AN1133: Dynamic Multiprotocol Development with *Bluetooth*® and Zigbee EmberZNet SDK 6.x and Lower

This application note provides details on developing Dynamic Multiprotocol applications using Bluetooth and Zigbee. It describes how to configure applications in Simplicity Studio using the Zigbee EmberZNet SDK. It then provides a detailed walkthrough on how the underlying code functions. For details on Dynamic Multiprotocol Application development that apply to all protocol combinations see *UG305: Dynamic Multiprotocol User's Guide*.

This document applies to Zigbee EmberZNet SDK 6.10.x and lower. Zigbee EmberZNet SDK 7 contains significant changes compared to earlier SDKs. Many of these changes are due to an underlying framework redesign that results in an improved developer experience within Simplicity Studio 5. Projects are now built on a component architecture instead of AppBuilder. Simplicity Studio 5 includes project configuration tools that provide an enhanced level of software component discoverability, configurability, and dependency management. See *AN1322: Dynamic Multiprotocol Development with Bluetooth and Zigbee EmberZNet 7.0 and Higher* if you are using or planning to use that version.

KEY POINTS

- Generating and loading dynamic multiprotocol example applications.
- Adding dynamic multiprotocol functionality to an existing project.
- Details on the application User Interface.
- How the Zigbee example applications function.
- How the Bluetooth application functions.

Copyright © 2021 by Silicon Laboratories

1 Introduction

The example applications referenced here can be controlled either from a protocol-specific switch application or from a Bluetooth-enabled smartphone app. This application note provides details on how these examples are designed and implemented. It also describes how to generate, compile, and load example application code, and how to add dynamic multiprotocol functionality to an existing Zigbee project. The application note is intended to be used when developing your own Zigbee/Bluetooth dynamic multiprotocol implementations.

Note: The Zigbee dynamic multiprotocol solution is currently only supported for SoC architectures. Support for NCP architectures is not yet available. Please contact Silicon Labs Sales for more information on our multiprotocol software roadmap.

1.1 Resources

- *UG305: Dynamic Multiprotocol User's Guide* provides details on:
  - Dynamic Multiprotocol Architecture
  - Radio Scheduler operation (with examples)
  - Task Priority management
- *AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage* explains how NVM3 can be used as non-volatile data storage in Dynamic Multiprotocol applications with Zigbee and Bluetooth.

Note: EmberZNet SDK 6.8.0.0 was released as part of Gecko SDK Suite 3.0.0.0 (GSDK v3.x) and is used with Bluetooth SDK v3.x and Simplicity Studio 5. EmberZNet SDK 6.7.x continues to be used with Bluetooth 2.13.x and Simplicity Studio 4. Because of changes to the Bluetooth SDK v3.x, a few instructions and examples in this document vary based on version. Both variants are included and are clearly noted in the text.

1.2 Development Environment Requirements

EmberZNet 6.7.x

- Simplicity Studio 4
- EmberZNet SDK version 6.4.0 or higher
- Bluetooth SDK version 2.10.0 or higher
- Micrium OS-5 kernel version 5.3. or higher (installed automatically with EmberZNet SDK in Simplicity Studio 4)
- An EFR32 chip with at least 512 kB of flash (required to run all the necessary software components)
- IAR Embedded Workbench for ARM (IAR-EWARM) version compatible with your SDK (see the release notes for version details).

EmberZNet 6.8.x and higher

- Simplicity Studio 5
- EmberZNet SDK version 6.8.0 or higher
- Bluetooth SDK version 3.0.0 or higher
- Micrium OS kernel version 6.0.0 or higher (installed automatically with EmberZNet SDK in Simplicity Studio)
- An EFR32 chip with at least 512 kB of flash (required to run all the necessary software components)
- IAR Embedded Workbench for ARM (IAR-EWARM) version compatible with your SDK (see the release notes for version details).

2 Working with the Zigbee/Bluetooth Examples

This section describes

- How to build and flash the dynamic multiprotocol applications supplied with the EmberZNet SDK.
- How to configure a Zigbee project into a dynamic multiprotocol project.

2.1 Application Generation

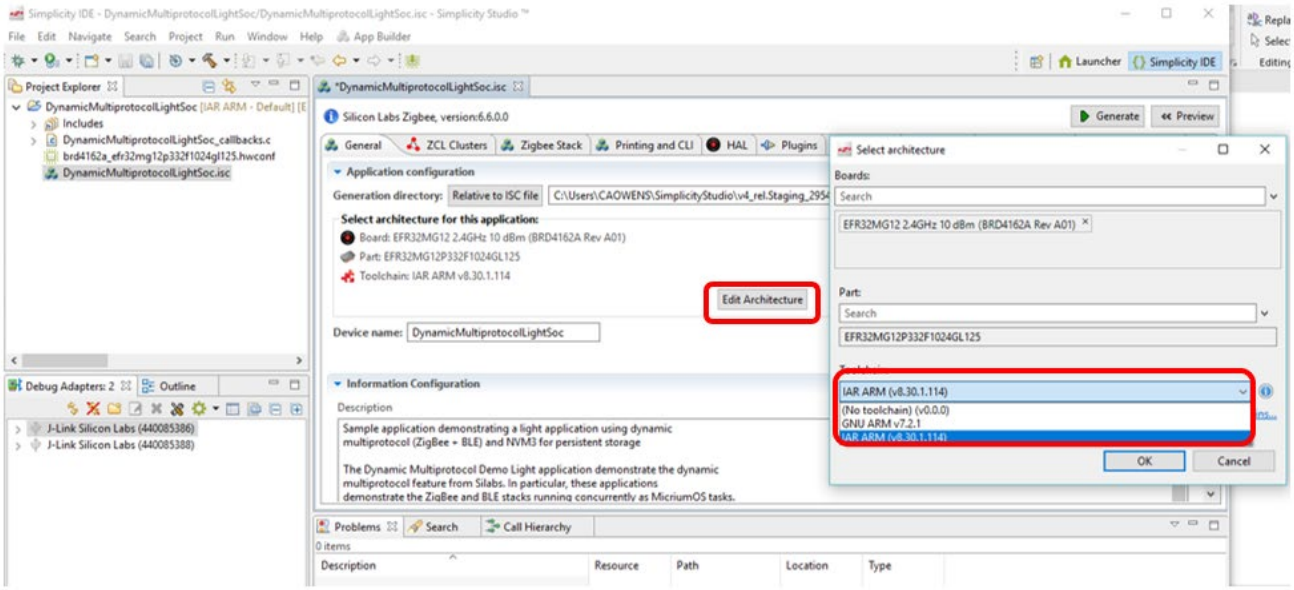
To work with Zigbee/Bluetooth dynamic multiprotocol applications you must install both the EmberZNet SDK and the Bluetooth SDK. The Micrium kernel is installed along with the EmberZNet SDK. IAR Embedded Workbench for ARM (IAR-EWARM) 8.30 must be installed and used as your compiler. See *QSG106: Getting Started with EmberZNet PRO* for information on installing the SDKs and IAR-EWARM.

Dynamic multiprotocol applications are generated, built, and uploaded in the same way as other applications. If you are not familiar with these procedures, see *QSG106: Getting Started with EmberZNet PRO* for details. The dynamic multiprotocol applications included with the EmberZNet SDK are:

- **DynamicMultiprotocolLight** is an application designed to demonstrate a DMP device with Zigbee 3.0 coordinator capabilities.
- **DynamicMultiprotocolLightSed** is an application designed to demonstrate a DMP device with SED capabilities.
- **DynamicMultiprotocolSwitch** is a Zigbee-only application designed to work with the two Zigbee/Bluetooth applications.

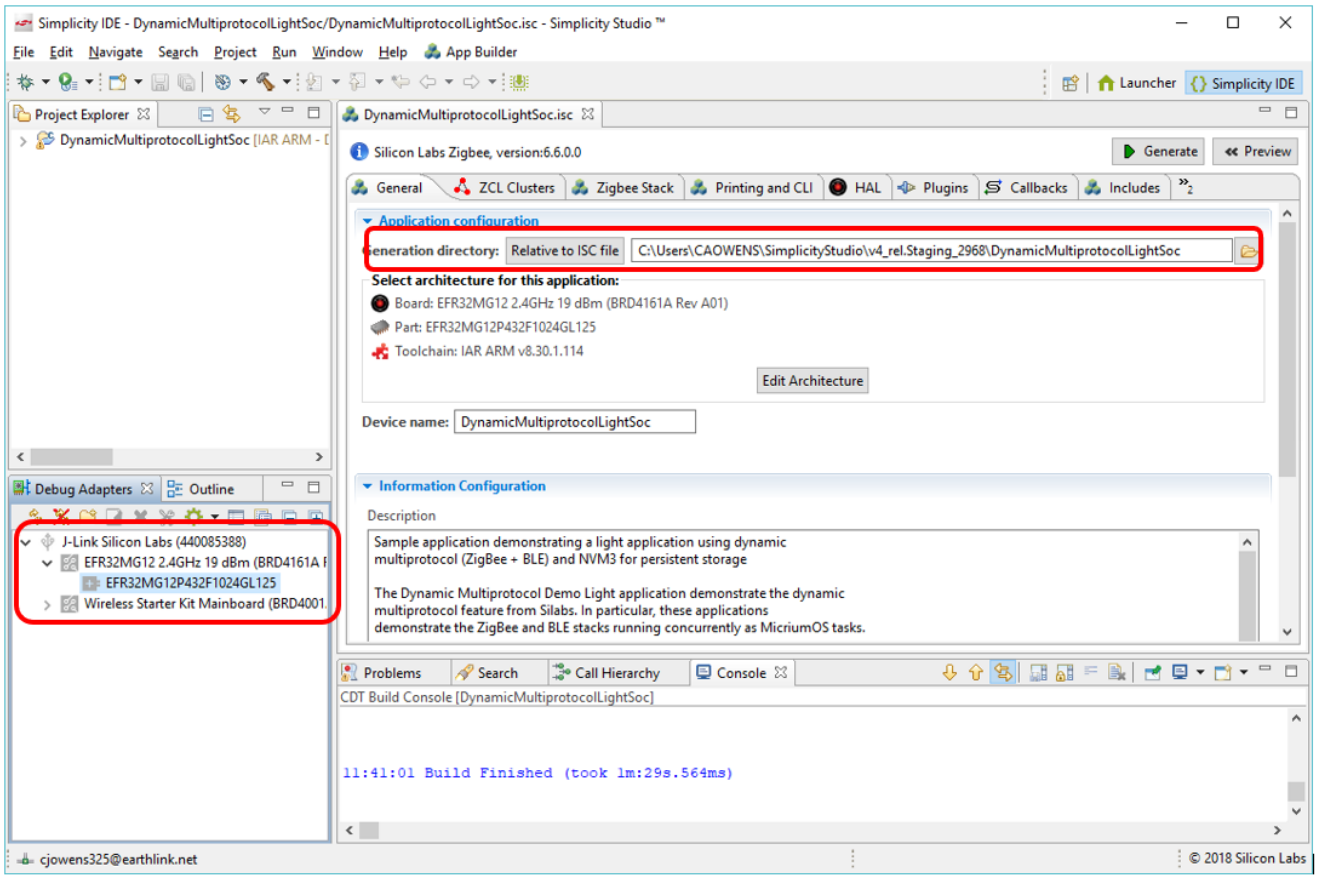
The following summary procedure uses the **DynamicMultiprotocolLight** example application.

1. In Simplicity Studio, start a new project selecting the **DynamicMultiprotocolLight** example.
2. If your project General tab shows GNU-ARM as a compiler, change to IAR EWARM.



3. Click **Generate** to generate project files.
4. Click **Build** (hammer icon) to build the application image.

5. Note the board and part number for your device and the directory for generated files.



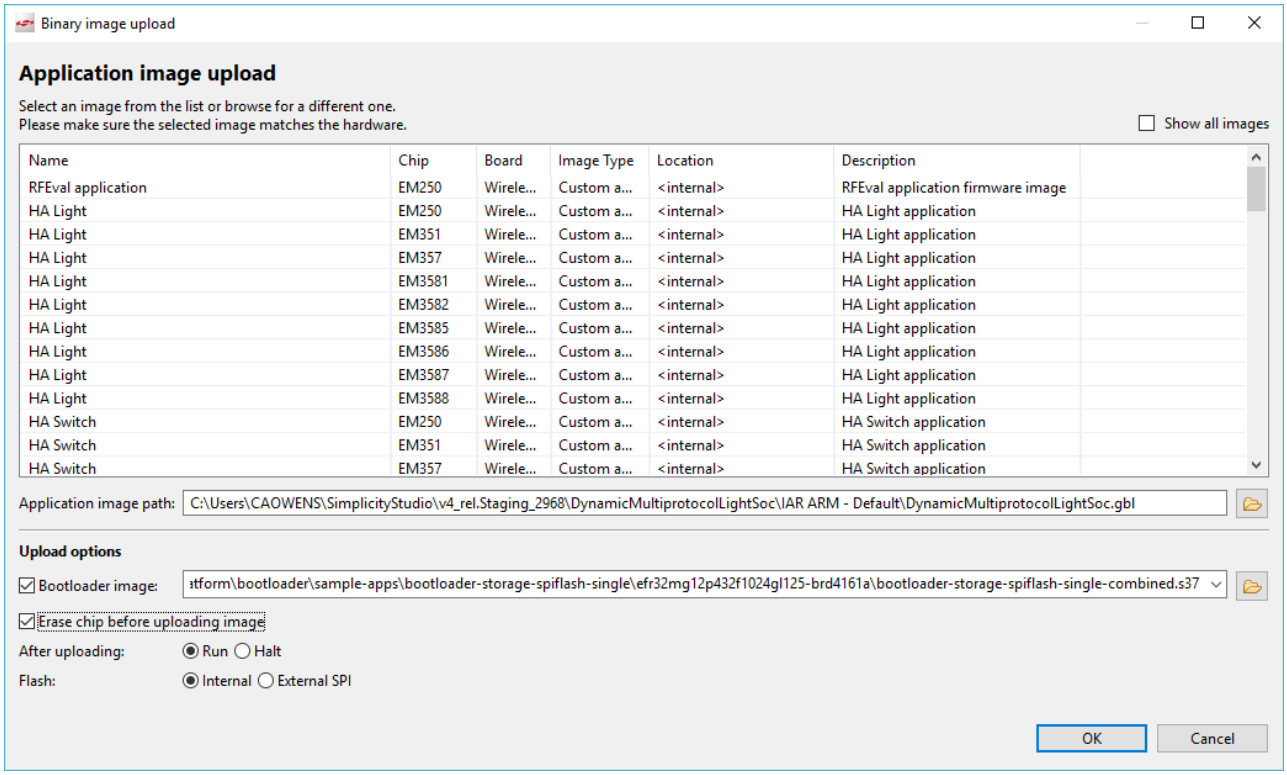
6. Right-click the target J-Link under Devices, and select **Upload Application**.
7. Browse to `<folder on General tab>\IAR ARM - <qualifier>\<project name>` and select the .gbl file.
8. Silicon Labs strongly recommends that, if you have not already loaded a bootloader onto your device, you do so now. Check **Erase chip before uploading image**. Check **Bootloader image**, then browse to the following folder:

   ```
   C:\SiliconLabs\SimplicityStudio_v5\developer\sdks\gecko_sdk_suite\<version>\platform\boot-loader\sample-apps\bootloader-storage-spiflash-single\
   ```

   Open the folder that corresponds to your board and part number and select the .s37 file, for example:

   ```
   \efr32mg12p432f1024gl125-brd4161a\bootloader-storage-spiflash-single-combined.s37
   ```

9. When both images are selected, the dialog should resemble the following figure. Click **OK**.



10. Application load success indicators are code-dependent. With the **DynamicMultiprotocolLight** example, the LCD should display the following before changing over to the light bulb display:

Whether the application is a full function or a sleepy end device is determined by the Device Type on the ZNet tab.

2.2 Converting a Zigbee Application to a Zigbee/Bluetooth LE Dynamic Multiprotocol Application

This section describes the configuration changes required to convert a working Zigbee application into a Zigbee/Bluetooth LE Dynamic Multiprotocol application. The instructions assume you have started with a non-DMP Zigbee sample application or your own Zigbee project, and that the application is working correctly.

Requirements:

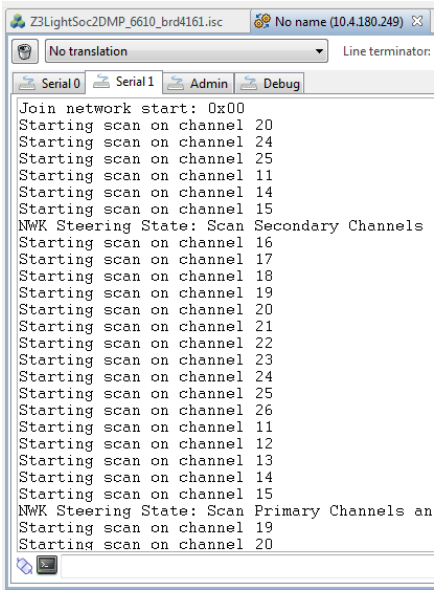
- Zigbee application set up to build with IAR ARM v8.30.1 (for these instructions we use Z3 Light)
- EFR32MG12 or other EFR32 with sufficient memory (for these instructions we assume BRD4161 (EFR32MG12P432F1024GL125))

Note: The Dynamic Multiprotocol sample applications supplied with Simplicity Studio are already correctly defined and do not require modification before project generation unless performing an OTA update. There is a potential conflict with the DMP sample app LCD screen and the external flash. If you need to perform OTA updates, check the **Dynamic Multiprotocol UI Demo Code Stub** plugin as described in the following section.

2.2.1 Generate and Build the Zigbee Application

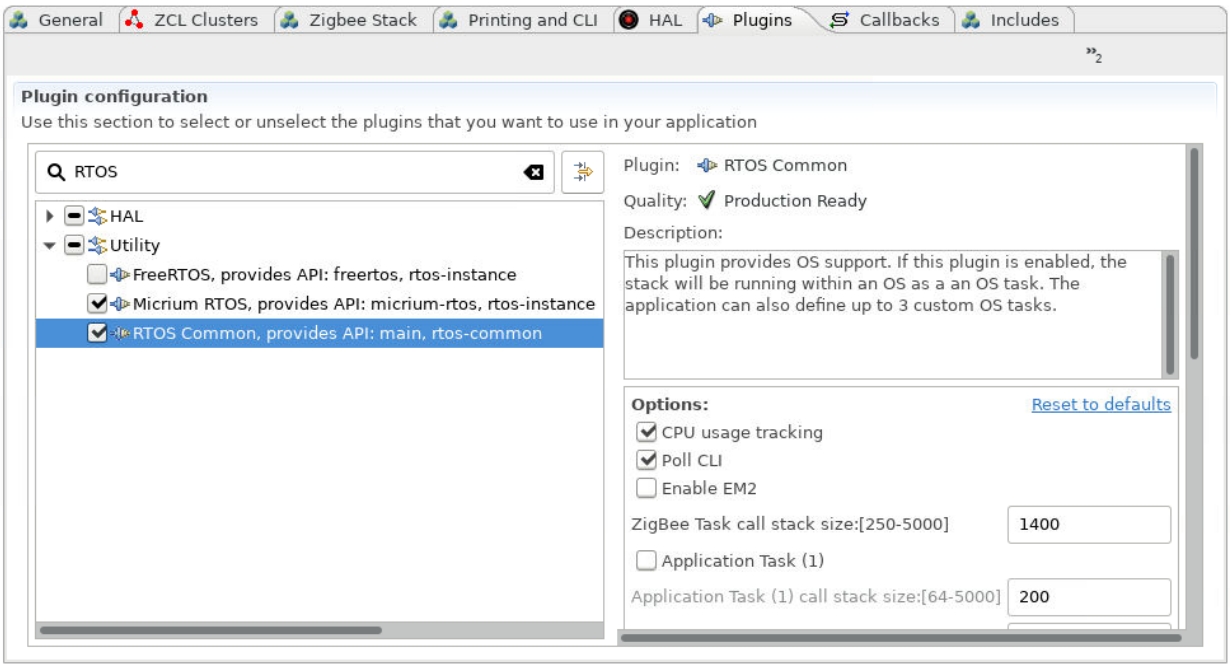
The purpose of this procedure is to verify that the base application had loaded and is working correctly, and that output is printing to the console. This example uses the Z3Light sample application in the EmberZNet SDK It begins with the default settings, so that the configuration changes are clear. Remember to select IAR as the toolchain.

Generate and build the project, load it to the board and check the Serial 1 output to make sure it's up and running.



2.2.2 Reconfigure the Project

The search bar at the top of the Plugins and other tabs is helpful when modifying the configuration. For plugins, the description explains its utility for DMP. The following figure illustrates finding the RTOS plugin and its description.

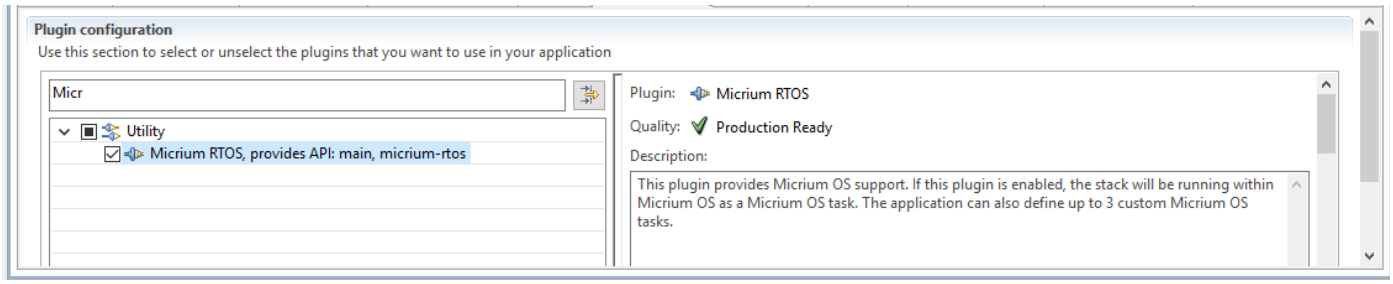


Note: Beginning with SDK 6.9.0, the RTOS configurations options moved from the Micrium plugin to the RTOS common plugin and the Zigbee Task call stack size is treated as words instead of bytes.

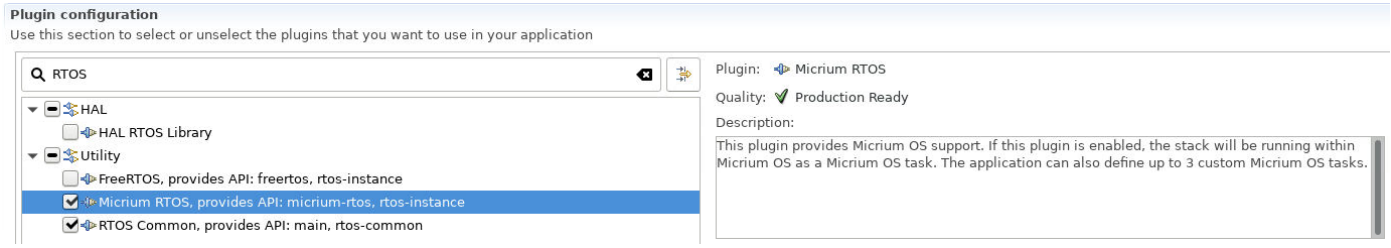
1. On the Plugins tab, check the following:

- **RTOS**

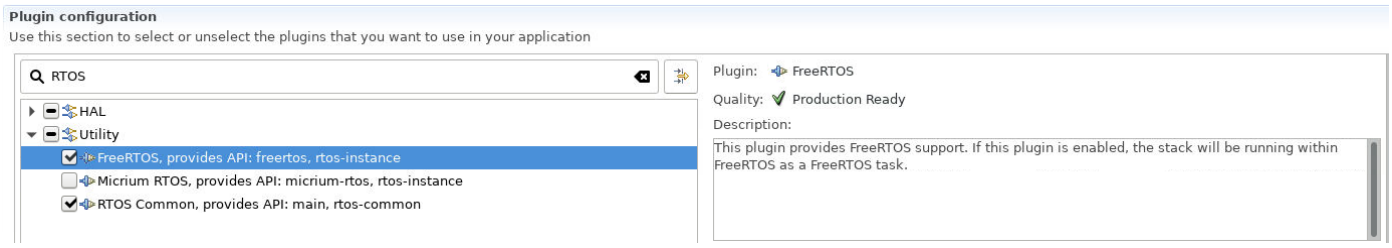
In SDKs 6.8.x or lower, check Micrium RTOS.



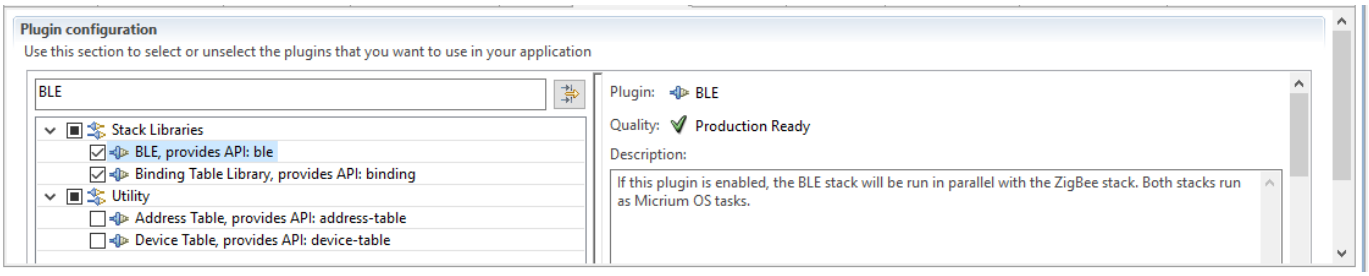
In SDKs 6.9.0 and higher, you have the option of using Micrium RTOS or Free RTOS. For Micrium RTOS, check Micrium RTOS and RTOS Common.



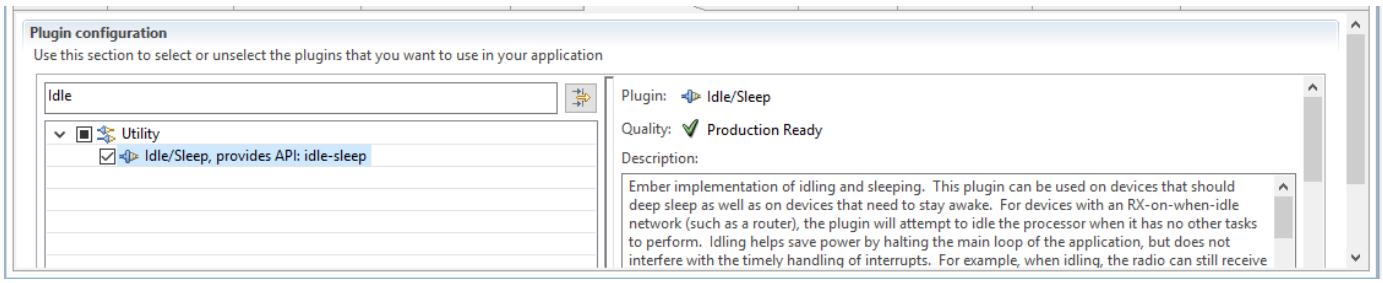
From SDK 6.9.x you can use FreeRTOS by selecting the FreeRTOS plugin and unchecking the Micrium RTOS plugin FreeRTOS is not supported yet. Theoriticially it should work, but we do not plan on testing it before Q4.



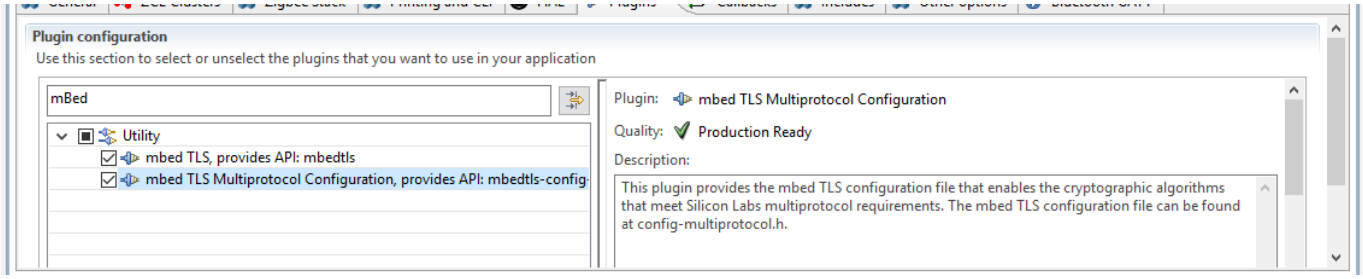
- **BLE**



- **Idle/Sleep**

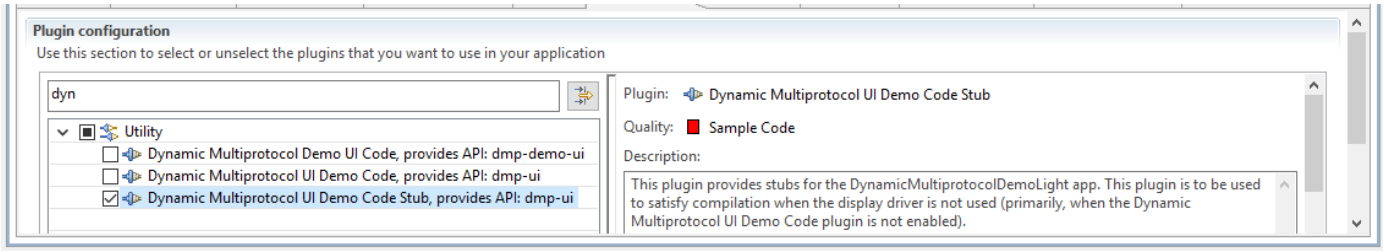


- **mbed TLS Multiprotocol Configuration**



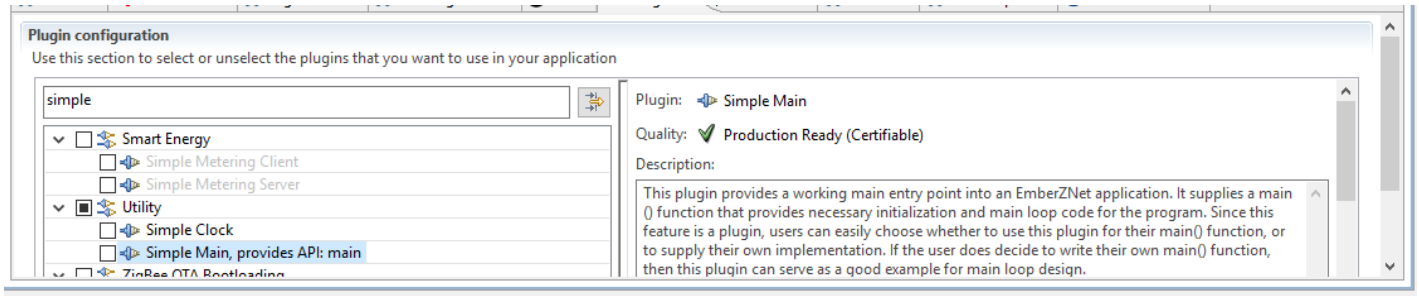
- (optional) **Dynamic Multiprotocol UI Demo Code Stub**

The USART used to communicate with the external flash on Silicon Labs radio boards is the same USART that communicates with the LCD display on the WSTK. In order to perform an OTA update on a sample app that uses the LCD display, check **Dynamic Multiprotocol UI Demo Code Stub**. This disables the LCD display but allows the OTA to take place.



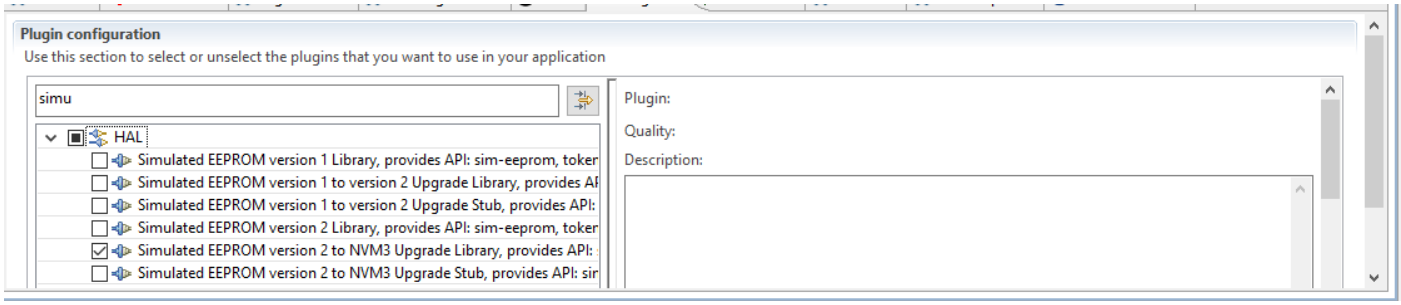
2. On the Plugins tab, uncheck (disable) the following:

- **Simple Main**

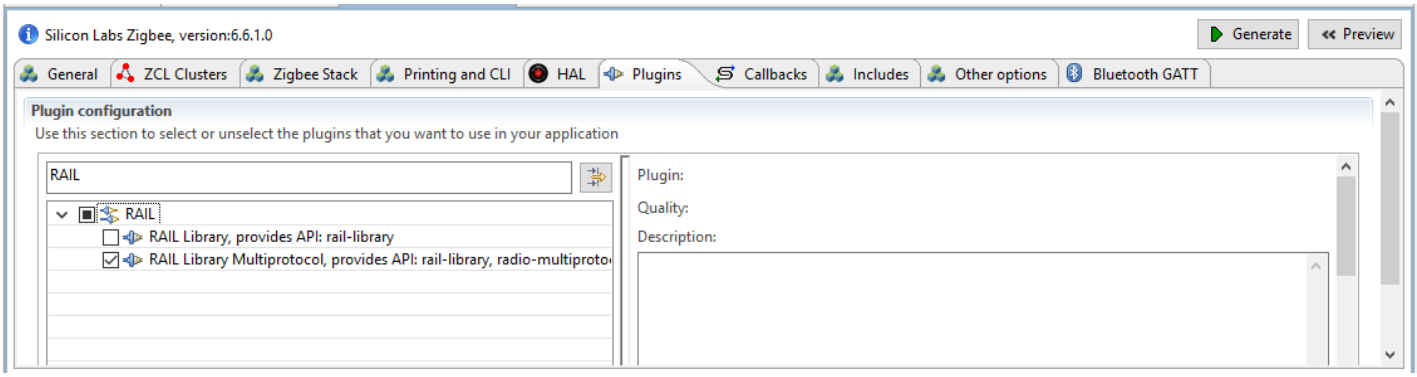


3. On the Plugins tab, change settings for the following:

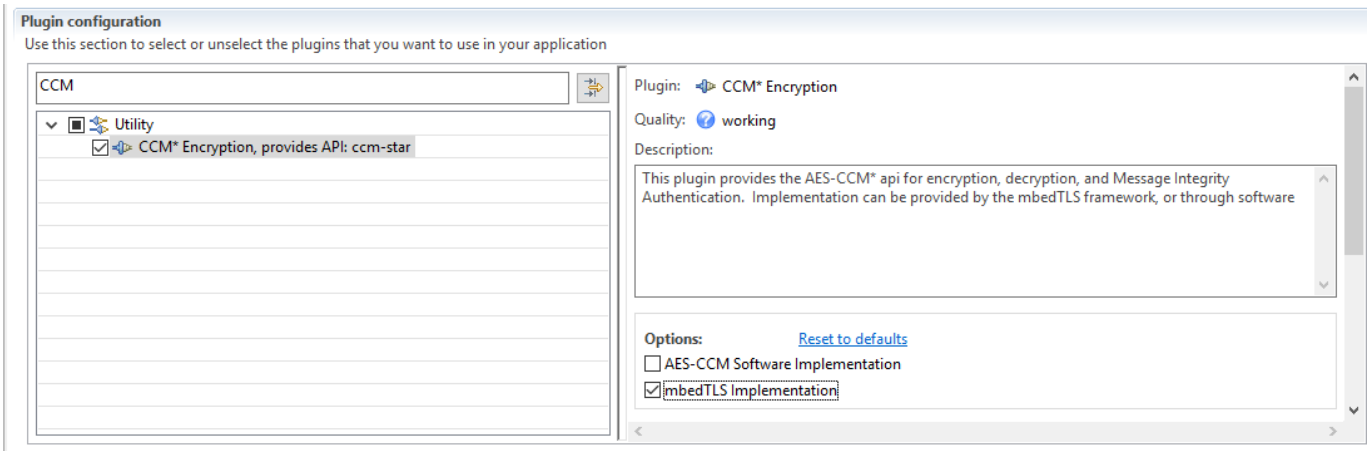
- HAL
  - Uncheck **Simulated EEPROM version 1 Library**.
  - Check **Simulated EEPROM Version 2 to NVM3 Upgrade Library** (this will also enable the NVM plugin).



- RAIL
  - Uncheck **RAIL Library**.
  - Check **RAIL Library Multiprotocol**.



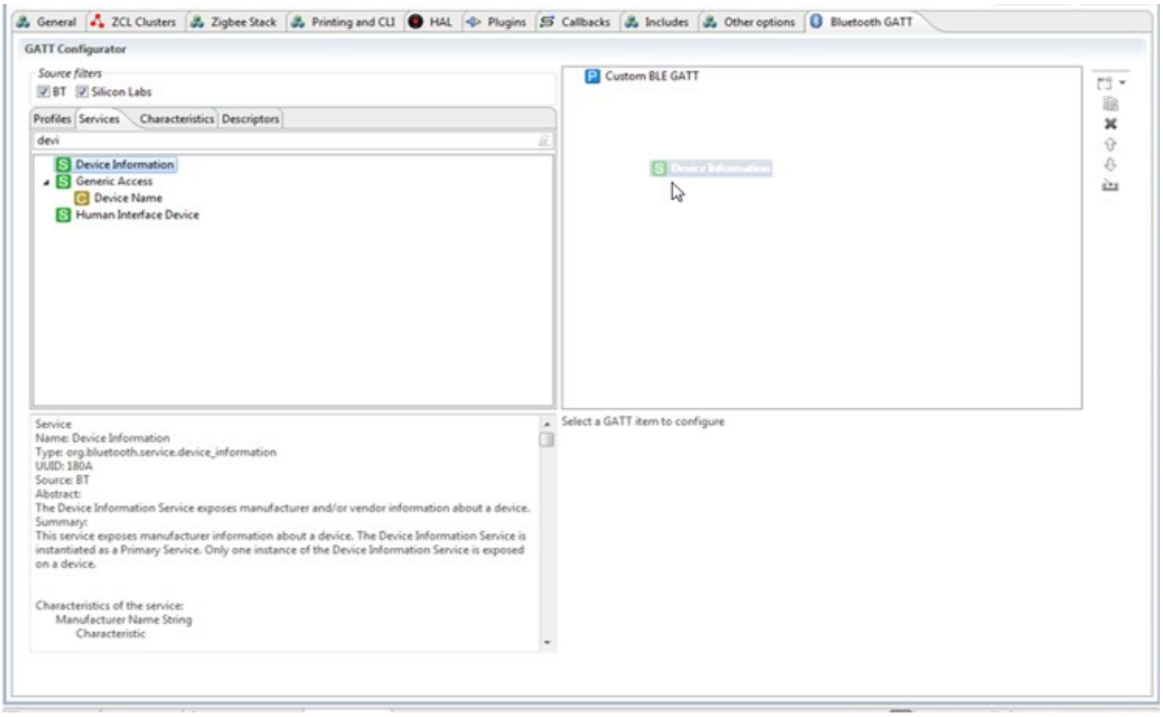
- UTILITY
  - In the CCM* Encryption plugin, uncheck **AES-CCM Software Implementation** and check **mbedTLS Implementation**.



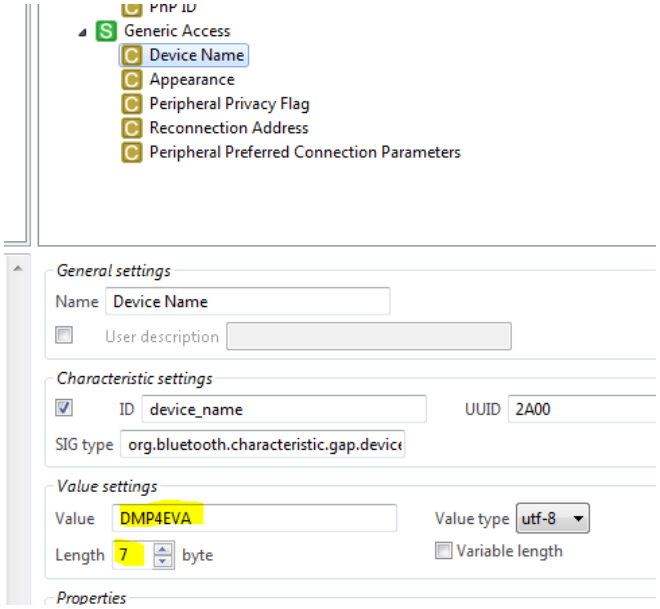
4. Add Bluetooth GATT elements:

   On the **Bluetooth GATT** tab, **Services** tab, drag and drop the following into the Custom BLE GATT window:

   - Device Information
   - Generic Access

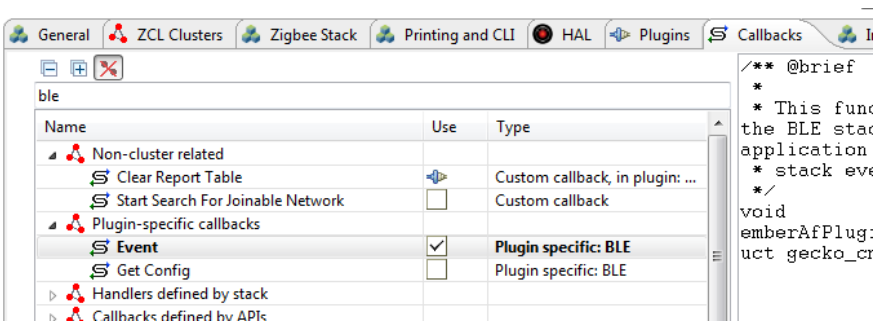


5. Add an identifiable name for your device to advertise by modifying the **Generic Access** > **Device Name Value**. Be sure to increase the **Length** 1 byte per character.



6. Add BLE callback code:

   In the Callbacks tab, enable Plugin-specific callbacks: **Event.**



7. Generate the project.
8. Add `emberAfPluginBleEventCallback` code to the **[project_name]_callbacks.c** file, based on the SDK version you are using..

   For SDK 6.7.x:

```
/** @brief
 *
 * This function is called from the BLE stack to notify the application of a
 * stack event.
 * In this case it will provide simple advertising for the BLE component of the Z3Light DMP app
 */
    void emberAfPluginBleEventCallback(struct gecko_cmd_packet* evt){

        switch (BGLIB_MSG_ID(evt->header)) {

        case gecko_evt_system_boot_id:

                gecko_cmd_le_gap_start_advertising(0,le_gap_general_discoverable,le_gap_connectable_scanna-
ble);
                emberAfCorePrintln("BLE Advertising started");
                break;
        case gecko_evt_le_connection_opened_id:
                emberAfCorePrintln("BLE connection opened"); //Will cause advertising to stop
                break;

        case gecko_evt_le_connection_closed_id:
                emberAfCorePrintln("BLE connection closed");
                gecko_cmd_le_gap_start_advertising(0,le_gap_general_discoverable,le_gap_connectable_scanna-
ble); // restarting advertising
                break;
        default :
                emberAfCorePrintln("unhandled BLE event\r\n");
                break; }

}
```

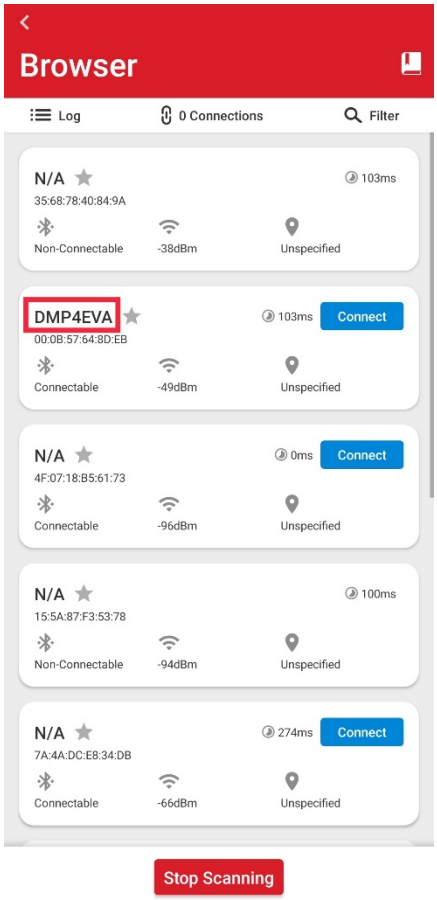
For SDK 6.8.x:

```
#include "sl_bt_rtos_adaptation.h"
static uint8_t advertising_set_handle = 0xff;

/** @brief
 *
 * This function is called from the BLE stack to notify the application of a
 * stack event.
 * In this case it will provide simple advertising for the BLE component of the Z3Light DMP app
 */
    void emberAfPluginBleEventCallback(sl_bt_msg_t* evt){

        switch (SL_BT_MSG_ID(evt->header)) {

        case sl_bt_evt_system_boot_id:

               sl_bt_advertiser_create_set(&advertising_set_handle);
               sl_bt_advertiser_start(advertising_set_handle,            // advertising set handle
                                      advertiser_general_discoverable,   // discoverable mode
                                      advertiser_connectable_scannable); // connectable mode

               emberAfCorePrintln("BLE Advertising started");
               break;
        case sl_bt_evt_connection_opened_id:
               emberAfCorePrintln("BLE connection opened"); //Will cause advertising to stop
               break;

        case sl_bt_evt_connection_closed_id:
               emberAfCorePrintln("BLE connection closed");
               sl_bt_advertiser_start(advertising_set_handle,            // advertising set handle
                                      advertiser_general_discoverable,   // discoverable mode
                                      advertiser_connectable_scannable); // connectable mode
               break;
        default:
               emberAfCorePrintln("unhandled BLE event\r\n");
               break;
       }

}
```

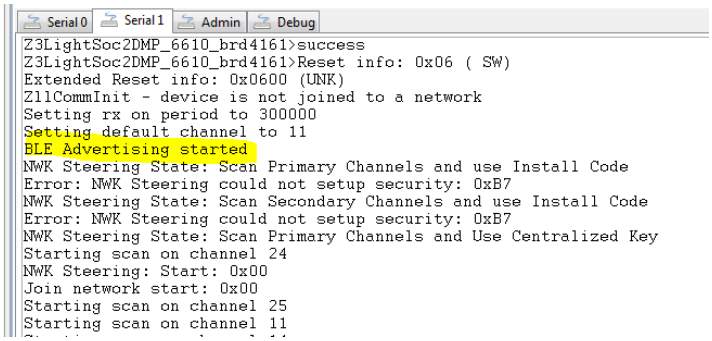
9. For **SDK 6.8.x and up**, the following defines need to be added to the file **mbedtls-config-generated.h**:

```
#define MBEDTLS_CTR_DRBG_C
#define MBEDTLS_SHA256_C
#define MBEDTLS_ENTROPY_C
```

10. Build and flash your project and look for your device in the Bluetooth Browser screen of the EFR Connect cell phone app.



You can also see comments for the BLE activity in the Serial 1 window among the Zigbee prints



This is very basic Bluetooth functionality. To learn more about programming BLE functionality, see *QSG139: Bluetooth® SDK v2.x Quick Start Guide/QSG169: Bluetooth® SDK v3.x Quick Start Guide*, included with the corresponding Bluetooth SDK.

3 About the Zigbee/Bluetooth LE Examples

The Zigbee/Bluetooth LE Dynamic Multiprotocol examples demonstrate a light that can be controlled from both Bluetooth and a Zigbee network. Software is included both as compiled demonstrations and as example code in the EmberZNet SDK. The purpose of the examples is to show the way of implementing a dynamic multiprotocol application using the Silicon Labs EmberZNet stack.

The Dynamic Multiprotocol Demo application has three main components.

1. User Interface (LCD and Buttons)
2. Zigbee application (FFD and/ or SED)
3. Bluetooth application

3.1 User Interface

The user interface is developed specifically for the dynamic multiprotocol demonstration, and APIs to update the text and graphic on the LCD are called directly from Zigbee and Bluetooth event handlers. The implementation to manipulate the LCD is contained in the following files,

```
bitmaps.h //Contains the arrays containing the bitmap of the graphics drawn on the LCD

dmp_ui.c //Contains the functions to change the state of the display based on the state of the application

dmp_ui.h //Header file exporting functions implemented in the dmp_ui.c
```

The above uses the display driver library supplied by Silicon Labs to update the content on the LCD display mounted on the WSTK.

3.2 Zigbee Application

The example **DynamicMultiprotocolLight** is set up to be a light and a coordinator on the Zigbee network.

The following cluster set is supported by both the **DynamicMultiprotocolLight** and **DynamicMultiprotocolLightSed** applications.

Supported Clusters
Basic
Identify
Scenes
Groups
On/Off
ZLL Commissioning

The **DynamicMultiprotocolLight** example also supports Green Power Proxy Basic behavior. Please note that the examples were developed with a focus on demonstrating dynamic multiprotocol features and may not be Zigbee-certifiable.

The On/Off cluster controls the LEDs and the bulb icon on the WSTK board to represent the state of the light.

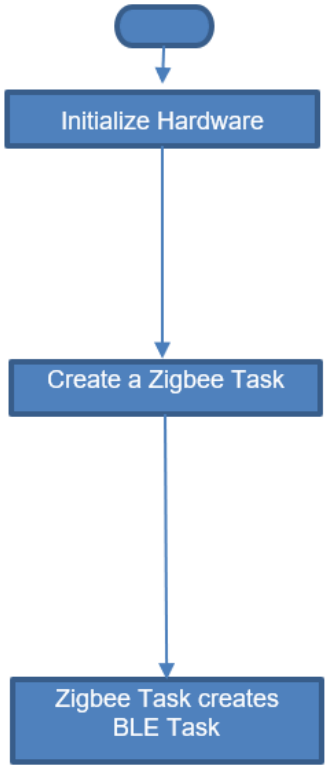
The dynamic multiprotocol applications make use of Micrium OS and the Zigbee applications are run as a task of Micrium OS.

The hardware and peripherals of the chip are initialized before any tasks are created. A Zigbee task is created after initialization, which then creates the application tasks and Bluetooth task.

The Micrium plugin also includes the source file micrium-rtos-sleep.c, which enables the sleepy DMP application to manage the sleep functionality.

Note that in Bluetooth SDK v3.x commands were renamed and restructured. The code example in the last box illustrates both.

From: micrium-rtos-main.c



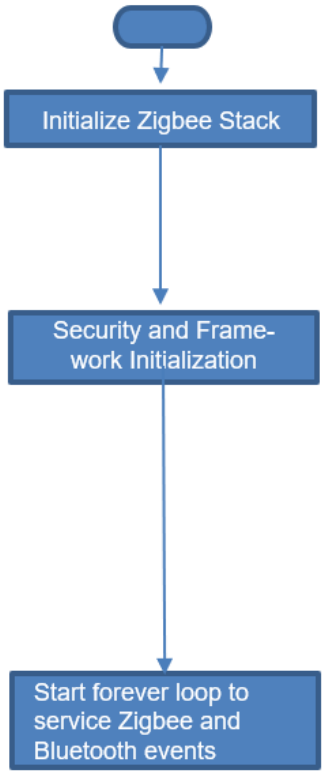
Micrium-rtos.main.c

```
halInit();
initMicriumCpu();
emberAfMainInit();
```

```
OSTaskCreate(&zigbeeTaskControlBlock,
             "Zigbee Stack",
             zigbeeTask,
             NULL,
             ZIGBEE_STACK_TASK_PRIORITY,
             &zigbeeTaskStack[0],
             EMBER_AF_PLUGIN_MICRIUM_RTOS_ZIGBEE_STACK_SIZE / 10,
             EMBER_AF_PLUGIN_MICRIUM_RTOS_ZIGBEE_STACK_SIZE,
             0, // Not receiving messages
             0, // Default time quanta
             NULL, // No TCB extensions
             OS_OPT_TASK_STK_CLR | OS_OPT_TASK_STK_CHK,
             &err);
```

```
bluetooth_start_task(BLE_LINK_LAYER_TASK_PRIORITY,
                     BLE_STACK_TASK_PRIORITY);       /* v2.x */
sl_bt_rtos_init();                                    /* v3.x */
```

From: af-main-soc.c



Af-main-soc.c

```
status=emberInit();
```

```
emAfInitializeNetworkIndexStack();
// Initialize messageSentCallbacks table
emAfInitializeMessageSentCallbackArray();
emberAfEndpointConfigure();
emAfInit();

// The address cache needs to be initialized and used with the
source routing
// code for the trust center to operate properly.
securityAddressCacheInit(EMBER_AF_PLUGIN_ADDRESS_TABLE_SIZE,
// offset
      EMBER_AF_PLUGIN_ADDRESS_TABLE_TRUST_CENTER_CACHE_SIZE);
// size

EM_AF_NETWORK_INIT();
```

```
while (true) {
    halResetWatchdog();   // Periodically reset the watchdog.
    emberTick();          // Allow the stack to run.
    // Allow the ZCL clusters and plugin ticks to run. This
should go
    // immediately after emberTick
    // Skip these ticks if a crypto operation is ongoing
    if (0 == emAfIsCryptoOperationInProgress()) {
      emAfTick();
    }

    emberSerialBufferTick();
    emberAfRunEvents();
}
```

On either DMP light application, once the Zigbee stack is set up to run, subsequent interactions with the stack occurs via event handlers, as shown in the following figures. The following figure shows the event handlers in the full function light application.

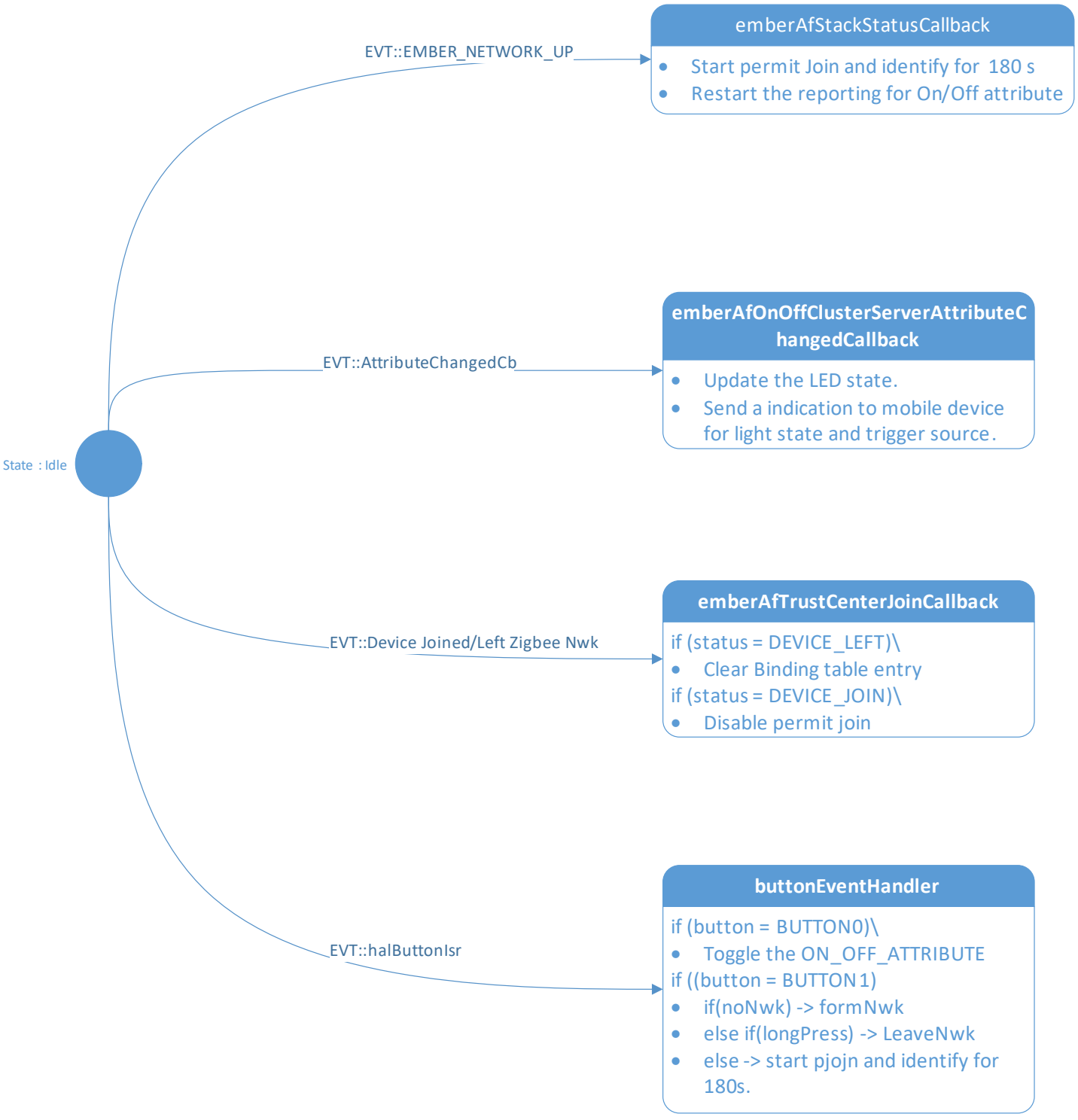


Figure 3-1. DMP Full Function Light Event Handler Definition

Note: Whenever the light starts `pjoin`, it starts identifying **and** also puts all the connected lights in identify mode. This helps the joining switch to identify all the lights present in the network.

The following figure shows the application interaction with the stack with the event handlers used for the sleepy light application.

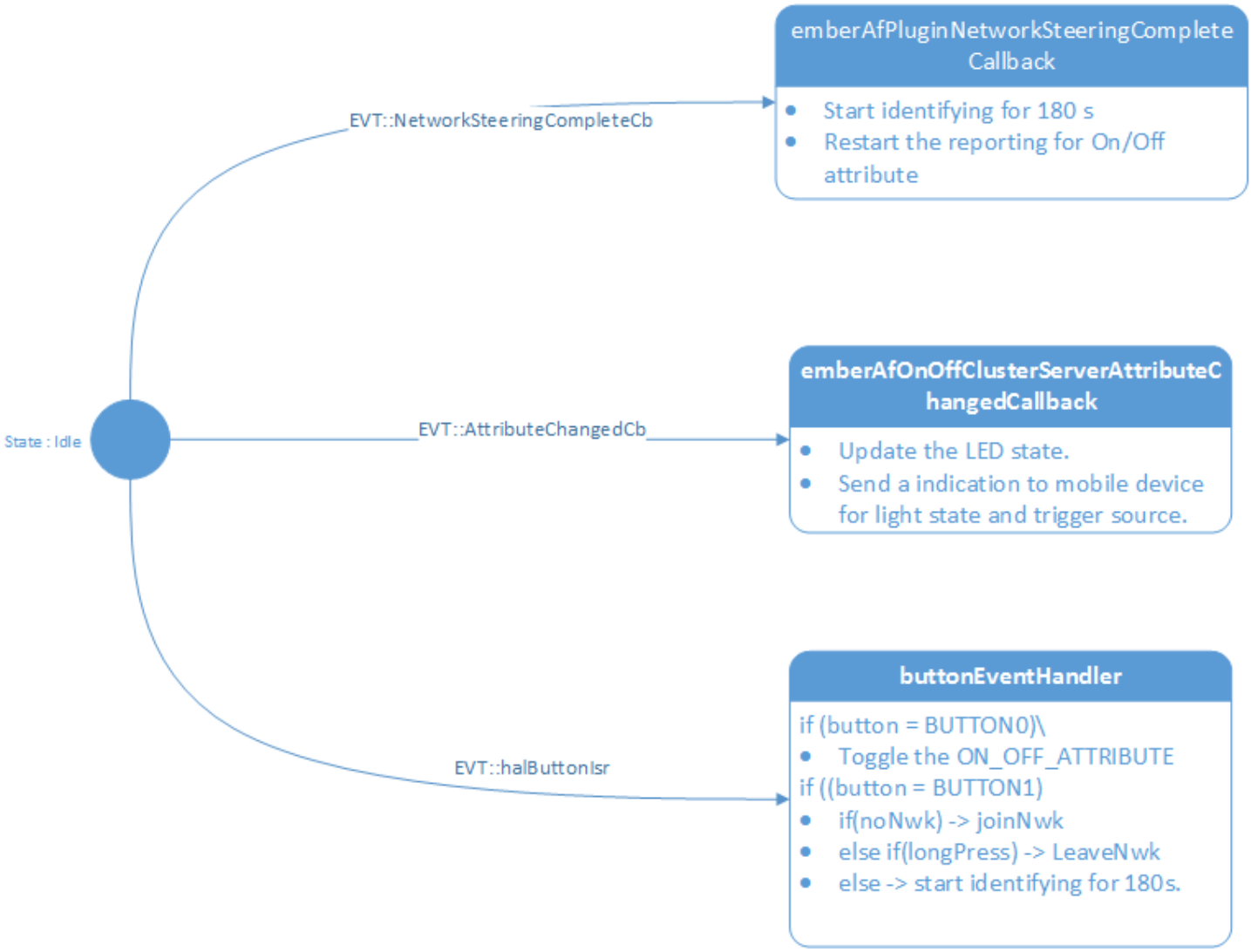


Figure 3-2 DMP Sleepy Light Event Handler Definition

Note: To avoid the risk of shared resources, if you want to send Zigbee messages from a task other than the Zigbee Stack Task, we advise you to schedule a custom event from within the non-Zigbee Stack task. In the corresponding event handler function for the custom event the Zigbee stack APIs can be used, as the event handler will be called from the Zigbee Stack Task context.

3.3 Bluetooth Application

The Bluetooth application supports following services and characteristics. These are pre-selected in the GATT editor during project generation.

Service	Characteristic
Device Information	Manufacturer Name String Model Number String Serial Number String Firmware Revision String
Generic Access	Device Name Appearance
Silabs DMP Light	Light Trigger Source

3.3.1 Silabs DMP Light Service

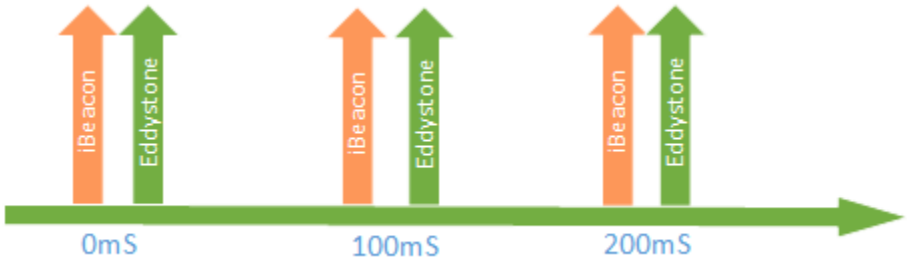
In the above table the Silabs DMP Light is a custom service with a UUID of `bae55b96-7d19-458d-970c-50613d801bc9`. This custom UUID is used to uniquely identify the Light by the Wireless Gecko application.

The Service has two characteristics,

Characteristic	Data Type	Description
Light	8bit Boolean	Used to get and set the light state 1 = Light On 0 = Light Off
Trigger Source	8bit enum	Indicates the source of the Light state change command. 0 = Bluetooth 1 = Zigbee 2 = Button Press

3.3.2 Beacons

The application implements both an iBeacon as well as an Eddystone beacon. The default behavior is to transmit each beacon at 100 mS intervals.



3.3.3 Bluetooth Event Handling

The Bluetooth stack is initialized as part of the Zigbee Task, as shown in the Zigbee implementation section. The Bluetooth task handles the Bluetooth LE link layer messaging and management. The Bluetooth stack's interaction with the user application is through a framework plugin. A number of events that are called in the context of the Zigbee task allow the user application to interact with the Bluetooth stack. The following diagram describes the Bluetooth-related events. In Bluetooth v3.x commands and events were renamed, substituting **sl_bt_** for **gecko_cmd_** and **gecko_**, respectively. Both variants are shown.

Note: Bluetooth event handling is same for both DMP demos.

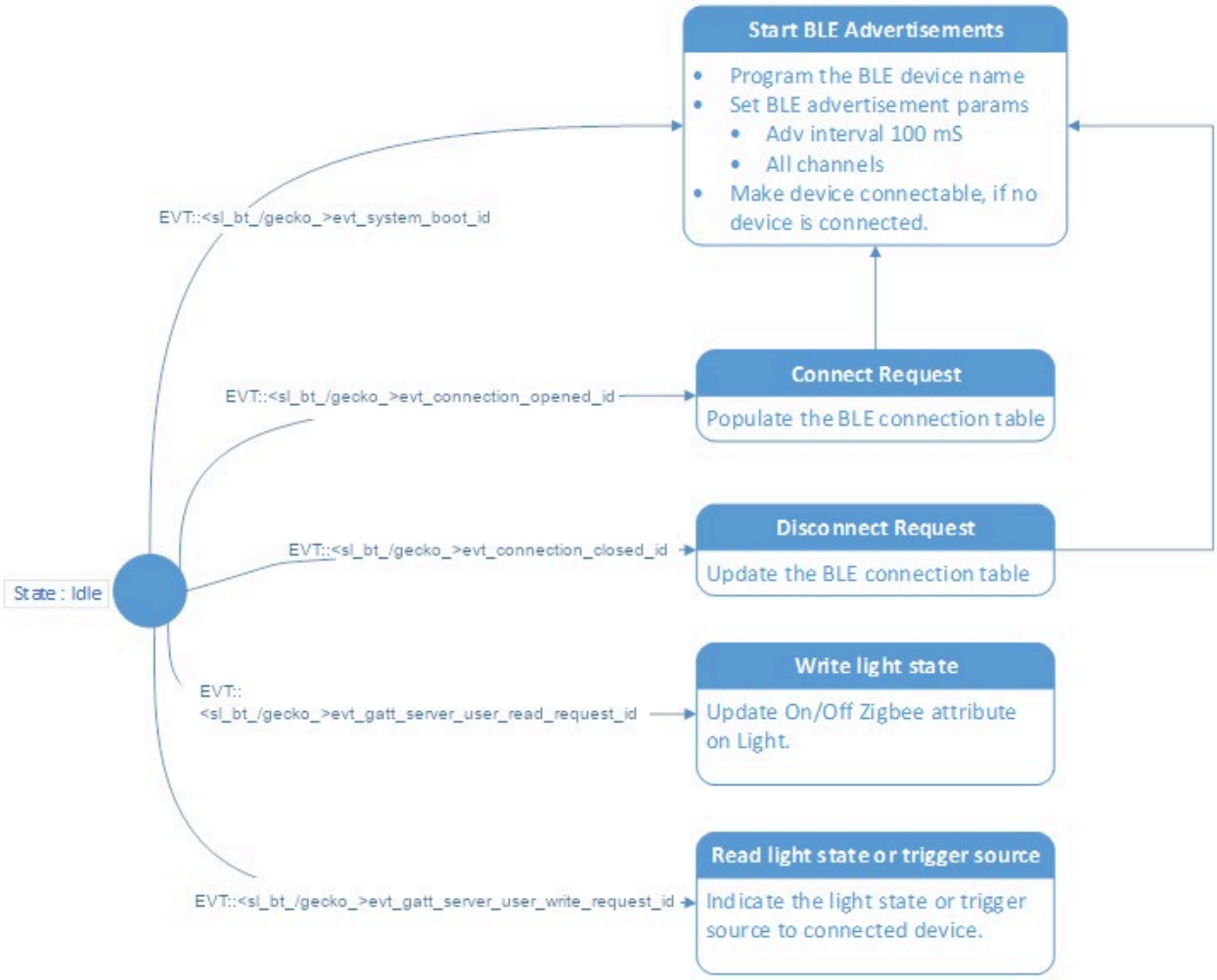


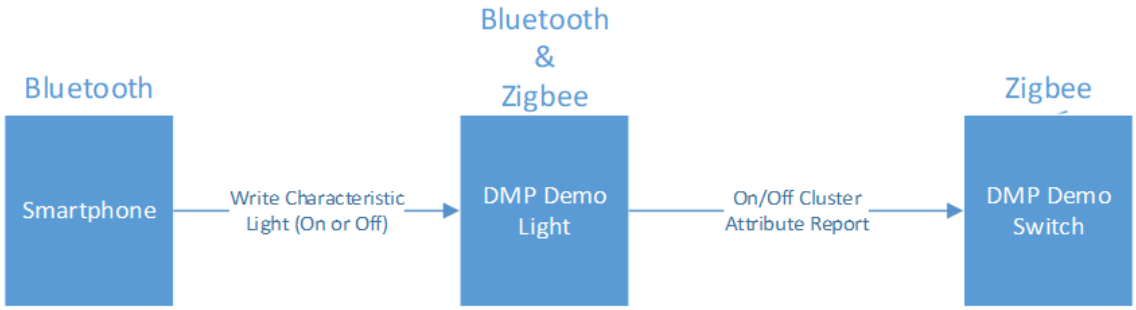
Figure 3-3 DMP Bluetooth Event Handler Definition

3.3.4 Bluetooth and Zigbee Interaction

The primary purpose of the example applications is to show Zigbee and Bluetooth working together on a device. For this purpose, when the Light receives a command to change its state through one protocol, it executes the command and sends out a notification to the other devices using the other protocol to keep everything in sync. Their interaction is the same in both examples.

Two basic operations are described below, first a write to Light characteristics from a Bluetooth connected device (shown in the following figure) and then a change in the Light state from a Zigbee device.

Write from the Bluetooth Connected Device



The application's services and characteristics are pre-selected in the GATT editor in Simplicity Studio. Upon generation the characteristics are #define in the gatt_db.h. Using the #define reference, the characteristics can then be coupled to read and write Bluetooth requests. For example the Light characteristic is reference from GATT as `gatt_light_state` which is then tied to an application specific write API of `writeLightState` in the `AppCfgGattServerUserWriteRequest` as shown below.

```
static const AppCfgGattServerUserWriteRequest_t appCfgGattServerUserWriteRequest[] =
{
  { gattdb_light_state, writeLightState },
  { 0, NULL }
};
```

The application implements the Zigbee attribute write and a Bluetooth write response in the `writeLightState` function as follows:

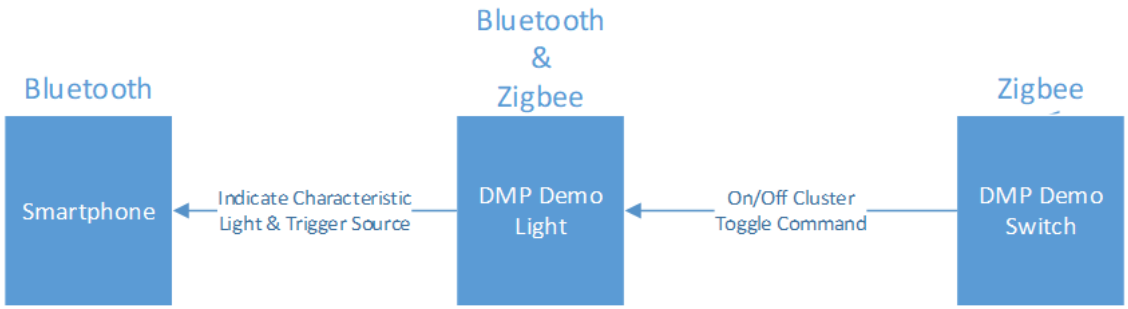
```
static void writeLightState(uint8_t connection, uint8array *writeValue)
{
  lightDirection = DMP_UI_DIRECTION_BLUETOOTH;
  emberAfWriteAttribute(emberAfPrimaryEndpoint(),
                        ZCL_ON_OFF_CLUSTER_ID,
                        ZCL_ON_OFF_ATTRIBUTE_ID,
                        CLUSTER_MASK_SERVER,
                        (int8u *)&writeValue->data[0],
                        ZCL_BOOLEAN_ATTRIBUTE_TYPE);
  <sl_bt_/gecko_cmd>gatt_server_send_user_write_response(
    connection,
    gattdb_light_state,
    ES_WRITE_OK
    );
}
```

The `emberAfWriteAttribute()` is used to write the attribute table of the Zigbee application with the value supplied by the Bluetooth connected device above. Since the on-off attribute of the on-off server cluster is a reportable attribute it is reported to all devices setup in the binding table of the Light.

The `emberAfOnOffClusterServerAttributeChangedCallback()` is then used to change the state of the LEDs and the LCD to indicate the state of the light on the WSTK main board.

Write from the Zigbee Connected Device

The flow in the other direction, that is a change in the Light state from Zigbee connected device, is shown in the following figure.



Any on-off client on the same network as the Light can send an on-off cluster's On, Off or Toggle command to the Light to change its state. Once such a command is received over the Zigbee interface the Silicon Labs Zigbee framework interprets it and calls an appropriate handler to change the value of on-off attribute of the on-off server cluster. In the example **DynamicMultiprotocolSwitch** application the on-off client sends a Toggle command to the Light, which toggles the value of the on-off attribute and triggers the `emberAfOnOffClusterServerAttributeChangedCallback()`. The callback is then used to change the state of the light as well as send notifications for both Trigger Source and Light characteristics to the connected Bluetooth devices and to update the LEDs and the LCD to indicate the change in the Light state.

```
void emberAfOnOffClusterServerAttributeChangedCallback(int8u endpoint,
                                                       EmberAfAttributeId attributeId)
{
  EmberStatus status;
  int8u data;

  if (attributeId == ZCL_ON_OFF_ATTRIBUTE_ID) {
    status =  emberAfReadAttribute(endpoint,
                                   ZCL_ON_OFF_CLUSTER_ID,
                                   ZCL_ON_OFF_ATTRIBUTE_ID,
                                   CLUSTER_MASK_SERVER,
                                   (int8u*)&data,
                                   sizeof(data),
                                   NULL);

    if (status == EMBER_ZCL_STATUS_SUCCESS) {
      if (data == 0x00) {
        halClearLed(BOARDLED0);
        halClearLed(BOARDLED1);
        dmpUiLightOff();
        notifyLight(currentConnection, 0);
      } else {
        halSetLed(BOARDLED0);
        halSetLed(BOARDLED1);
        notifyLight(currentConnection, 1);
        dmpUiLightOn();
      }
      if ( (lightDirection == DMP_UI_DIRECTION_BLUETOOTH)
           || (lightDirection == DMP_UI_DIRECTION_SWITCH) ) {
        dmpUiUpdateDirection(lightDirection);
      } else {
        lightDirection = DMP_UI_DIRECTION_ZIGBEE;
        dmpUiUpdateDirection(lightDirection);
      }
      ble_lastEvent = lightDirection;
      lightDirection = DMP_UI_DIRECTION_INVALID;

      if (ble_lastEvent != DMP_UI_DIRECTION_INVALID) {
        if    (    (ble_lightState_config    !=    GAT_RECEIVE_INDICATION)    &&    (ble_lastEvent_config    ==
GAT_RECEIVE_INDICATION)) {
          notifyTriggerSource(currentConnection, ble_lastEvent);
        }
      }
    }
  } else {
  }
}}
```

4 Document Revision History

Revision 0.9

- Documents the new freeRTOS support

Revision 0.8

- Removed the “Define the mbedTLS path” step in the procedure in section 2.2.
- Modified the 6.8.x code that illustrates adding `emberAfPluginBleEventCallback` code to the **[project_name]_callbacks.c** file,

Revision 0.7

- Update to reflect changes for EmberZNet 6.8.0/Bluetooth SDK 3.0.0.

Revision 0.6

- Section 2.2 re-inserted with functional instructions.
- Fixed duplicated bookmarks in PDF.

Revision 0.5

- Updated section 2.1. Temporarily removed section 2.2 on project configuration.

Revision 0.4

- Added note about conflict between LCD and external flash for OTA.

Revision 0.3

- Added note about threat-safe implementation to section 3.2.

Revision 0.2

- Modifications for supporting sleepy light device.

Revision 0.1

- Initial release

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 & Community
www.silabs.com/community

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.
Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

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

www.silabs.com