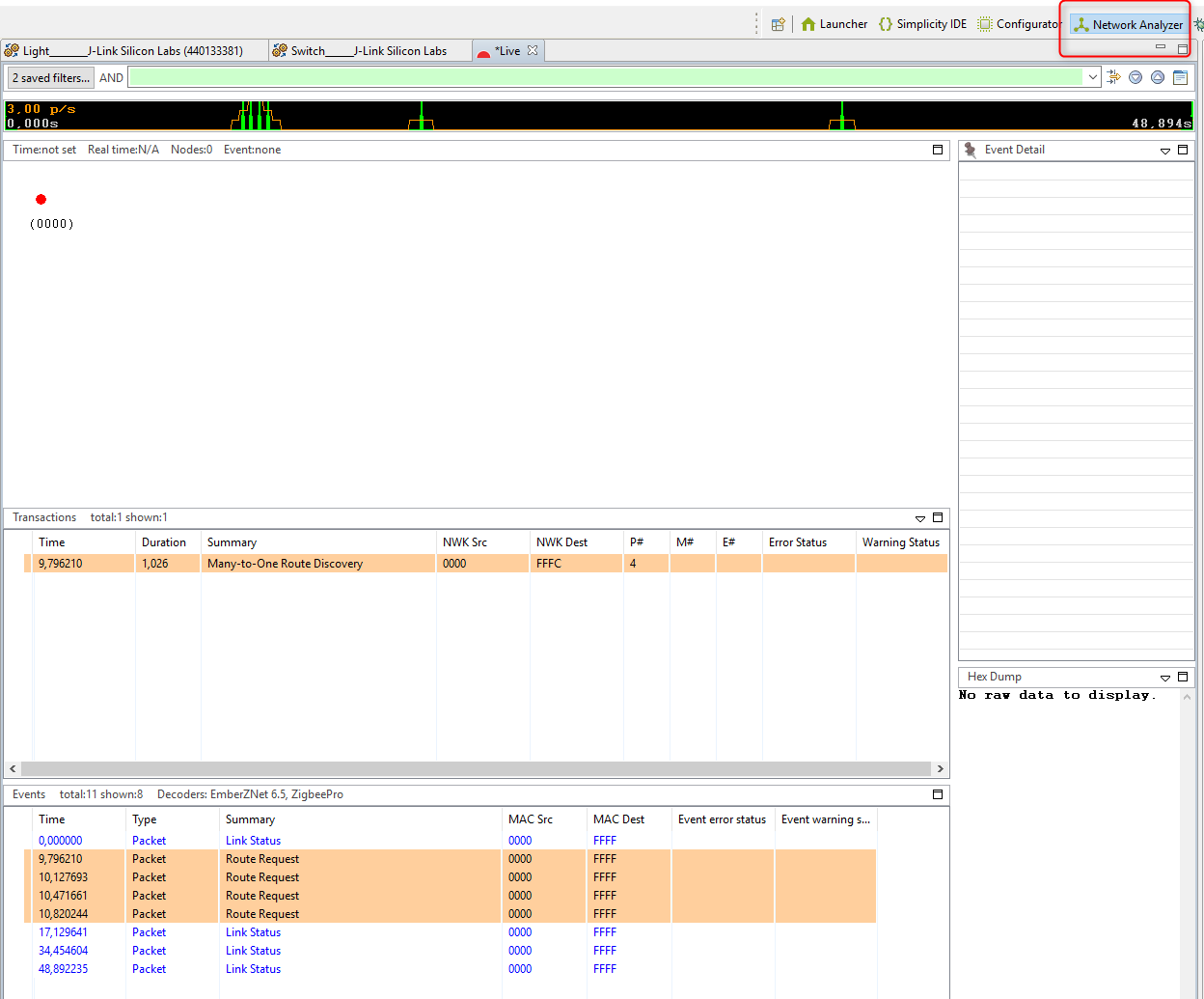


Figure 7‑4  
Capturing on Light.

The capture file (\*Live) should show the packets on the network.

## Open the network

Go back to the Serial console of the Light and permit devices to join.

|  |
| --- |
| Zigbee\_Light\_ZC>plugin network-creator-security open-network |

Result:

|  |
| --- |
| NWK Creator Security: Open network: 0x00  Zigbee\_Light\_ZC>pJoin for 254 sec: 0x00  NWK Creator Security: Open network: 0x00 |

This command triggers the Light to send a “Permit Join Request” broadcast message to every FFD.

By default, the network will be opened for 300 seconds.

## Join the Switch

Join to this network with the Switch device with TC Link key update.

|  |
| --- |
| Zigbee\_Switch\_ZR>plugin network-steering start 0 |

Result:

|  |
| --- |
| NWK Steering State: Scan Primary Channels and use Install Code  Error: NWK Steering could not setup security: 0xB5  NWK Steering State: Scan Secondary Channels and use Install Code  Error: NWK Steering could not setup security: 0xB5  NWK Steering State: Scan Primary Channels and Use Centralized Key  Starting scan on channel 19  NWK Steering: Start: 0x00  Zigbee\_Switch\_ZR>Starting scan on channel 20  Starting scan on channel 24  Starting scan on channel 25  NWK Steering joining 0xD216  EMBER\_NETWORK\_UP 0xDA42  NWK Steering network joined.  Processing message: len=12 profile=0000 cluster=0013  RX: ZDO, command 0x0013, status: 0x00  Device Announce: 0xDA42  Update TC Link Key: Starting update trust center link key process: 0x00  Processing message: len=17 profile=0000 cluster=8002  RX: ZDO, command 0x8002, status: 0x00  RX: Node Desc Resp, Matches: 0x0000  Update TC Link Key: New key established: 0x03  Partner: 57 A6 DE FE FF 57 0B 00  NWK Steering: Trust center link key update status: 0x03  Update TC Link Key: New key established: 0x65  Partner: 57 A6 DE FE FF 57 0B 00  NWK Steering: Trust center link key update status: 0x65  pJoin for 180 sec: 0x00  NWK Steering: Broadcasting permit join: 0x00  NWK Steering Stop. Cleaning up.  Join network complete: 0x00 |

## Joining process in Network Analyzer

Have a look at the Network Analyzer how the joining process works. See Figure 7‑5.

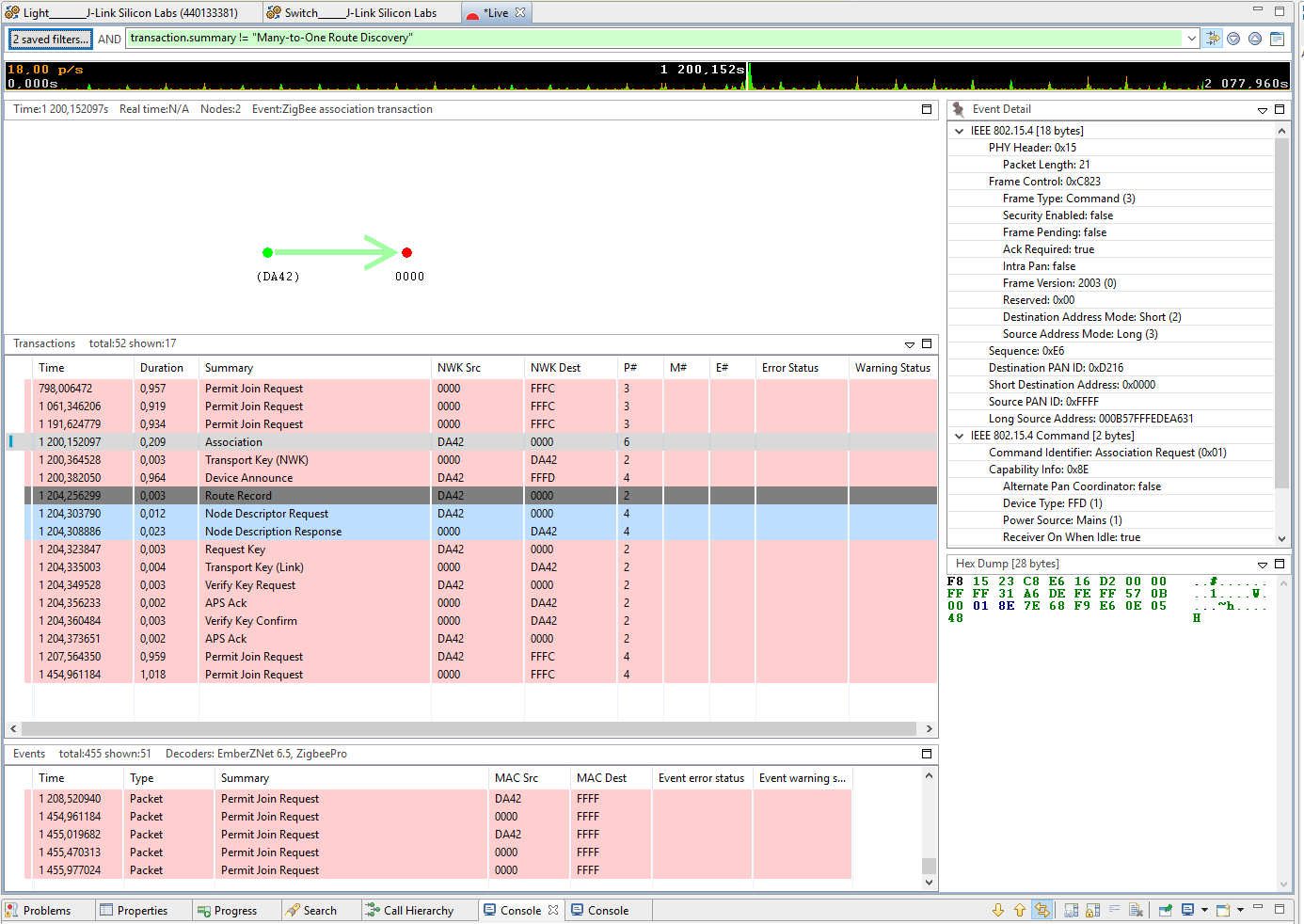


Figure 7‑5  
Joining process in Network Analyzer.

Note: Probably a lot of “Many-to-One Route Discovery” appear in the log. The upper green filter box can be used to filter these messages out. Right click on this package and “Show only summary: Many…..”, then negate the condition from “==” to “!=”.

The 1st phase is done. Both devices are on the same network and now they can exchange messages.

Let’s step to the 2nd phase.

# Sending On/Off/Toggle command

The 2nd phase.

Now the devices are ready to transmit and receive data on the network, although these messages are not handled by default. The Light application should know what to do with the incoming commands and execute the appropriate operations.

In this application the Light should turn on/off- or toggle LED1 based on the received data.

The Switch device should send one of these commands based on which button has been pressed.

Our task is to prepare the devices for these features.

## Command handling on Light device

To become aware of any received command from the user application level, the Callback functions should be used.

These functions can be enabled in the Callbacks tab of the AppBuilder.

Open this tab, find and enable the On-Off-Toggle callbacks under “General/OnOff Cluster” menu. See Figure 8‑1.

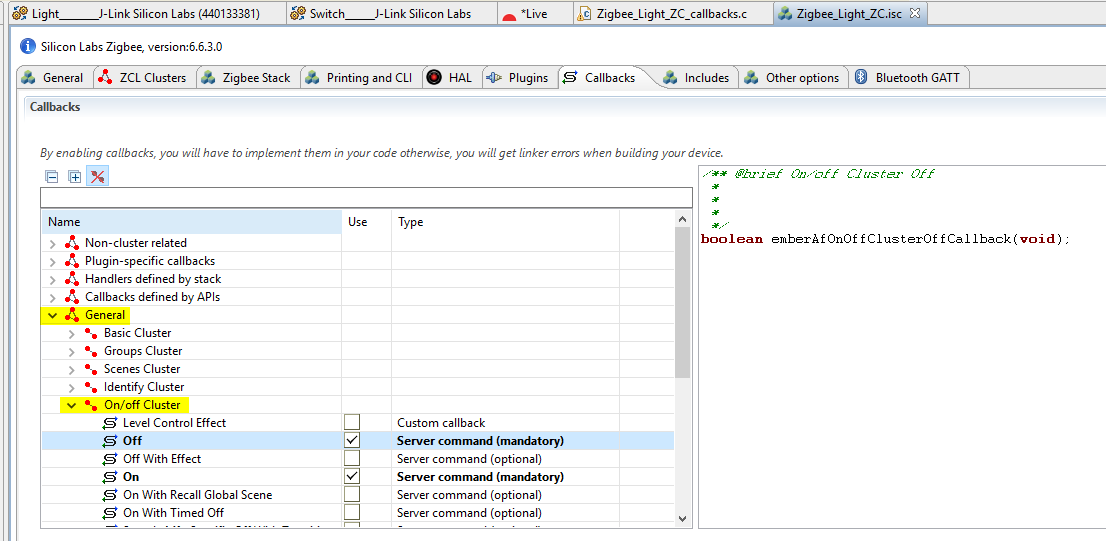


Figure 8‑1  
On/Off Cluster callbacks enabling.

Save the modified .isc file and press *Generate*.

Maybe noticed that the *<ProjectName>\_callbacks.c* is not overwritten at re-generating time, but the *callback-stub.c* is. The reason behind this is that all the callbacks which are defined by the ZCL or Plugins could be called by the stack. These should be placed somewhere to avoid compiler errors, even if these callbacks are not used by the user. The *callback-stub.c* serves this purpose. When a callback is enabled, it should be taken off from the *callback-stub.c* and reside into the *<ProjectName>\_callbacks.c*. It means for use that the enabled callbacks should be added manually to the *Zigbee\_Light\_ZC\_callbacks.c* file and implement the desired functionality.

The affected part of the *Zigbee\_Light\_ZC\_callbacks.c* looks like the below:

|  |
| --- |
| boolean **emberAfOnOffClusterOnCallback**(**void**){  emberAfCorePrintln("On command is received");  **halSetLed**(1);  }  boolean **emberAfOnOffClusterOffCallback**(**void**){  emberAfCorePrintln("Off command is received");  **halClearLed**(1);  }  boolean **emberAfOnOffClusterToggleCallback**(**void**){  emberAfCorePrintln("Toggle command is received");  **halToggleLed**(1);  } |

## Command sending from Switch device

First, a place should be found to reside our code for sending the command. For this purpose, a callback is triggered by button press is used.

The button operations are handled by the *Button Interface* plugin, so it should be enabled. The plugin defines some callbacks, so these can be found in the *Callbacks* tab. Move there and enable the *Button1 Pressed Short* callback function.

Save and generate.

Similarly to chapter 8.1 Command handling on Light device, add the function manually to the *Zigbee\_Switch\_ZR\_callbacks.c* file.

Save the modified .isc file and press *Generate*.

Every command is stored in a buffer before it had been sent. The transmitted data buffer should be built up as below:

The actual ZCL command is made by the macro. Replace <> to “On” or “Off”.

|  |
| --- |
| emberAfFillCommandOnOffCluster<>() |

It has to be set which endpoint send to which endpoint.

|  |
| --- |
| emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(),1); |

Send the message as unicast to the device 0x0000, so to the Coordinator.

|  |
| --- |
| emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000); |

The complete function should be something like the below:

|  |
| --- |
| **void** **emberAfPluginButtonInterfaceButton1PressedShortCallback**(uint16\_t timePressedMs){  emberAfCorePrintln("Button1 is pressed for %d milliseconds",timePressedMs);  EmberStatus status;  **if**(timePressedMs < 200){  emberAfFillCommandOnOffClusterOn()  emberAfCorePrintln("Command is zcl on-off ON");  }**else**{  emberAfFillCommandOnOffClusterOff()  emberAfCorePrintln("Command is zcl on-off OFF");  }  **emberAfSetCommandEndpoints**(emberAfPrimaryEndpoint(),1);  status=**emberAfSendCommandUnicast**(*EMBER\_OUTGOING\_DIRECT*, 0x0000);  **if**(status == *EMBER\_SUCCESS*){  emberAfCorePrintln("Command is successfully sent");  }**else**{  emberAfCorePrintln("Failed to send");  emberAfCorePrintln("Status code: 0x%x",status);  }  } |

## Testing

The previous 2 chapters presented how to make the devices to be able to send and receive commands through some APIs.

Build the applications and download the output files to the target devices but exit from the capturing on the Light side. It’s necessary because the debugger has no access to the chip while the Network Analyzer (or Energy Profiler) is connected. If the programming doesn’t erase the full flash memory, the device can rejoin to the network right after the programming, because the “znet” tokes are not lost.

Press Button1 for less than 200ms to send the ON command or between 200ms to 1000ms to send the OFF command. In the meantime, have a look at the CLI of the devices and the Network Analyzer (if the capturing is re-started).

The Switch should print something like the followings:

|  |
| --- |
| Command is zcl on-off ON  Command is successfully sent  Button1 is pressed for 107 milliseconds  Command is zcl on-off ON  Command is successfully sent  Button1 is pressed for 519 milliseconds  Command is zcl on-off OFF |

The CLI of the Light is:

|  |
| --- |
| T00000000:RX len 3, ep 01, clus 0x0006 (On/off) FC 01 seq 00 cmd 01 payload[]  On command is received  Processing message: len=3 profile=0104 cluster=0006  T00000000:RX len 3, ep 01, clus 0x0006 (On/off) FC 01 seq 01 cmd 01 payload[]  On command is received  Processing message: len=3 profile=0104 cluster=0006  T00000000:RX len 3, ep 01, clus 0x0006 (On/off) FC 01 seq 02 cmd 00 payload[]  Off command is received  Processing message: len=3 profile=0104 cluster=0006 |

The above transactions can be observed in the Network Analyzer as well. See Figure 8‑2.

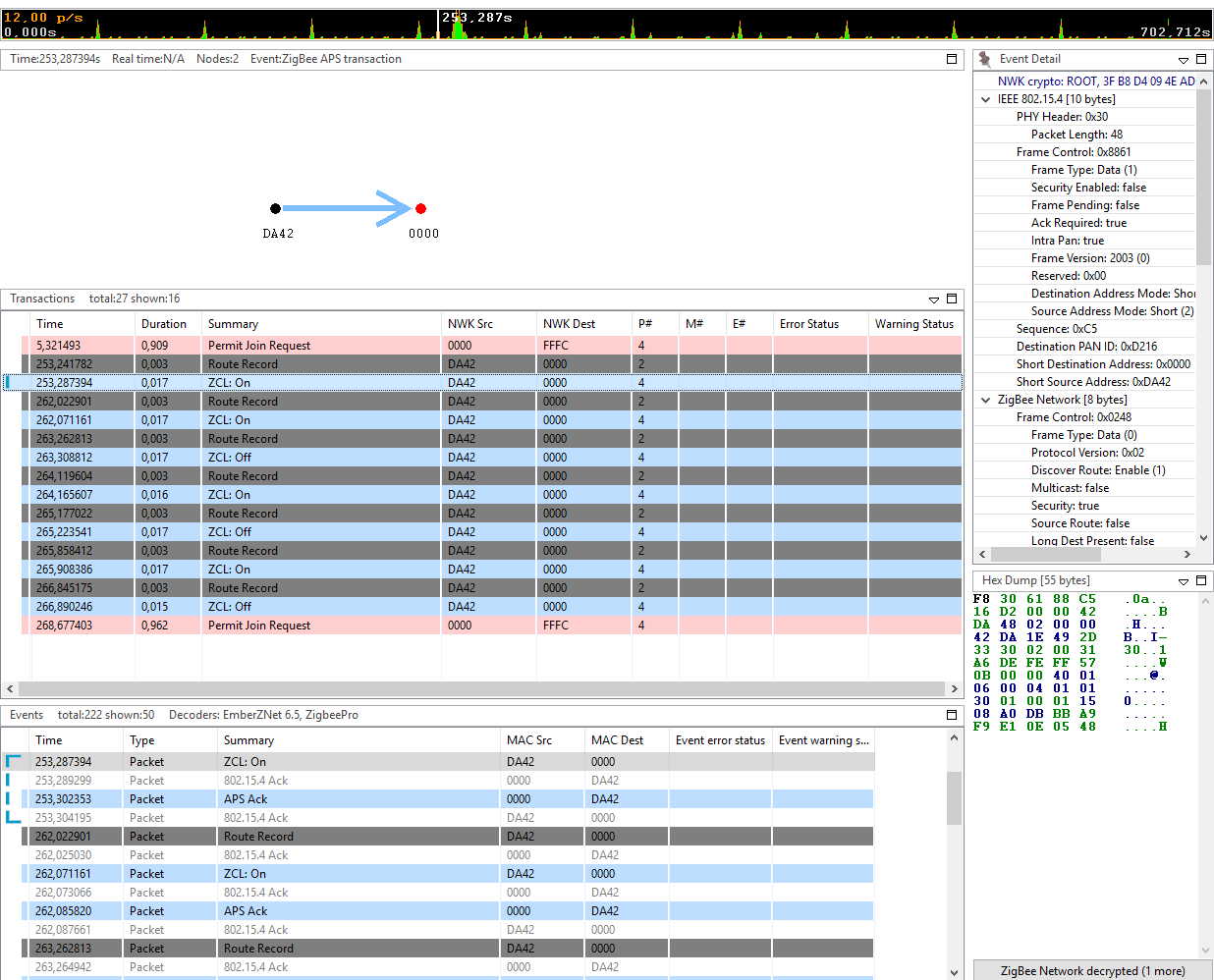


Figure 8‑2  
ZCL On/Off commands in Network Analyzer.

# Custom event

The 3rd phase.

EmberZNet Stack has Event control mechanism that basically allows application to run a piece of code at desired time interval.

An event should be initialized somewhere in the code, hence a function should be used which is called at the begging of the application.

The *Main Init* callback is called from the application’s main function. It can be imagined like a function at the top of the *“main()”* before the classical “*while(true)*”.

Enable this callback in the AppBuilder’s Callbacks tab. See Figure 9‑1.

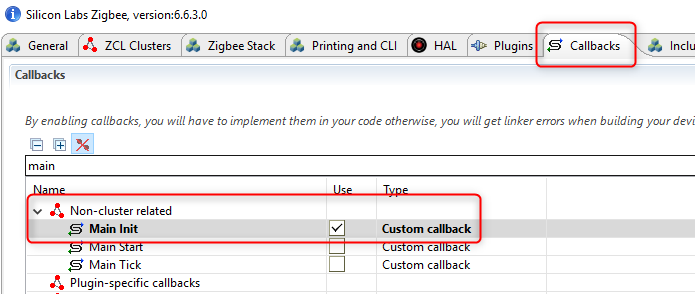


Figure 9‑1  
Main Init callback enabling.

The AppBuilder provides manner to add any custom event to the application. Basically, two things need for this.

* Event Controller – structure of the task
* Event Handler – function on the task

Open the *AppBuilder*->*Includes* tab. Add the custom event to the *Event Configuration* window. See Figure 9‑2.

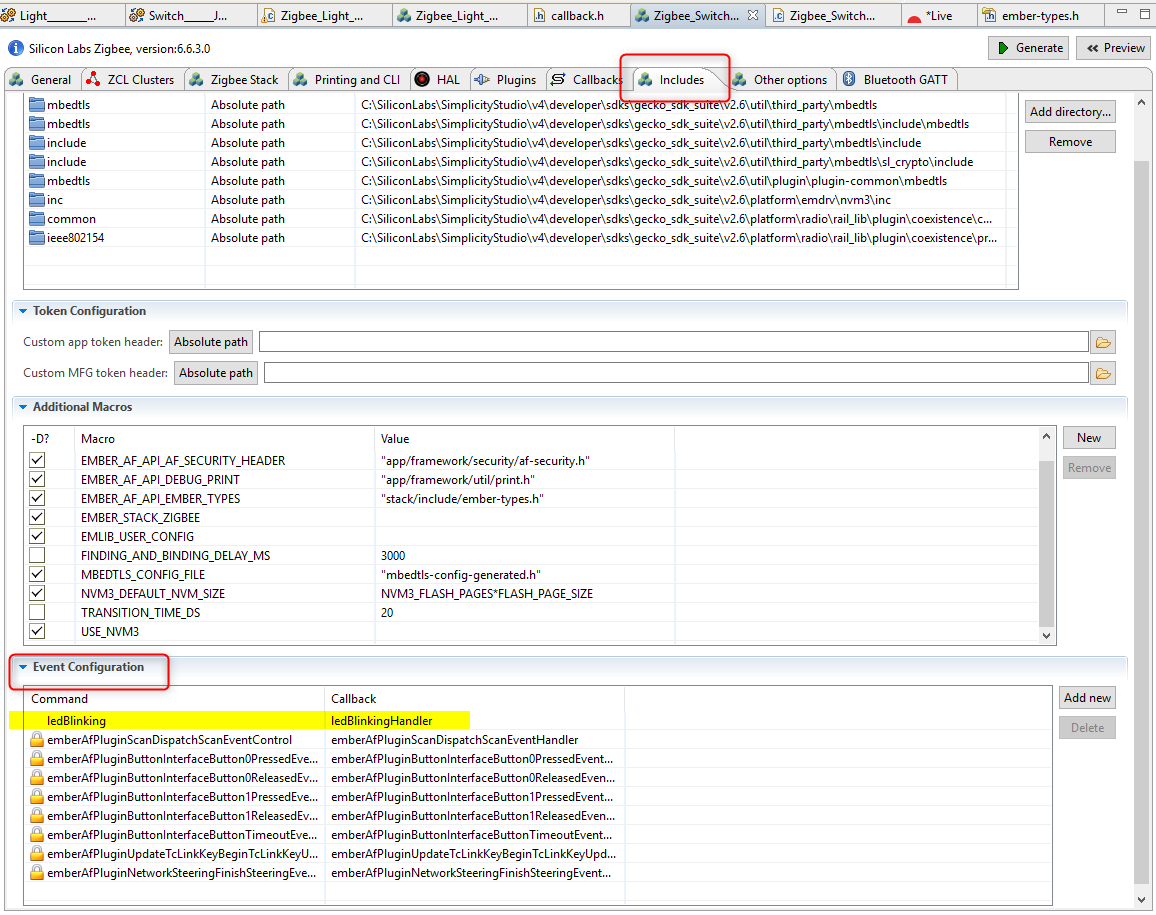


Figure 9‑2  
Custom event adding in AppBuilder.

Save and Generate the project.

As earlier, the callback should be added to the *Zigbee\_Switch\_ZR\_callbacks.c* file and initialize the event.

The related code snippet should be like the followings:

|  |
| --- |
| EmberEventControl ledBlinking;  **void** **emberAfMainInitCallback**(**void**)  {  emberEventControlSetDelayMS(ledBlinking, 1000);  }  **void** **ledBlinkingHandler**(**void**)  {  // First thing to do inside a delay event is to disable the event till next usage  emberEventControlSetInactive(ledBlinking);  **halToggleLed**(1);  //Reschedule the event after a delay of 1 seconds  emberEventControlSetDelayMS(ledBlinking, 1000);  } |

It’s worth to mention that the event should be set to inactive right after its function starts to be executed and re-schedule after it’s done.

# Non-volatile memory