AN1418: Running Zigbee, OpenThread, and *Bluetooth*® Concurrently on a System-on-Chip

This document describes how to run a combination of Zigbee, OpenThread, and Bluetooth networking stacks and the Zigbee application layer on a System-on-Chip (SoC). One of the main functions of a Concurrent Multiprotocol (CMP) device is to act as a bridge between Zigbee and OpenThread networks.

Note that, depending on the chip, memory size restrictions may prevent running Matter on SoC devices.

KEY POINTS

- Important features of the sample application
- Making a Zigbee-OpenThread CMP application from a Z3Light

1 Introduction

This document describes a Concurrent Multiprotocol (CMP) application that runs Zigbee, Bluetooth, and OpenThread stacks on a single EFR32 radio. The primary use for such an application is to allow Zigbee line-powered devices to also be part of an OpenThread network simultaneously and therefore serve as a bridge between the two networks.

2 Concurrent Multiprotocol (CMP) Sample Application (z3-light_ot-ftd)

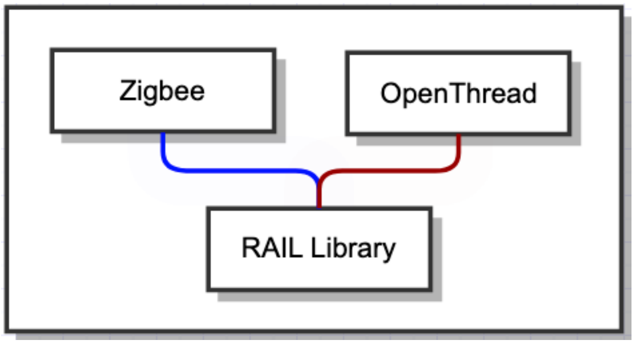


Figure 2-1. Zigbee + OpenThread Concurrent Multiprotocol Application

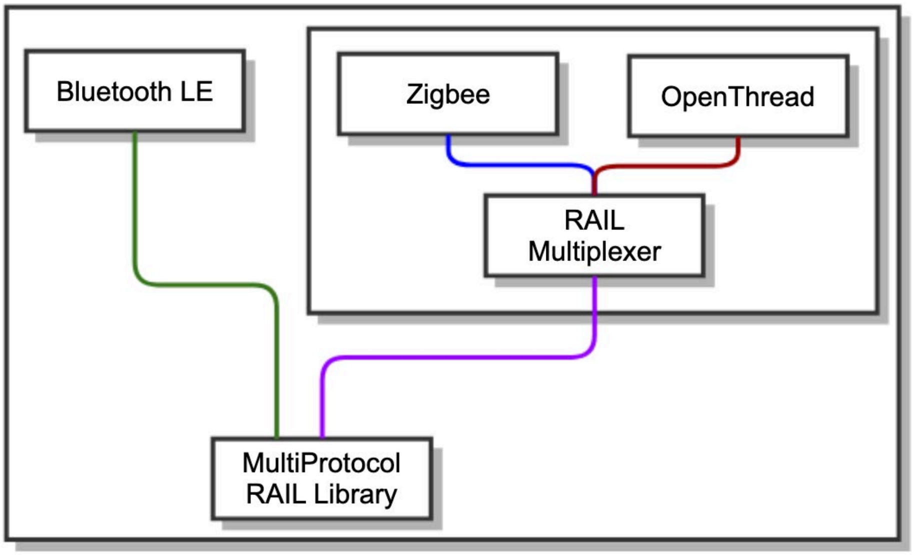


Figure 2-2. Zigbee + Bluetooth + OpenThread Concurrent Multiprotocol Application

The CMP sample application consists of a Z3Light, which is a Zigbee router, and an OpenThread FTD (Full Thread Device). Both protocol stacks operate by multiplexing a single EFR32 radio. Both protocols need to use the same radio channel to ensure proper operation. Bluetooth functionality can be added to the sample application by including additional components.

2.1 RTOS

Within the CMP application, scheduling is managed using a Real Time Operating System (RTOS). Each protocol runs in a dedicated RTOS task. The Zigbee and OpenThread tasks operate at the same priority while the Command Line Interface (CLI) is made available using a CLI RTOS task that operates at a lower priority.

Caution: It is critical to note that Zigbee and OpenThread APIs are not thread-safe. Calling them from different threads can result in unexpected behavior. In addition, any references to EmberMessageBuffer must be contained within the Zigbee task.

2.2 Command Line Interface

This application supports all CLI commands that can be found in the Z3Light sample application. A subset of the OpenThread CLI has been ported to demonstrate form, join and ping operations. This functionality can be extended further, if necessary, by following the example commands in the ot_up_cli.c file from the ot_up_cli component. Note that OpenThread APIs are only invoked from sl_ot_rtos_application_tick since they are not thread-safe.

2.2.1 OpenThread Commissioning

This device can be commissioned on to an OpenThread network out-of-band using CLI commands. Setting the OpenThread network parameters, such as network key and channel, before starting the network allows the CMP device to join a Thread network as a child or router device.

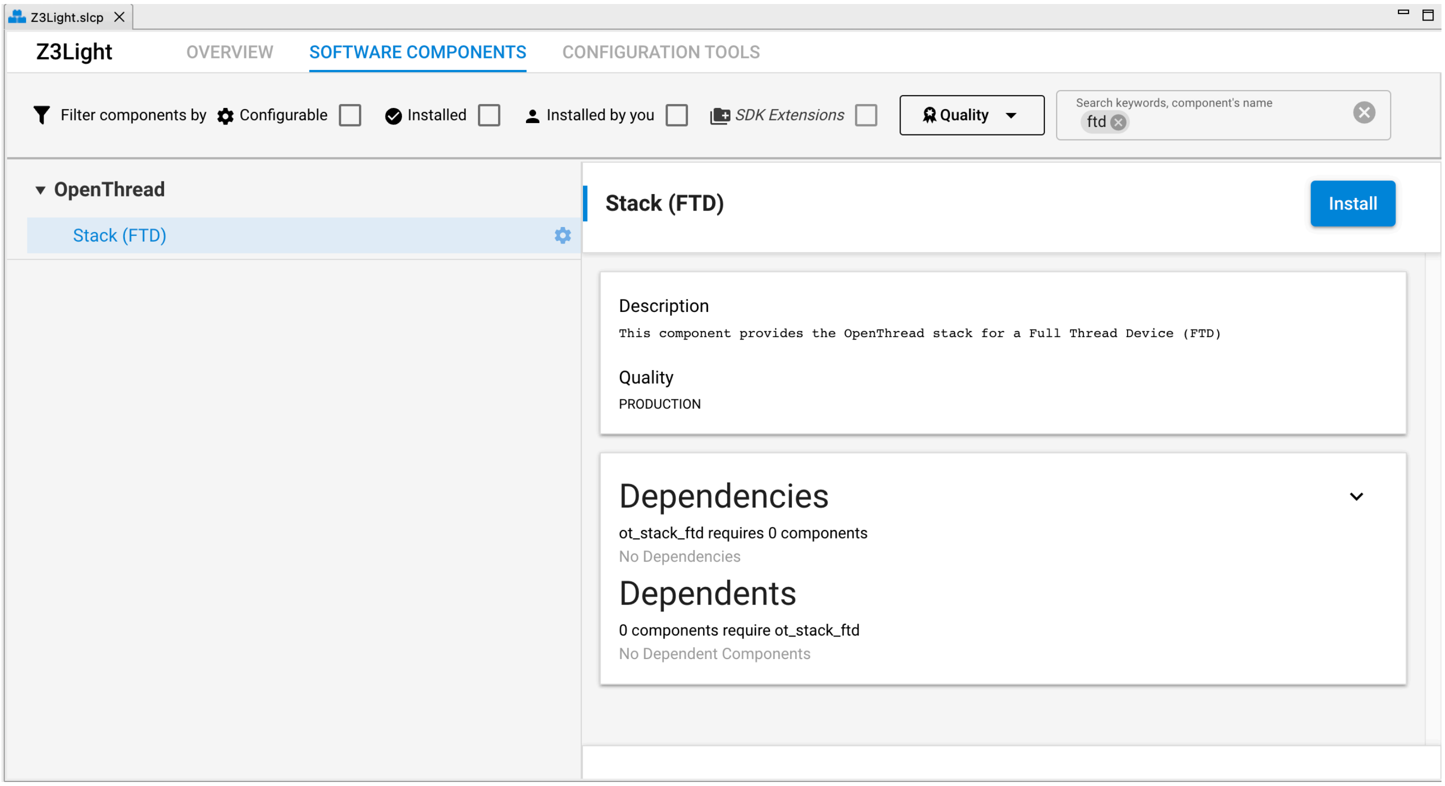
Table 2.1. OpenThread CLI commands

CLI Command	Description
dataset	View OpenThread network configuration.
dataset_new	Creates a new OpenThread dataset.
dataset_commit_active	Commits dataset to NVM.
factory_reset	Removes all NVM OpenThread settings.
dataset_networkkey	Presets the network key on the device to help with joining an existing OpenThread network out-of-band.
dataset_channel	Presets the radio channel used by the OpenThread network. This command can be used to force both Zigbee and OpenThread networks to use the same radio channel.
dataset_pan_id	Presets the PAN ID on the device to help with joining the OpenThread network out-of-band.
dataset_extended_pan_id	Presets the extended PAN ID on the device to help with joining the OpenThread network out-of-band.
ifconfig_up	Enables OpenThread interface.
thread_start	Enables and attaches OpenThread protocol operation.
thread_state	Reads current status: offline, disabled, detached, child, router, or leader.

3 Converting a Zigbee Application into a Zigbee-OpenThread CMP Application

This section describes the steps involved in converting a Z3Light into a Concurrent Multiprotocol application that includes the OpenThread stack.

1. Use the Simplicity Studio “Create New Project” wizard to create a Zigbee – SoC Light project for your board of choice.
2. Open the Software Components tab of the generated project to add the OpenThread > **Stack (FTD)** component. Note that the addition of this component automatically adds a Real Time Operating System (RTOS) to the project.



Stack (FTD)

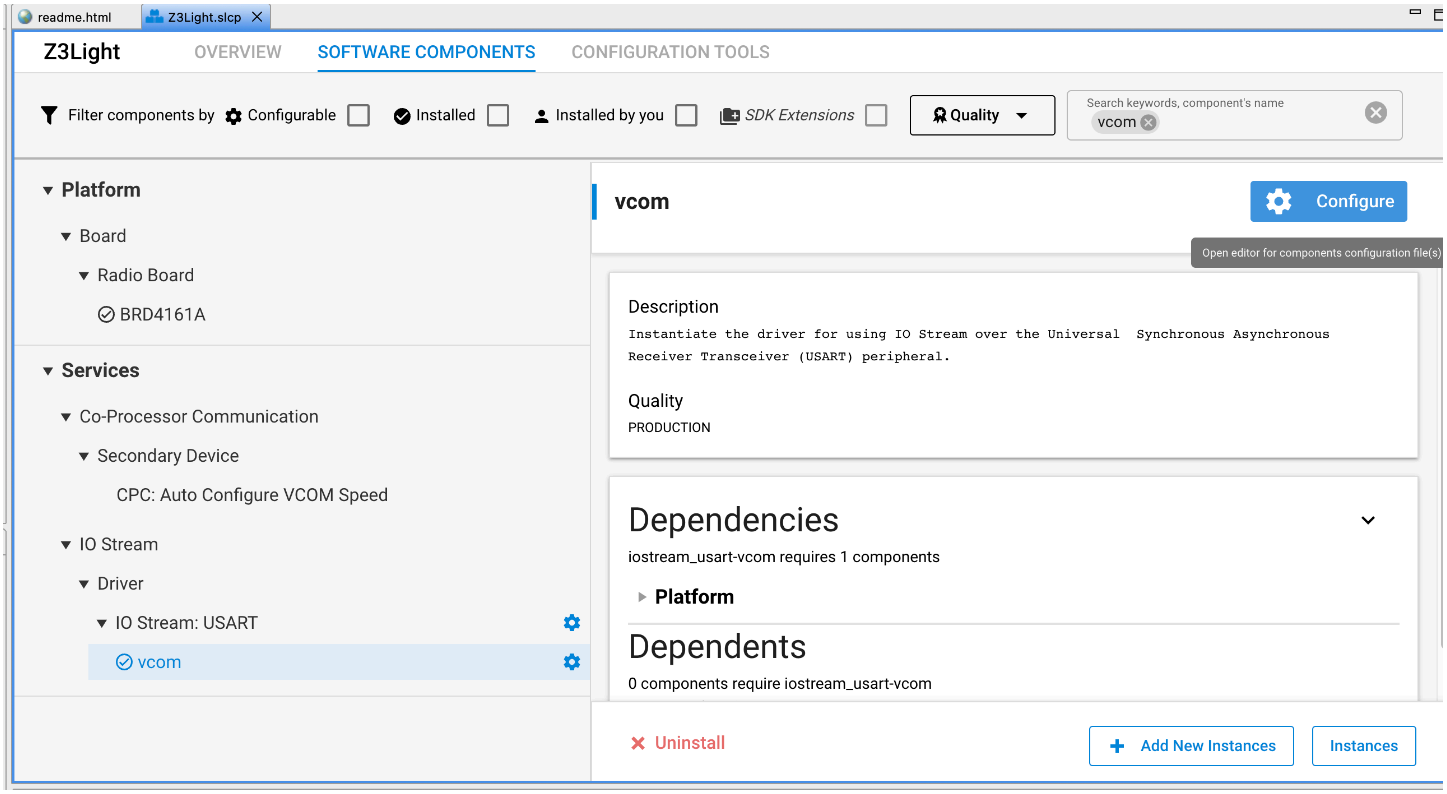
Description
This component provides the OpenThread stack for a Full Thread Device (FTD)

Quality
PRODUCTION

Dependencies
ot_stack_ftd requires 0 components
No Dependencies

Dependents
0 components require ot_stack_ftd
No Dependent Components

3. Select the IO Stream > Driver > IOS Stream: USART > **vcom** component and configure it by increasing “Receive buffer size” to 128.



readme.html | Z3Light.slcp

Z3Light — OVERVIEW | SOFTWARE COMPONENTS | CONFIGURATION TOOLS

Filter components by: Configurable □ Installed □ Installed by you □ *SDK Extensions* □ Quality ▾

Search keywords, component's name: vcom

- Platform
  - Board
    - Radio Board
      - BRD4161A
- Services
  - Co-Processor Communication
    - Secondary Device
      - CPC: Auto Configure VCOM Speed
  - IO Stream
    - Driver
      - IO Stream: USART
        - vcom

vcom — Configure (Open editor for components configuration file(s))

Description

```
Instantiate the driver for using IO Stream over the Universal  Synchronous Asynchronous
Receiver Transceiver (USART) peripheral.
```

Quality

PRODUCTION

Dependencies

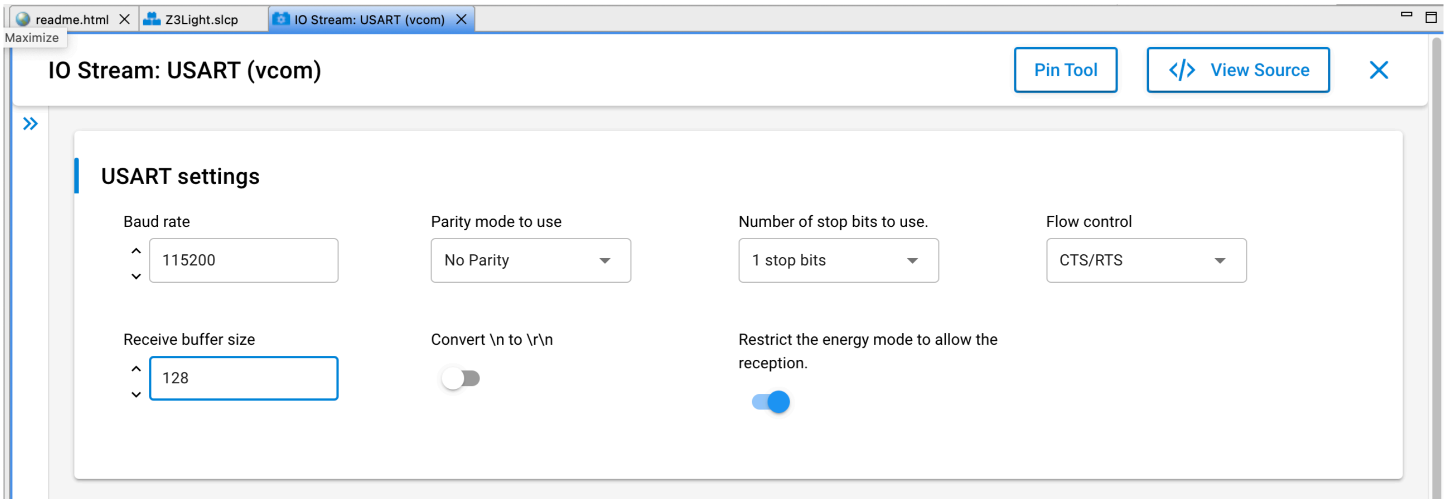
iostream_usart-vcom requires 1 components

Platform

Dependents

0 components require iostream_usart-vcom

✕ Uninstall | + Add New Instances | Instances



readme.html | Z3Light.slcp | IO Stream: USART (vcom)

Maximize

IO Stream: USART (vcom) — Pin Tool | View Source | ✕

USART settings

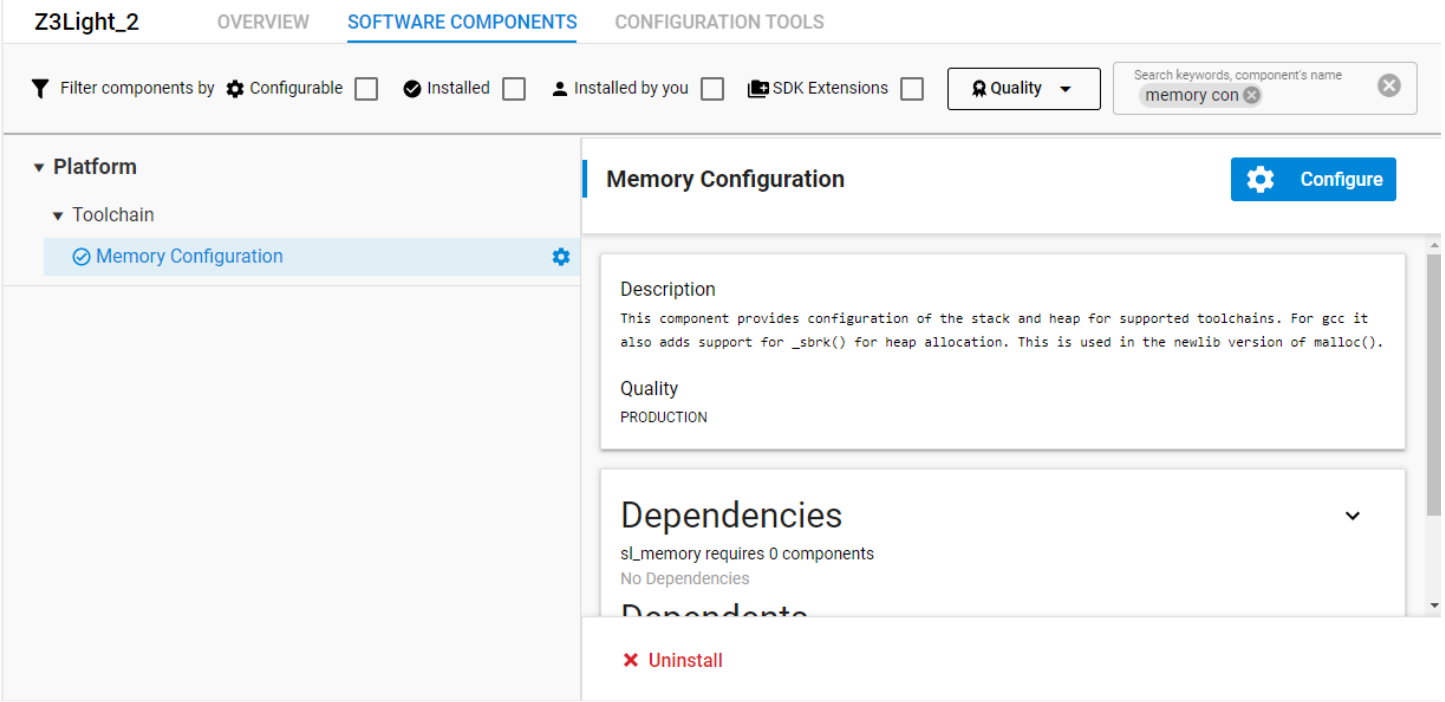
Baud rate	Parity mode to use	Number of stop bits to use.	Flow control
115200	No Parity	1 stop bits	CTS/RTS

Receive buffer size: 128

Convert \n to \r\n: ○ off

Restrict the energy mode to allow the reception.: ● on

4. Select the Toolchain > **Memory Configuration** component and configure it by increasing stack size to 4608 and heap size to 16384 to account for the addition of OpenThread networking stack.



Description

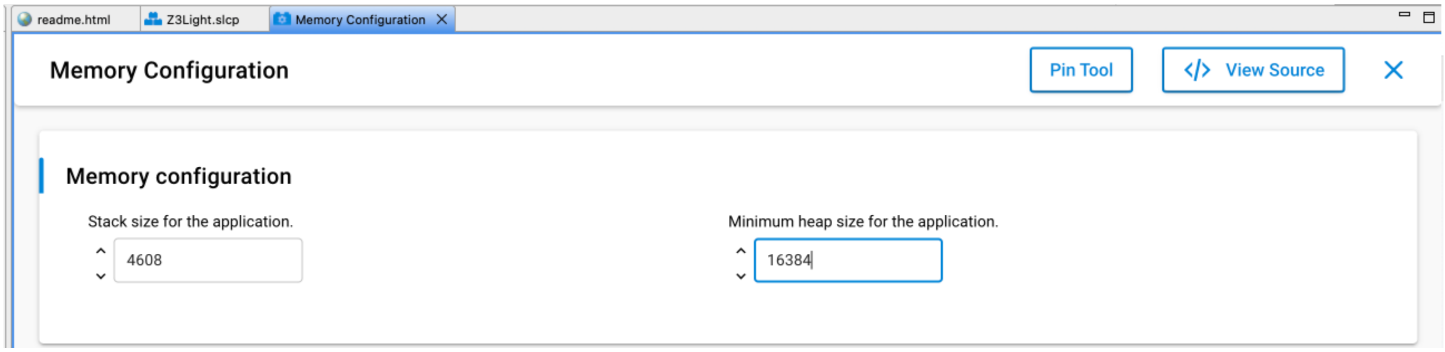
```
This component provides configuration of the stack and heap for supported toolchains. For gcc it
also adds support for _sbrk() for heap allocation. This is used in the newlib version of malloc().
```

Quality

PRODUCTION

sl_memory requires 0 components

No Dependencies



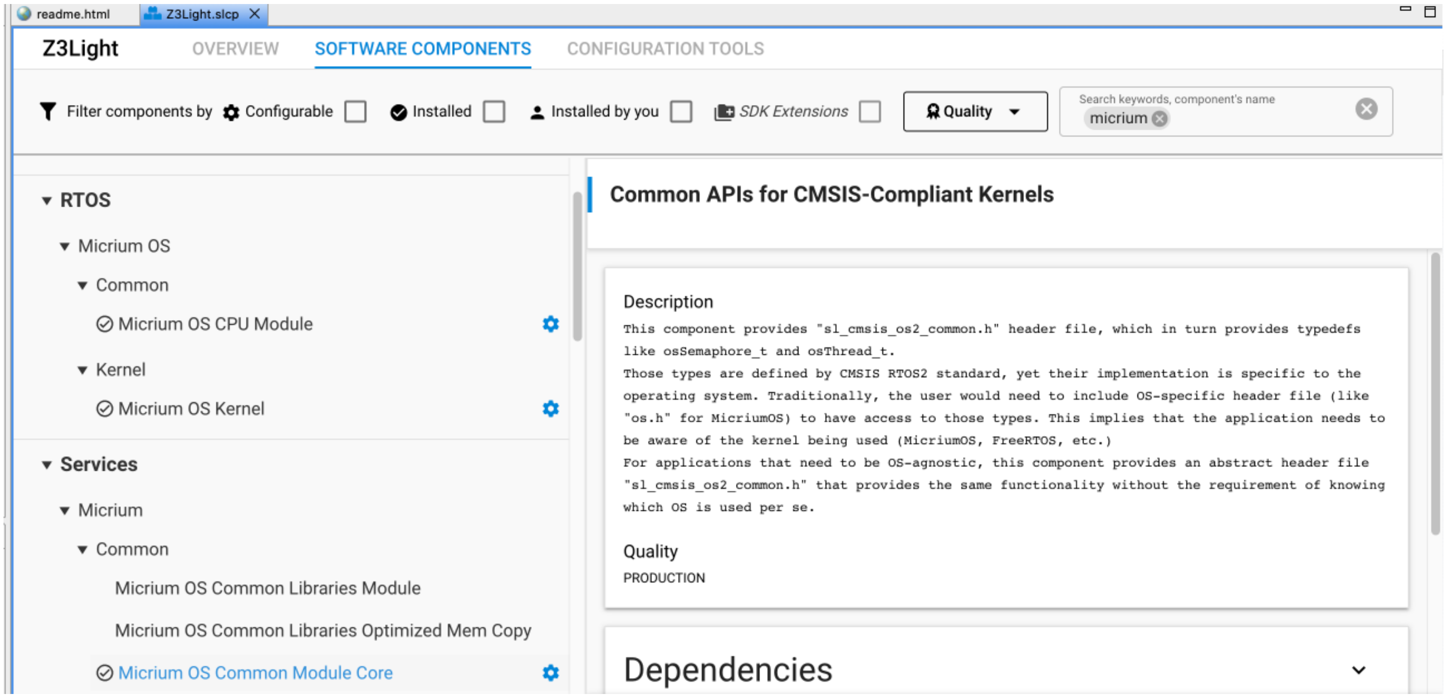
Stack size for the application.

4608

Minimum heap size for the application.

16384

5. Select Micrium > Common > **Micrium OS Common Module Core** component and configure it to decrease “size of heap memory” to 0 to prevent Micrium RTOS from allocating its own heap memory.

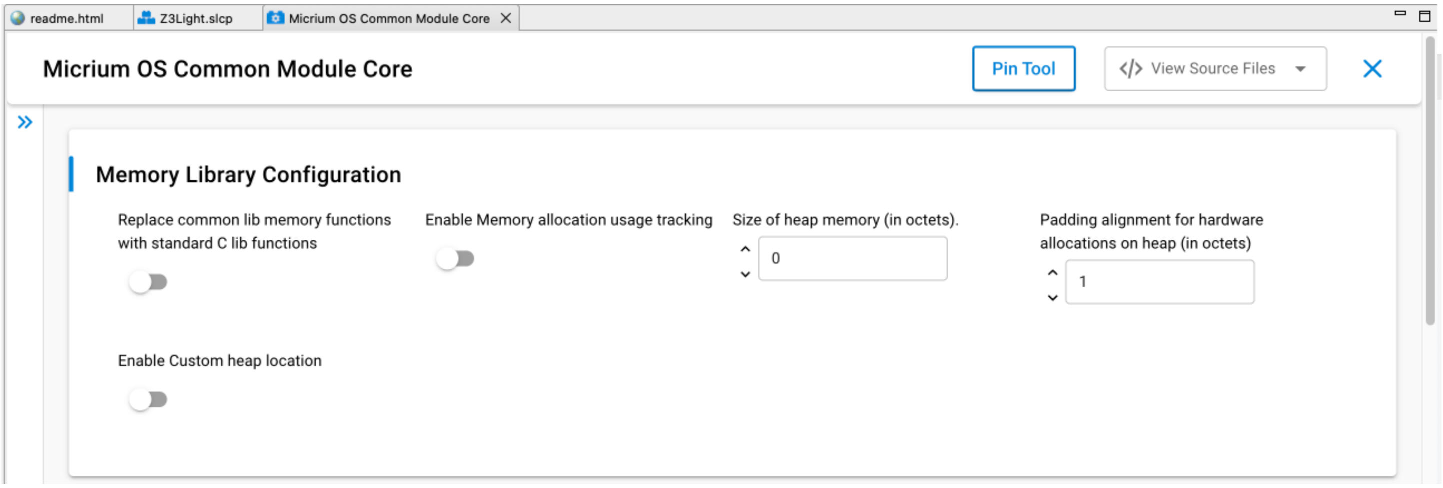


Common APIs for CMSIS-Compliant Kernels

Description

This component provides "sl_cmsis_os2_common.h" header file, which in turn provides typedefs like osSemaphore_t and osThread_t.
Those types are defined by CMSIS RTOS2 standard, yet their implementation is specific to the operating system. Traditionally, the user would need to include OS-specific header file (like "os.h" for MicriumOS) to have access to those types. This implies that the application needs to be aware of the kernel being used (MicriumOS, FreeRTOS, etc.)
For applications that need to be OS-agnostic, this component provides an abstract header file "sl_cmsis_os2_common.h" that provides the same functionality without the requirement of knowing which OS is used per se.

Quality
PRODUCTION



Micrium OS Common Module Core

Memory Library Configuration

Replace common lib memory functions with standard C lib functions: off

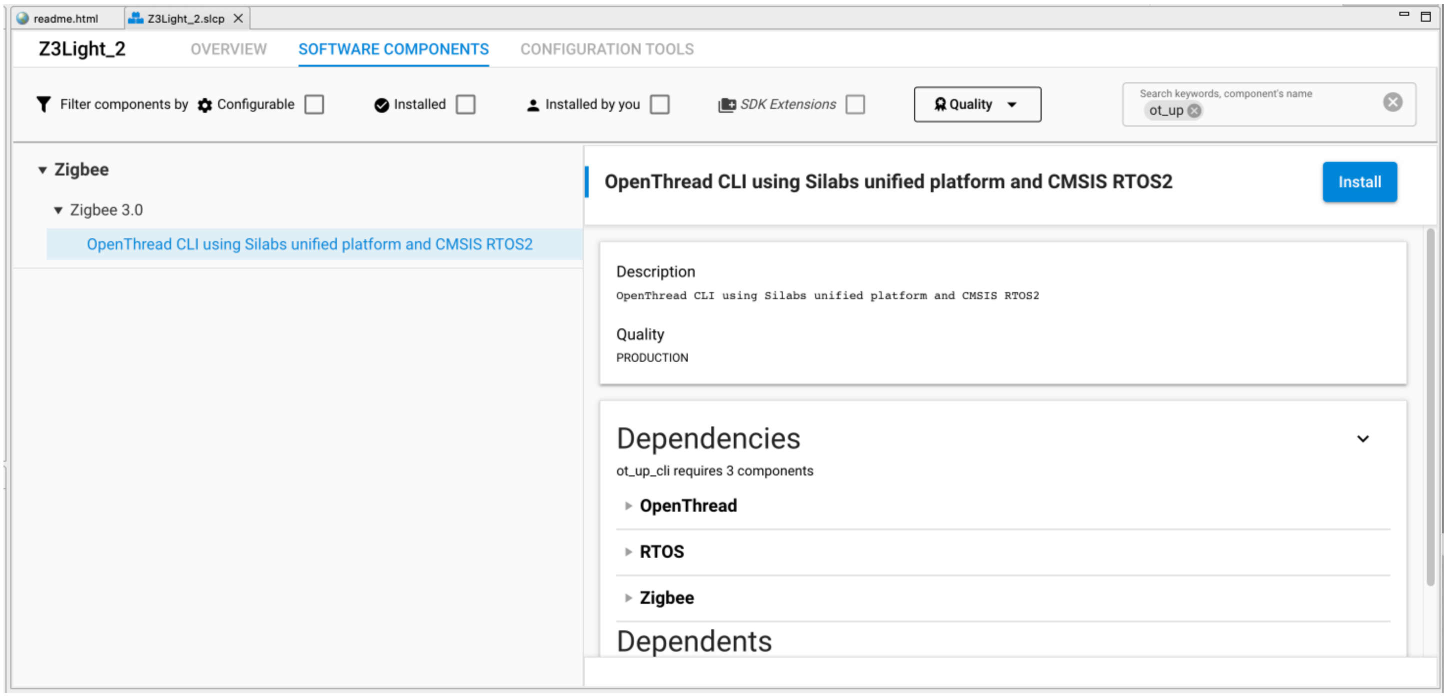
Enable Memory allocation usage tracking: off

Size of heap memory (in octets).: 0

Padding alignment for hardware allocations on heap (in octets): 1

Enable Custom heap location: off

6. Add OpenThread CLI commands by installing the Zigbee > Zigbee 3.0 > **OpenThread CLI using Silabs unified platform** (ot_up_cli) component.



Search: ot_up

- Zigbee
  - Zigbee 3.0
    - OpenThread CLI using Silabs unified platform and CMSIS RTOS2

OpenThread CLI using Silabs unified platform and CMSIS RTOS2

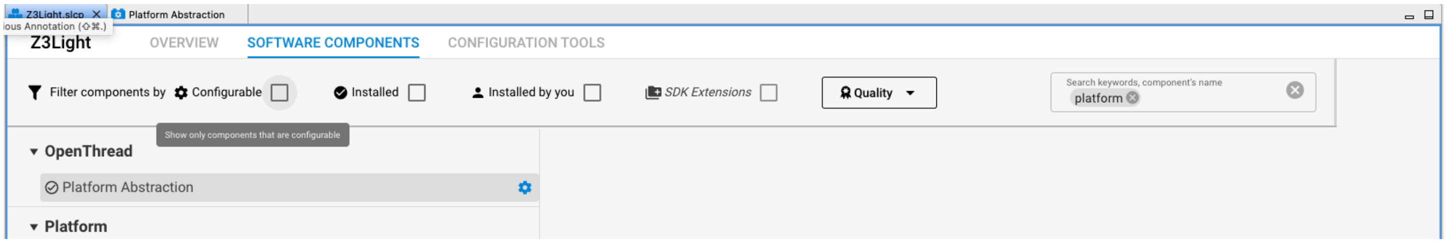
Description
OpenThread CLI using Silabs unified platform and CMSIS RTOS2

Quality
PRODUCTION

Dependencies
ot_up_cli requires 3 components
- OpenThread
- RTOS
- Zigbee

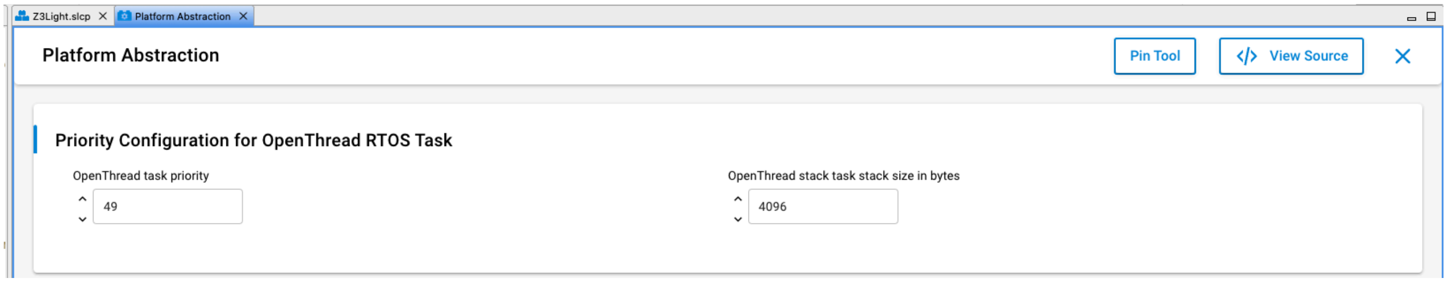
Dependents

7. Select the OpenThread > **Platform Abstraction** component and configure it by setting priority to 49 to match the Zigbee RTOS task priority.



Search: platform

- OpenThread
  - Platform Abstraction
- Platform



Platform Abstraction

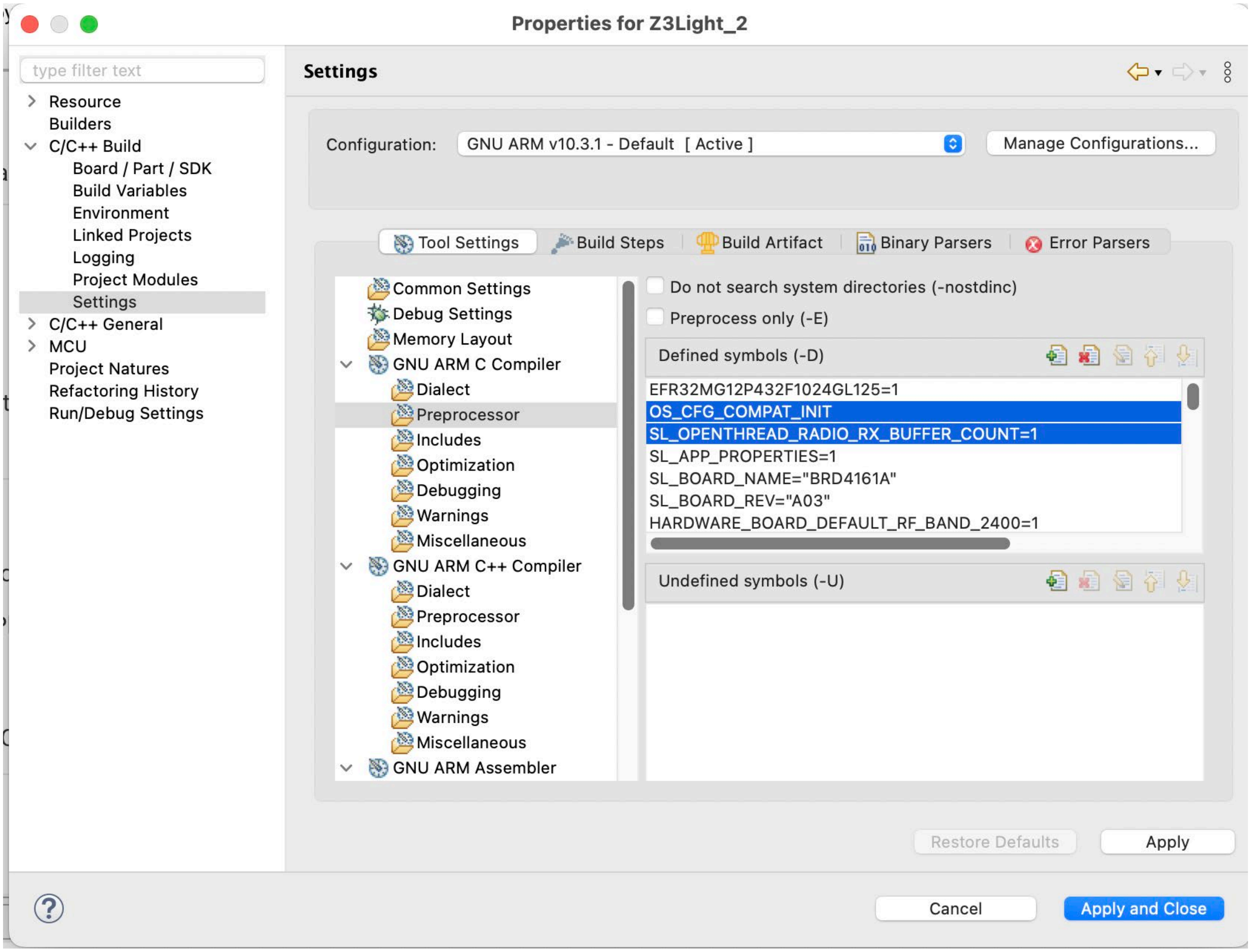
Priority Configuration for OpenThread RTOS Task

OpenThread task priority: 49

OpenThread stack task stack size in bytes: 4096

8. Right click the Z3Light project in Simplicity Studio's Project Explorer view and click Properties. Open C/C++ Build Settings and Under GNU ARM C Compiler, select Preprocessor. Add two preprocessor define symbols:
   - OS_CFG_COMPAT_INIT (Used in conjunction with LIB_MEM_CFG_HEAP_SIZE to allow the application to handle heap allocation)
   - SL_OPENTHREAD_RADIO_RX_BUFFER_COUNT=1 (This is a workaround for an issue where the Zigbee network cannot send beacons when the OpenThread network is up)

   Click Apply and Close to save.



Configuration: GNU ARM v10.3.1 - Default [Active]

□ Do not search system directories (-nostdinc)

□ Preprocess only (-E)

Defined symbols (-D)

```
EFR32MG12P432F1024GL125=1
OS_CFG_COMPAT_INIT
SL_OPENTHREAD_RADIO_RX_BUFFER_COUNT=1
SL_APP_PROPERTIES=1
SL_BOARD_NAME="BRD4161A"
SL_BOARD_REV="A03"
HARDWARE_BOARD_DEFAULT_RF_BAND_2400=1
```

Undefined symbols (-U)

9. Open app.c file in the project folder and add the code below to the beginning of the file to initialize OpenThread. Save file and build the project.

```
#if defined(OPENTHREAD_FTD)
  #include <assert.h>
  #include <openthread-core-config.h>
  #include <openthread/config.h>

  #include <openthread/ncp.h>
  #include <openthread/diag.h>
  #include <openthread/tasklet.h>

  #include "openthread-system.h"
```

```
static otInstance *     sInstance       = NULL;

void sl_ot_create_instance(void)
{
  #if OPENTHREAD_CONFIG_MULTIPLE_INSTANCE_ENABLE
  size_t   otInstanceBufferLength = 0;
  uint8_t *otInstanceBuffer       = NULL;

  // Call to query the buffer size
  (void)otInstanceInit(NULL, &otInstanceBufferLength);

  // Call to allocate the buffer
  otInstanceBuffer = (uint8_t *)malloc(otInstanceBufferLength);
  assert(otInstanceBuffer);

  // Initialize OpenThread with the buffer
  sInstance = otInstanceInit(otInstanceBuffer, &otInstanceBufferLength);
  #else
  sInstance = otInstanceInitSingle();
  #endif
  assert(sInstance);
}

otInstance *otGetInstance(void)
{
  return sInstance;
}
#endif //#if defined(OPENTHREAD_FTD)
```

This application can now form a distributed Zigbee network or join any Zigbee network (centralized or distributed). It can also function as a leader, child, or router on the OpenThread network.

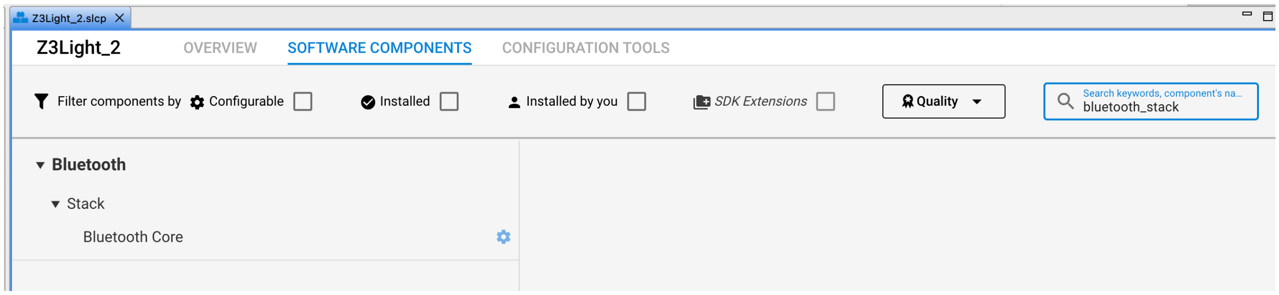
Caution: It is imperative to ensure that both networks operate on the same radio channel.

Any channel changes will need to be done in a controlled fashion. A channel change on one protocol's network can cause the other protocol to stop working until its network is also switched to the same channel. It is important to note that only certain Zigbee device types (trust center) may initiate a channel change on the Zigbee side.

3.1 Optional - Adding Bluetooth to the Concurrent Multiprotocol Application:

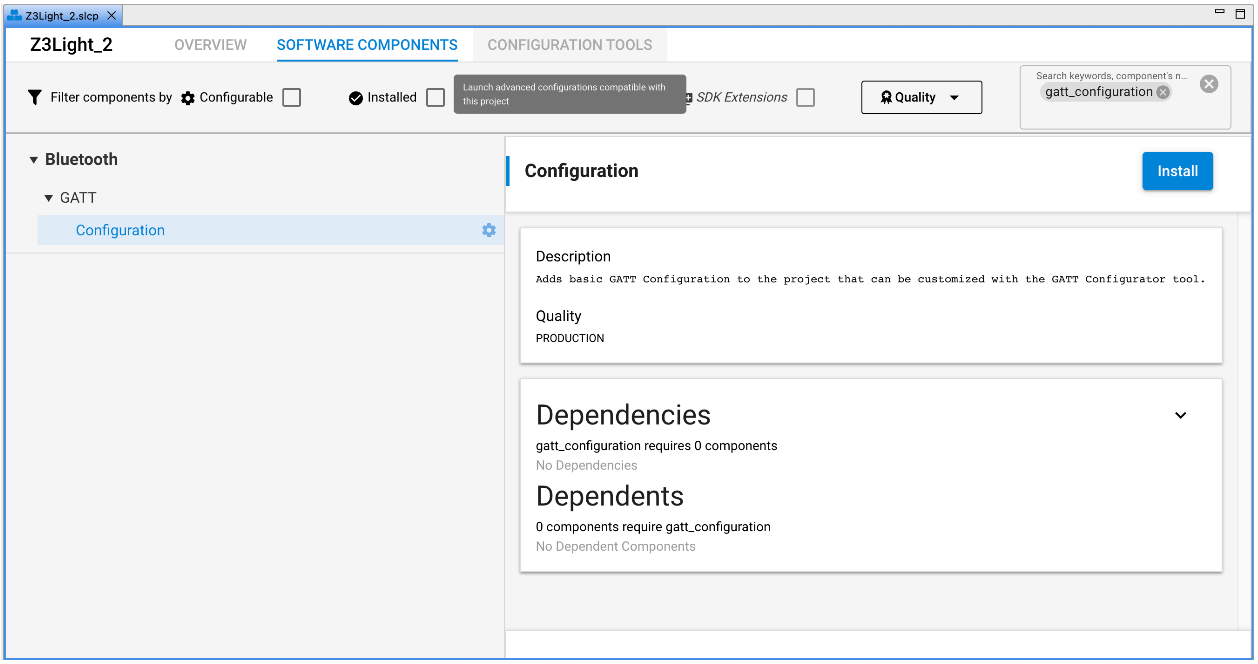
This section describes the steps involved in adding Bluetooth to the above application. Search for and install the following components:

- bluetooth_stack in the software components



Search keywords, component's na... bluetooth_stack

- gatt_configuration



Configuration

Description

Adds basic GATT Configuration to the project that can be customized with the GATT Configurator tool.

Quality

PRODUCTION

Dependencies

gatt_configuration requires 0 components

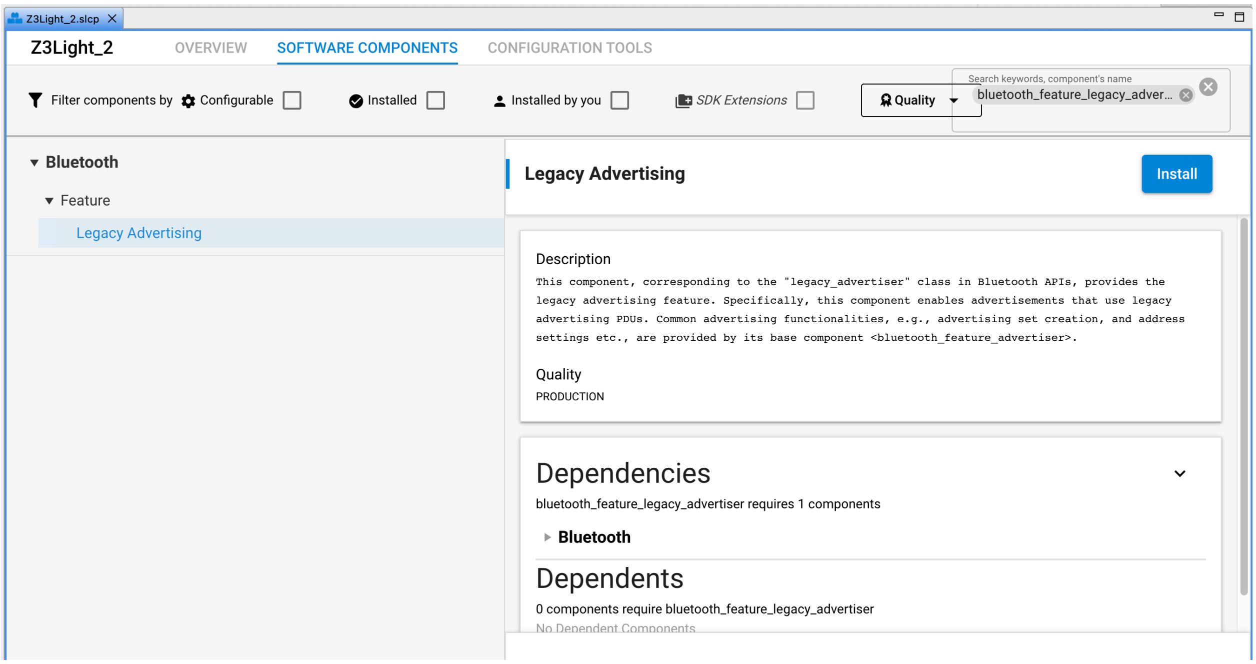
No Dependencies

Dependents

0 components require gatt_configuration

No Dependent Components

- Bluetooth_feature_legacy_advertiser



Legacy Advertising

Description

This component, corresponding to the "legacy_advertiser" class in Bluetooth APIs, provides the legacy advertising feature. Specifically, this component enables advertisements that use legacy advertising PDUs. Common advertising functionalities, e.g., advertising set creation, and address settings etc., are provided by its base component <bluetooth_feature_advertiser>.

Quality

PRODUCTION

Dependencies

bluetooth_feature_legacy_advertiser requires 1 components

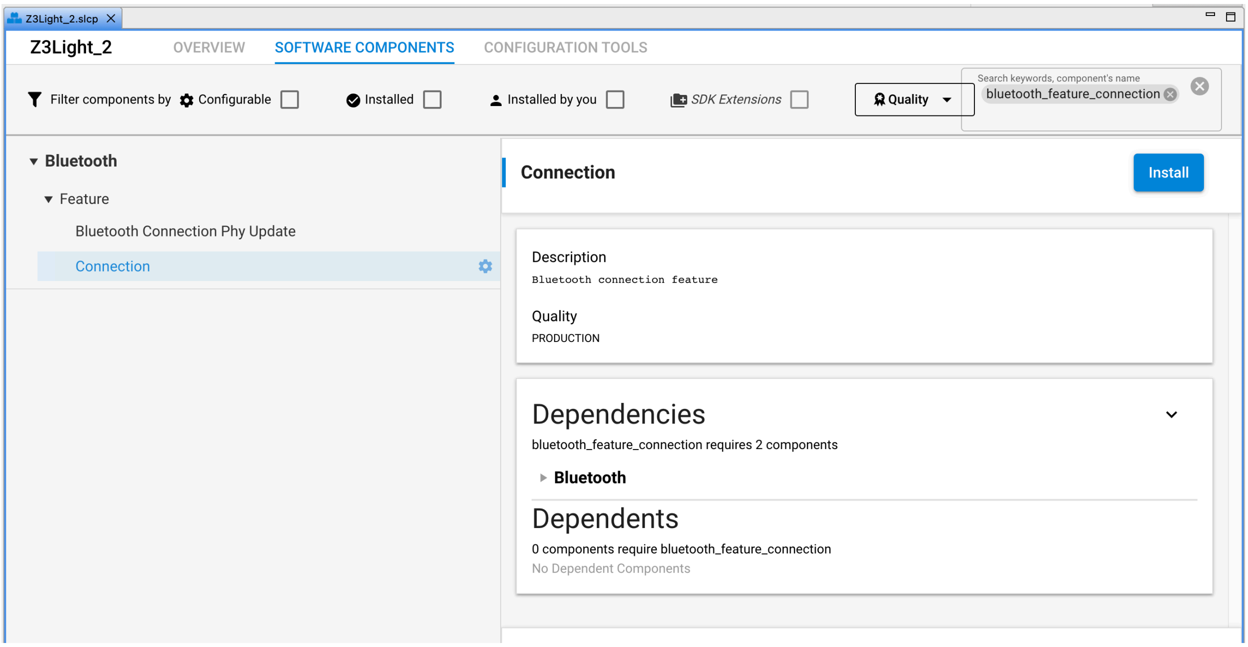
Bluetooth

Dependents

0 components require bluetooth_feature_legacy_advertiser

No Dependent Components

- Bluetooth_feature_connection



Connection

Description

Bluetooth connection feature

Quality

PRODUCTION

Dependencies

bluetooth_feature_connection requires 2 components

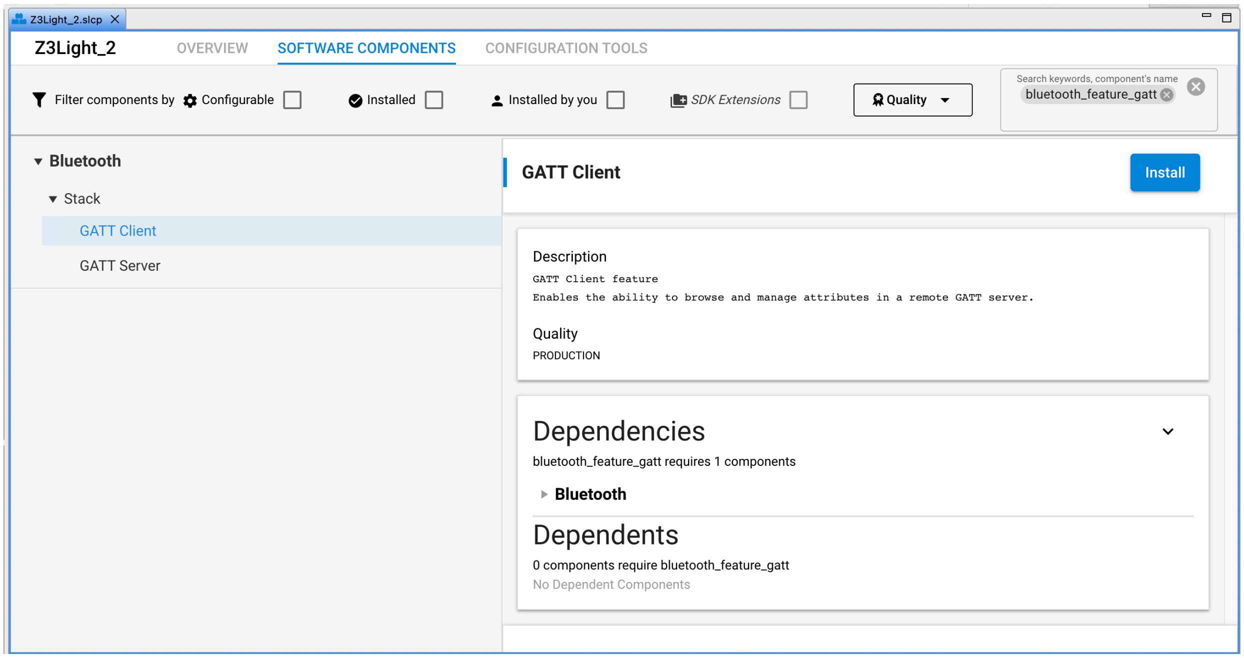
Bluetooth

Dependents

0 components require bluetooth_feature_connection

No Dependent Components

- bluetooth_feature_gatt – Install the GATT Client and GATT Server



GATT Client

Description

GATT Client feature
Enables the ability to browse and manage attributes in a remote GATT server.

Quality

PRODUCTION

Dependencies

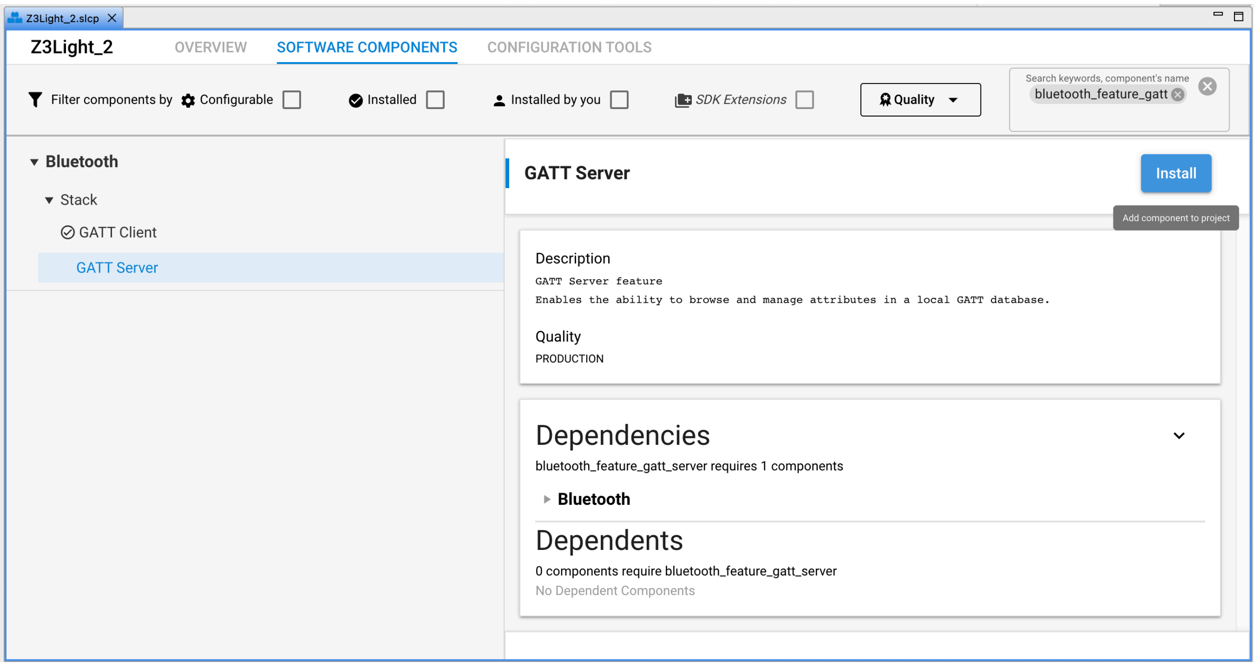
bluetooth_feature_gatt requires 1 components

Bluetooth

Dependents

0 components require bluetooth_feature_gatt

No Dependent Components



GATT Server

Description

```
GATT Server feature
Enables the ability to browse and manage attributes in a local GATT database.
```

Quality

PRODUCTION

Dependencies

bluetooth_feature_gatt_server requires 1 components

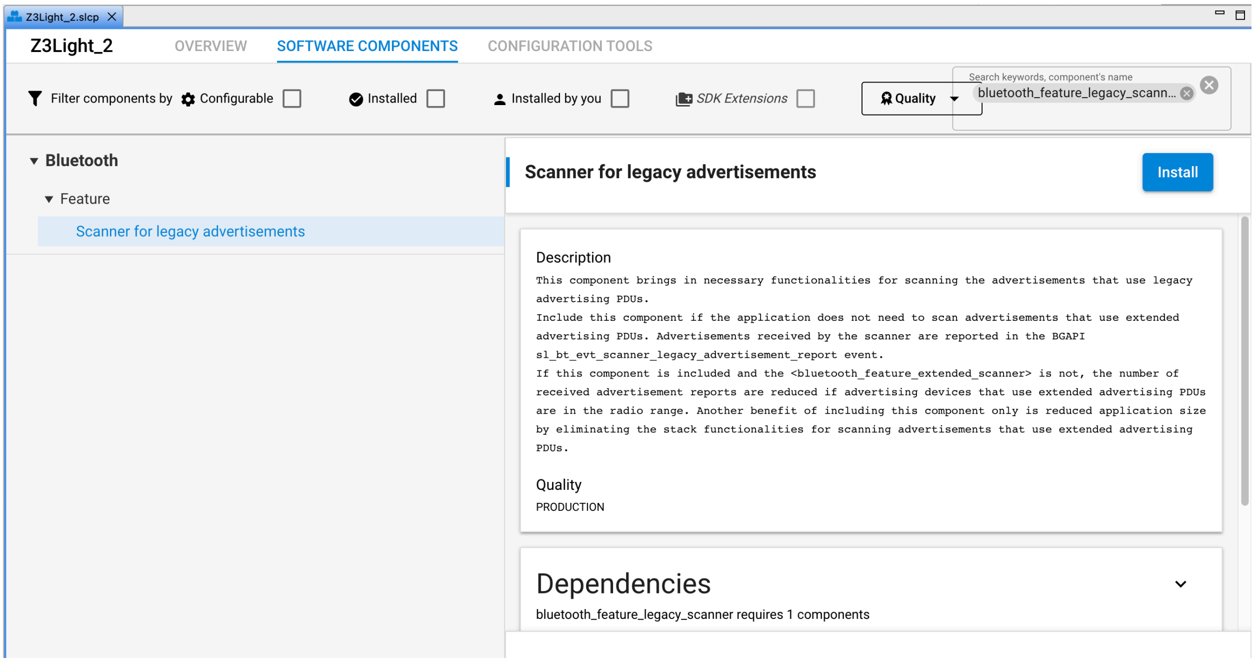
Bluetooth

Dependents

0 components require bluetooth_feature_gatt_server

No Dependent Components

- bluetooth_feature_legacy_scanner



Scanner for legacy advertisements

Description

```
This component brings in necessary functionalities for scanning the advertisements that use legacy
advertising PDUs.
Include this component if the application does not need to scan advertisements that use extended
advertising PDUs. Advertisements received by the scanner are reported in the BGAPI
sl_bt_evt_scanner_legacy_advertisement_report event.
If this component is included and the <bluetooth_feature_extended_scanner> is not, the number of
received advertisement reports are reduced if advertising devices that use extended advertising PDUs
are in the radio range. Another benefit of including this component only is reduced application size
by eliminating the stack functionalities for scanning advertisements that use extended advertising
PDUs.
```

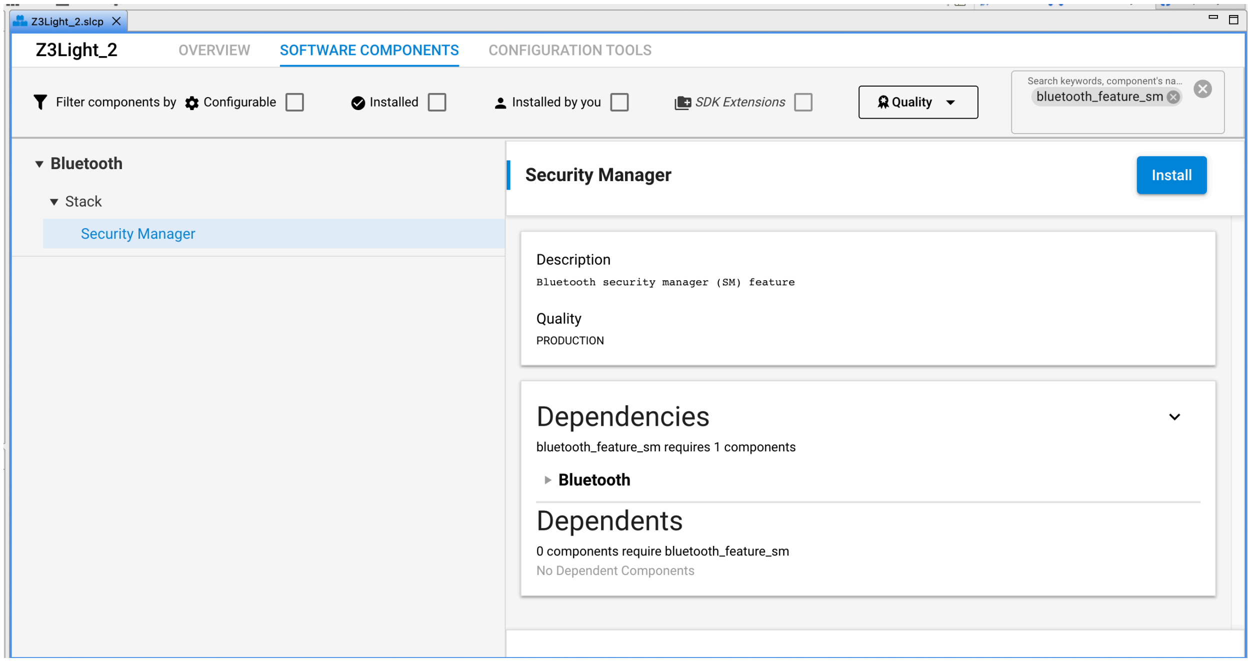
Quality

PRODUCTION

Dependencies

bluetooth_feature_legacy_scanner requires 1 components

- bluetooth_feature_sm



Z3Light_2.slcp

Z3Light_2 OVERVIEW SOFTWARE COMPONENTS CONFIGURATION TOOLS

Filter components by Configurable □ Installed □ Installed by you □ SDK Extensions □ Quality

Search keywords, component's na... bluetooth_feature_sm

Bluetooth
Stack
Security Manager

Security Manager Install

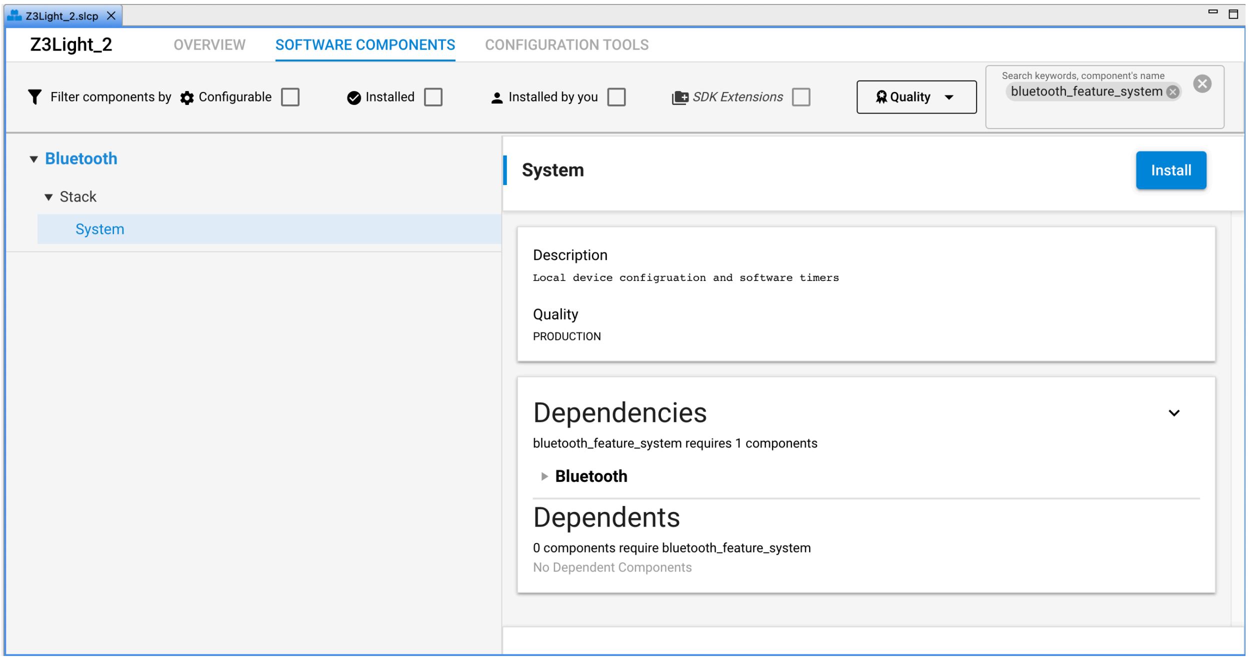
Description
`Bluetooth security manager (SM) feature`

Quality
PRODUCTION

Dependencies
bluetooth_feature_sm requires 1 components
Bluetooth

Dependents
0 components require bluetooth_feature_sm
No Dependent Components

- Bluetooth_feature_system



Z3Light_2.slcp

Z3Light_2 OVERVIEW SOFTWARE COMPONENTS CONFIGURATION TOOLS

Filter components by Configurable □ Installed □ Installed by you □ SDK Extensions □ Quality

Search keywords, component's name bluetooth_feature_system

Bluetooth
Stack
System

System Install

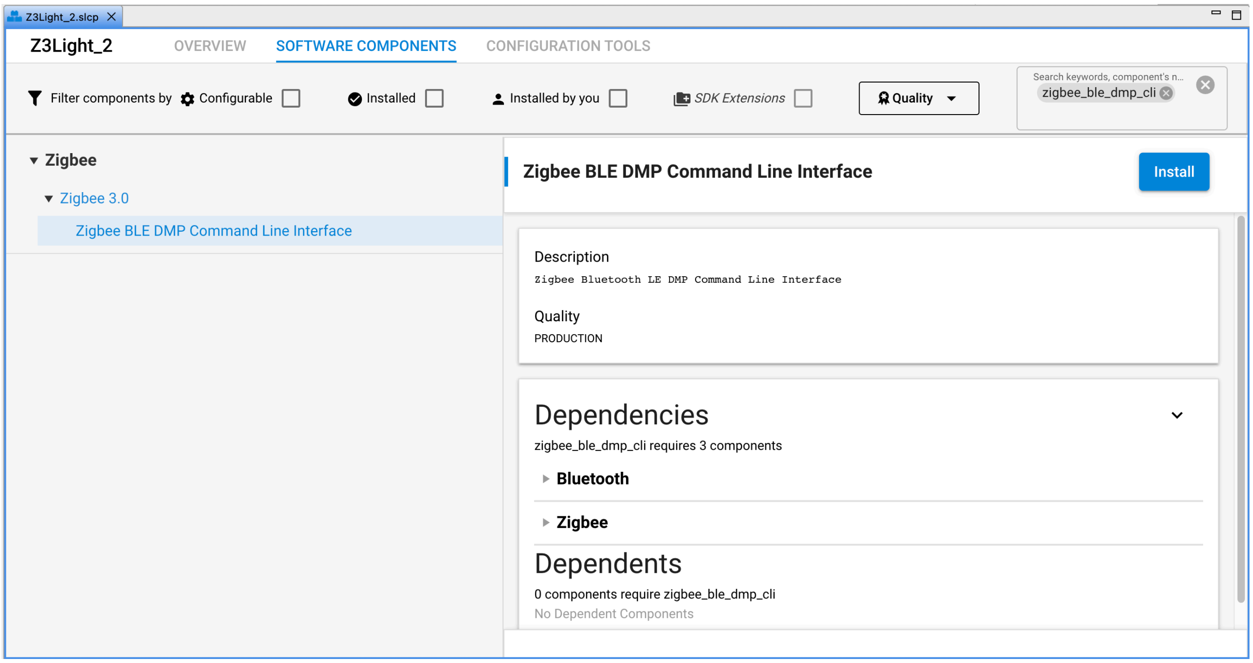
Description
`Local device configruation and software timers`

Quality
PRODUCTION

Dependencies
bluetooth_feature_system requires 1 components
Bluetooth

Dependents
0 components require bluetooth_feature_system
No Dependent Components

- zigbee_ble_dmp_cli



zigbee_ble_dmp_cli

- Zigbee
  - Zigbee 3.0
    - Zigbee BLE DMP Command Line Interface

Zigbee BLE DMP Command Line Interface

Description
`Zigbee Bluetooth LE DMP Command Line Interface`

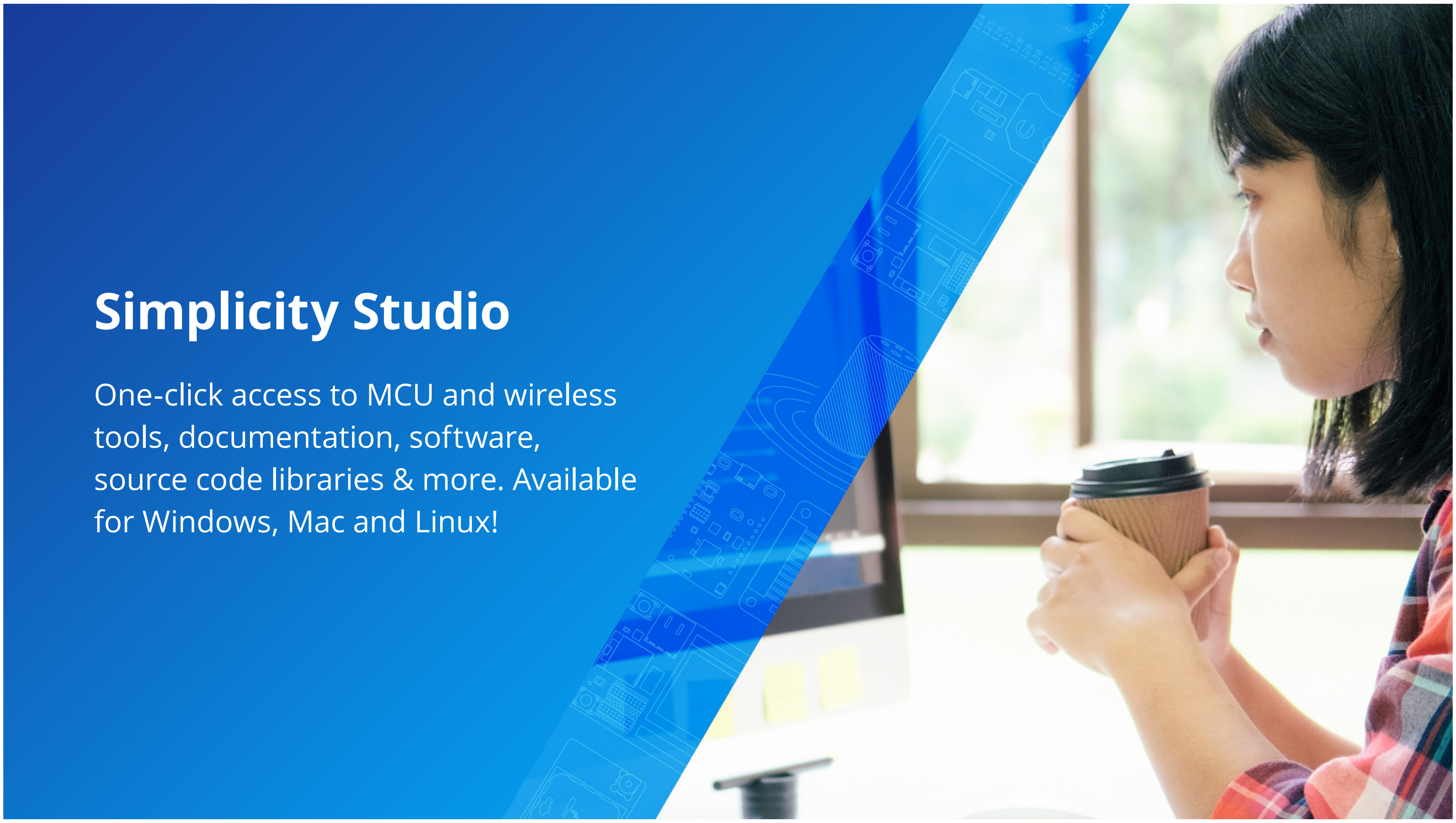
Quality
PRODUCTION

Dependencies
zigbee_ble_dmp_cli requires 3 components
- Bluetooth
- Zigbee

Dependents
0 components require zigbee_ble_dmp_cli
No Dependent Components

In addition to the above, add the following snippet of code in the project app.c file. This sample code provides an implementation for the Bluetooth event handler (sl_bt_on_event function)

```
#Include “sl_component_catalog.h”
#ifdef SL_CATALOG_BLUETOOTH_PRESENT

//------------------------------------------------------------------------------
// Bluetooth Event handler

#include "zigbee_app_framework_event.h"
#include "zigbee_app_framework_common.h"
#include "sl_bluetooth.h"
#include "sl_bluetooth_advertiser_config.h"
#include "sl_bluetooth_connection_config.h"
#include "sl_component_catalog.h"
static uint8_t cli_adv_handle;
void zb_ble_dmp_print_ble_address(uint8_t *address)
{
  sl_zigbee_app_debug_print("\nBLE address: [%02X %02X %02X %02X %02X %02X]\n",
                            address[5], address[4], address[3],
                            address[2], address[1], address[0]);
}

void sl_bt_on_event(sl_bt_msg_t* evt)
{
  switch (SL_BT_MSG_ID(evt->header)) {
    case sl_bt_evt_system_boot_id: {
      bd_addr ble_address;
      uint8_t type;
      sl_status_t status = sl_bt_system_hello();
      sl_zigbee_app_debug_println("BLE hello: %s",
                                  (status == SL_STATUS_OK) ? "success" : "error");

      #define SCAN_WINDOW 5
      #define SCAN_INTERVAL 10

      status = sl_bt_scanner_set_parameters(sl_bt_scanner_scan_mode_active,
                                            (uint16_t)SCAN_INTERVAL,
                                            (uint16_t)SCAN_WINDOW);
      status = sl_bt_system_get_identity_address(&ble_address, &type);
      zb_ble_dmp_print_ble_address(ble_address.addr);

      status = sl_bt_advertiser_create_set(&cli_adv_handle);
      if (status) {
        sl_zigbee_app_debug_println("sl_bt_advertiser_create_set status 0x%02x", status);
      }
    }
    break;

    case sl_bt_evt_connection_opened_id: {
      sl_zigbee_app_debug_println("sl_bt_evt_connection_opened_id \n");
      sl_bt_evt_connection_opened_t *conn_evt =
        (sl_bt_evt_connection_opened_t*) &(evt->data);
      sl_bt_connection_set_preferred_phy(conn_evt->connection, sl_bt_test_phy_1m, 0xff);
      sl_zigbee_app_debug_println("BLE connection opened");
    }
    break;
    case sl_bt_evt_connection_phy_status_id: {
      sl_bt_evt_connection_phy_status_t *conn_evt =
        (sl_bt_evt_connection_phy_status_t *)&(evt->data);
      // indicate the PHY that has been selected
      sl_zigbee_app_debug_println("now using the %dMPHY\r\n",
                                  conn_evt->phy);
    }
    break;
    case sl_bt_evt_connection_closed_id: {
      sl_bt_evt_connection_closed_t *conn_evt =
        (sl_bt_evt_connection_closed_t*) &(evt->data);

      sl_zigbee_app_debug_println(
        "BLE connection closed, handle=0x%02x, reason=0x%02x",
        conn_evt->connection, conn_evt->reason);
    }
    break;

    case sl_bt_evt_scanner_legacy_advertisement_report_id: {
      sl_zigbee_app_debug_print("Scan response, address type=0x%02x",
                                evt->data.evt_scanner_legacy_advertisement_report.address_type);
      zb_ble_dmp_print_ble_address(evt->data.evt_scanner_legacy_advertisement_report.address.addr);
      sl_zigbee_app_debug_println("");
    }
    break;

    case sl_bt_evt_connection_parameters_id: {
      sl_bt_evt_connection_parameters_t* param_evt =
        (sl_bt_evt_connection_parameters_t*) &(evt->data);
      sl_zigbee_app_debug_println(
        "BLE connection parameters are updated, handle=0x%02x, interval=0x%02x, latency=0x%02x,
timeout=0x%02x, security=0x%02x, txsize=0x%02x",
        param_evt->connection,
        param_evt->interval,
        param_evt->latency,
        param_evt->timeout,
        param_evt->security_mode,
        param_evt->txsize);
    }
    break;

    case sl_bt_evt_gatt_service_id: {
      sl_bt_evt_gatt_service_t* service_evt =
        (sl_bt_evt_gatt_service_t*) &(evt->data);
      uint8_t i;
      sl_zigbee_app_debug_println(
        "GATT service, conn_handle=0x%02x, service_handle=0x%04x",
        service_evt->connection, service_evt->service);
      sl_zigbee_app_debug_print("UUID=[");
      for (i = 0; i < service_evt->uuid.len; i++) {
        sl_zigbee_app_debug_print("0x%04x ", service_evt->uuid.data[i]);
      }
      sl_zigbee_app_debug_println("]");
    }
    break;

    default:
      break;
  }
}
#endif //SL_CATALOG_BLUETOOTH_PRESENT
```


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 LABS

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

www.silabs.com