AN1259: Using the v3.x Silicon Labs Bluetooth® Stack in Network Co-Processor Mode

This document is an essential reference for anyone developing a system for the Silicon Labs Wireless Gecko products using the Silicon Labs v3.x Bluetooth Stack in Network Co-Processor (NCP) mode. The document covers the C language application development flow, walks through the examples included in the stack, and shows how to customize them.

KEY POINTS

- Introduces the available tools for NCP system development.
- Walks through the NCP host and target examples.

1. Introduction

The Silicon Labs Bluetooth SDK allows you to develop System-On-Chip (SoC) firmware in C on a single microcontroller. The SDK also supports the Network Co-Processor (NCP) system model.

This document gives you a guide on how to get started with software development of an NCP system. It describes the development tools and example projects, then highlights the most important steps you need to follow when writing your own application.

1.1 SoC vs NCP System Models

On an SoC system, the Application code, the Bluetooth Host, and Controller code run on the same Wireless MCU.

On an NCP system, the Application runs on a Host MCU, and the Host and Controller code run on a Target MCU. The Host and Target MCUs communicate on a serial interface. The communication between the Host and Target is defined in the Silicon Labs Proprietary Protocol called BGAPI. The physical interface is UART. BGLib v3.x is an ANSI C reference implementation of the BGAPI protocol, which can be used in the NCP Host Application.

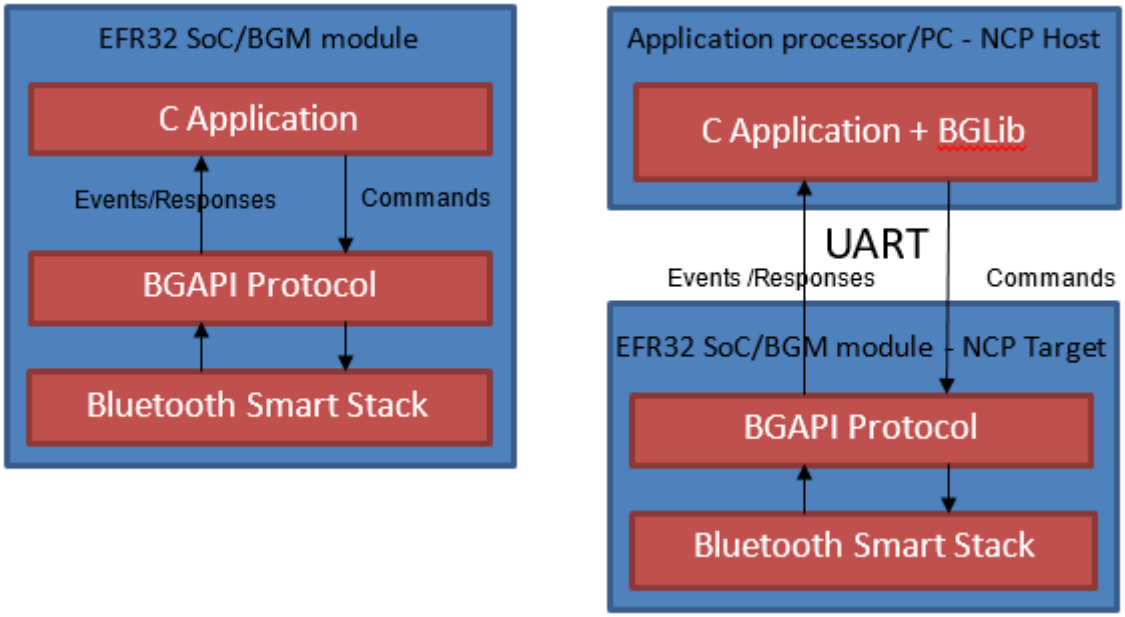


Figure 1.1. SoC vs NCP System Models

2. NCP Target Development

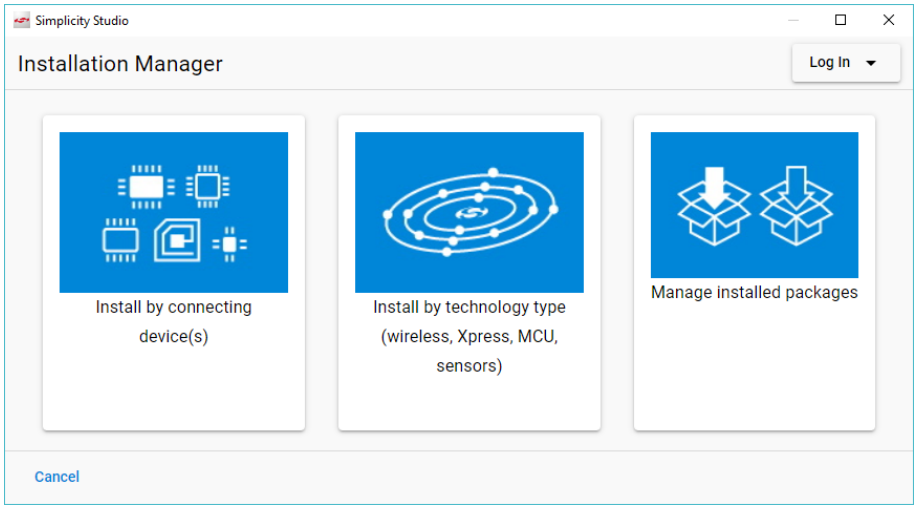
This chapter describes the available tools for compiling and flashing the NCP target firmware.

Before proceeding with compiling and flashing C-based firmware, you need to install Simplicity Studio 5 (SSv5). You can download it from the Silicon Labs website: http://www.silabs.com/simplicity.

Before installing Simplicity Studio, connect the WSTK and your PC with a USB cable. The white switch located on the left side of the WSTK must be in the AEM position. See *QSG169: Bluetooth® SDK v3.x Quick Start Guide* for details on installing Studio and the Bluetooth SDK.

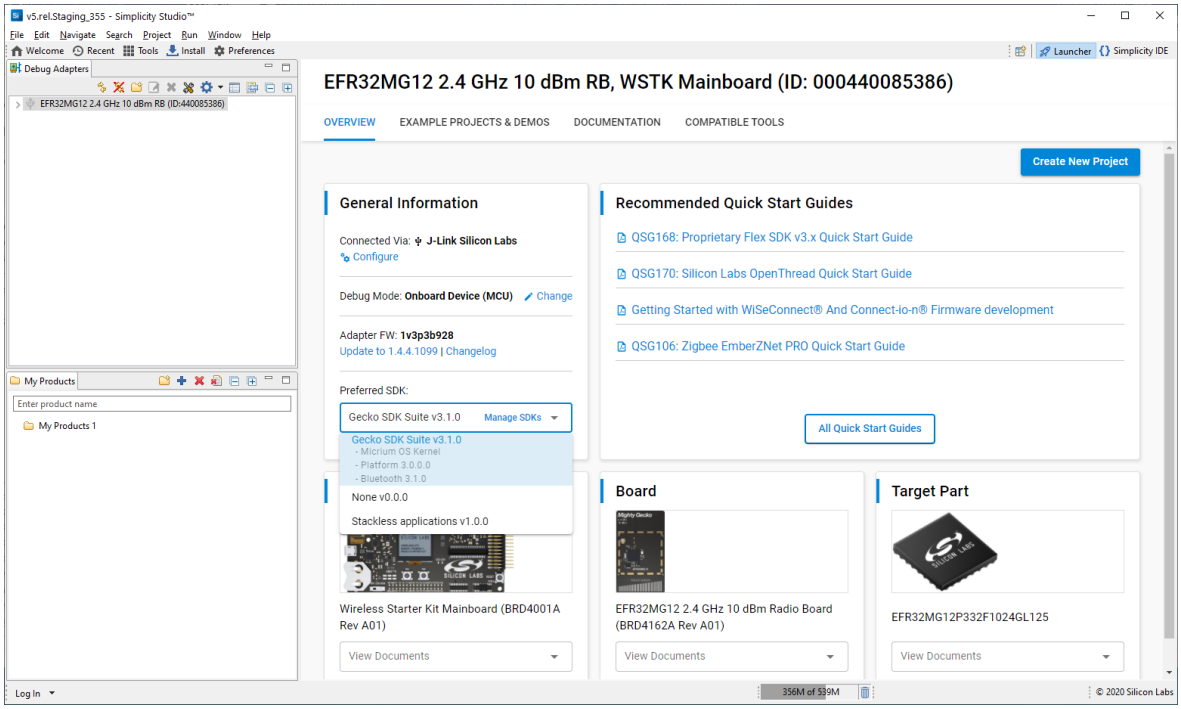
Note: *AN1042: Using the v2.x Silicon Labs Bluetooth® Stack in Network Co-Processor Mode* describes in detail how the NCP is implemented in the Gecko SDK v2. This application note explains extensively the code and tools on both the target and host side.

If you have already installed SSv5, you can download the Bluetooth SDK through the Simplicity Studio Package Manager. To open it, click the **Install** control on the SSv5 toolbar, and then click **Manage Installed Packages**. Go to the Stacks tab, and click [**Install**] next to the Bluetooth SDK.



To develop in C, you not only need Simplicity Studio 5 but also a supported compiler. *UG434: Silicon Labs Bluetooth® C Application Developers Guide for SDK v3.x* lists the supported compilers.

The NCP target firmware comes with the Bluetooth SDK. It is available in a precompiled binary format and as a project file you can build. The following procedures describe how to install the precompiled binary image and how to build and install the example project. Note that Simplicity Studio only shows the relevant examples for the preferred SDK, so you have to select **Gecko SDK Suite: Bluetooth** first, as shown in the following figure. (Note: Your SDK version may be different from the one shown in the figure.)

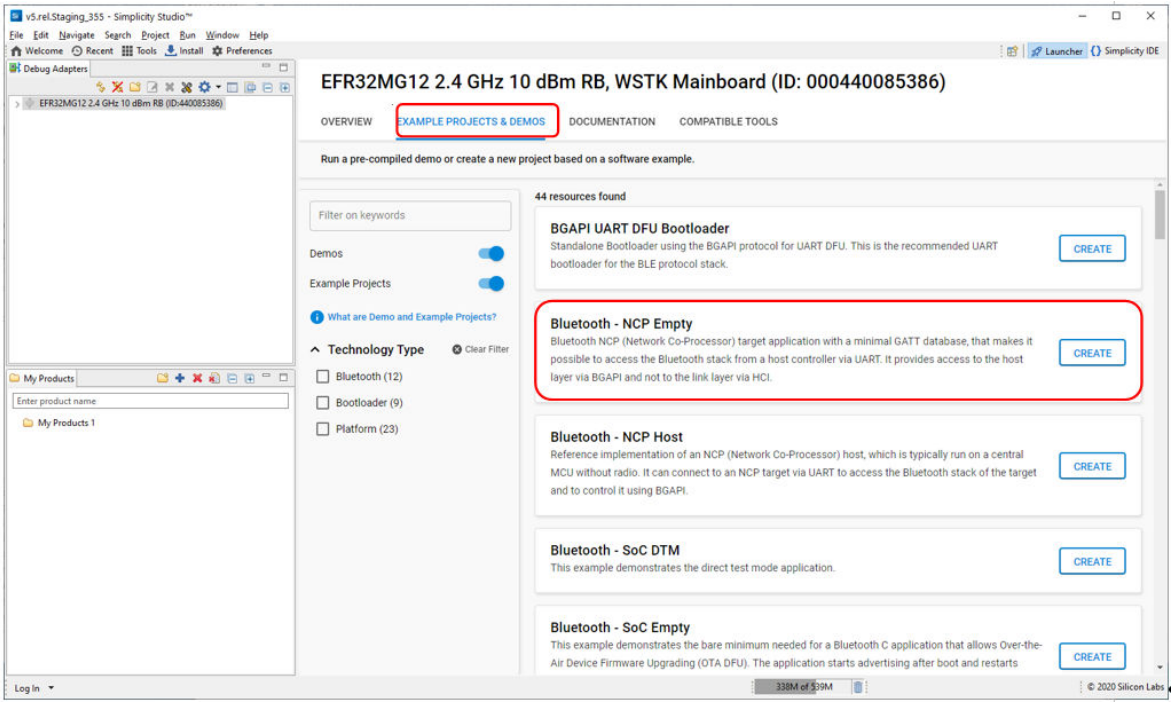


The following procedure describes how to build and load the example code. This procedure assumes you have already loaded a Gecko Bootloader in one of the following ways:

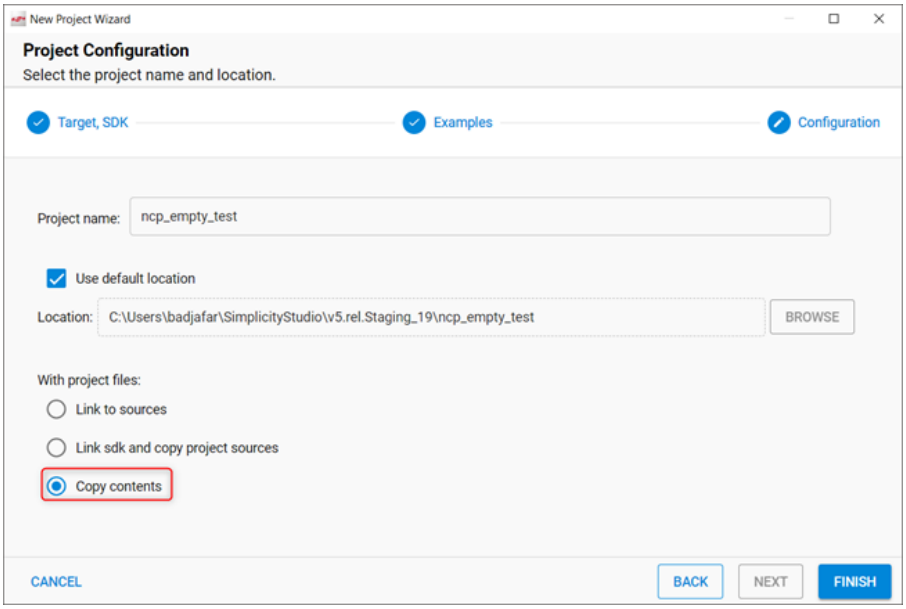
- Loading the Gecko Bootloader precompiled binary from the list of Demos. For an NCP application, you should load the BGAPI UART DFU bootloader.
- Building and loading your own Gecko Bootloader as described in chapter 6 of *UG266: Silicon Labs Gecko Bootloader User's Guide*.

1. Click **Example Projects & Demos**, select **Bluetooth - NCP Empty**, and click **[Create]**.

   Note: Bluetooth – NCP Empty has a minimal GATT database and is useful to test Bluetooth features without having to use the GATT database, or if you want to build a static GATT database on the target side, and you need a starting point. For any other cases, use **Bluetooth – NCP**, in which the host software builds the GATT database dynamically. Host software examples in the Bluetooth SDK build their GATT database dynamically, by default.



2. Name your project and make sure to select **Copy contents**. Click **[Finish]**.



3. Now your project is ready to build and flash. Click **Debug** (bug icon) in the top left menu to do it in one step. Once the flashing is completed press F8 to start the firmware.

   Note: If you get an error when you click **Debug**, click the project *.isc* file in the Project Explorer view. It may not be fully selected.

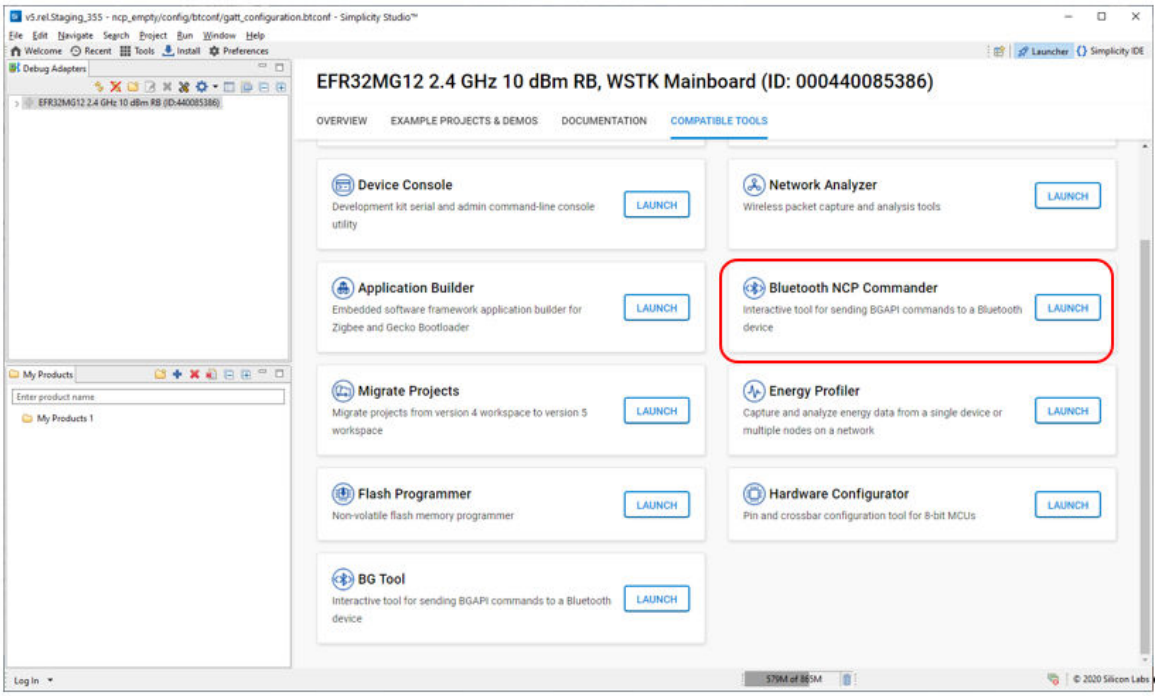
3. NCP Host Development

This chapter introduces the Bluetooth NCP Commander tool, which can be used to send BGAPI commands from a graphical user interface. It then walks through the process of building the PC Host examples provided in the Bluetooth SDK. And finally, it describes using Python for host side development.

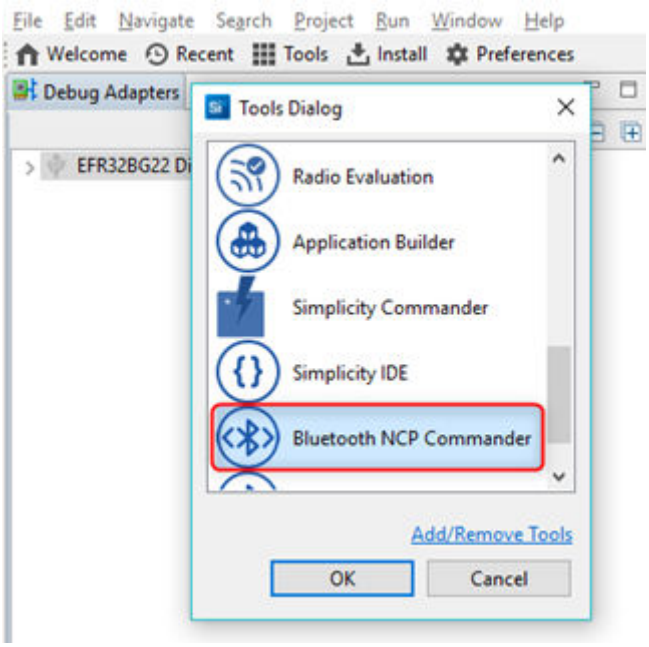
3.1 Bluetooth NCP Commander

Bluetooth NCP Commander is an easy-to-use tool that can be used for testing different stack features, by sending BGAPI commands to the target device. The tool has two versions: a version built into Simplicity Studio, which makes it easy to connect to your development kit and start testing, and a standalone version to test a board in an environment where Simplicity Studio cannot be installed, or if you want to test a custom board that can be accessed on UART interface, but not through a Simplicity Studio supported debug adapter using VCOM.

1. To open the built-in Bluetooth NCP Commander, select the target board in the **Debug Adapters** view, and check that the preferred SDK is set to **Gecko SDK Suite: Bluetooth**. Select the **Compatible Tools** tab, and click **[Launch]** next to Bluetooth NCP Commander.

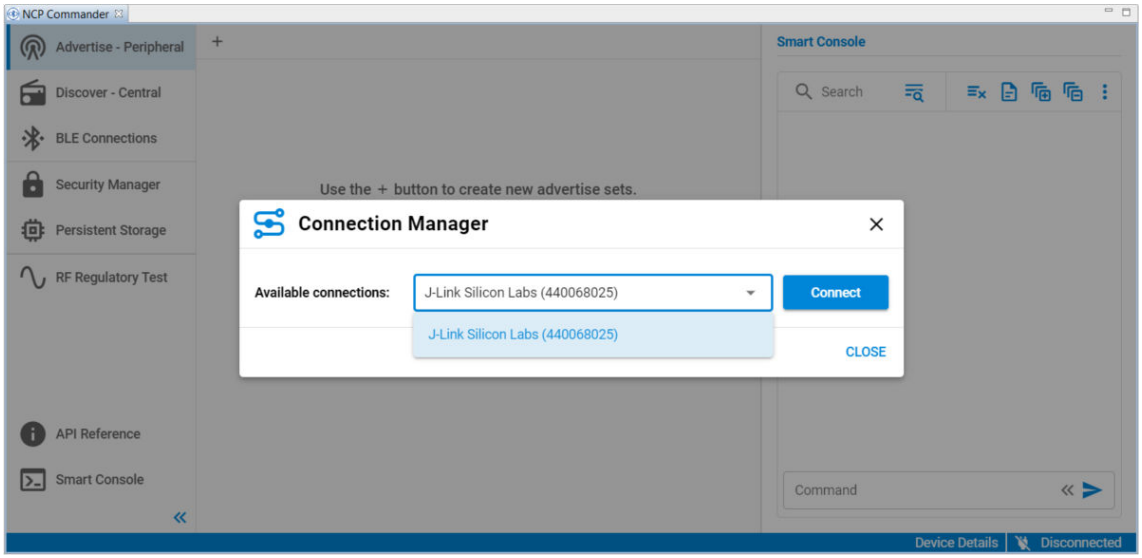


Alternatively, you can open Bluetooth NCP Commander from the **Tools** menu.

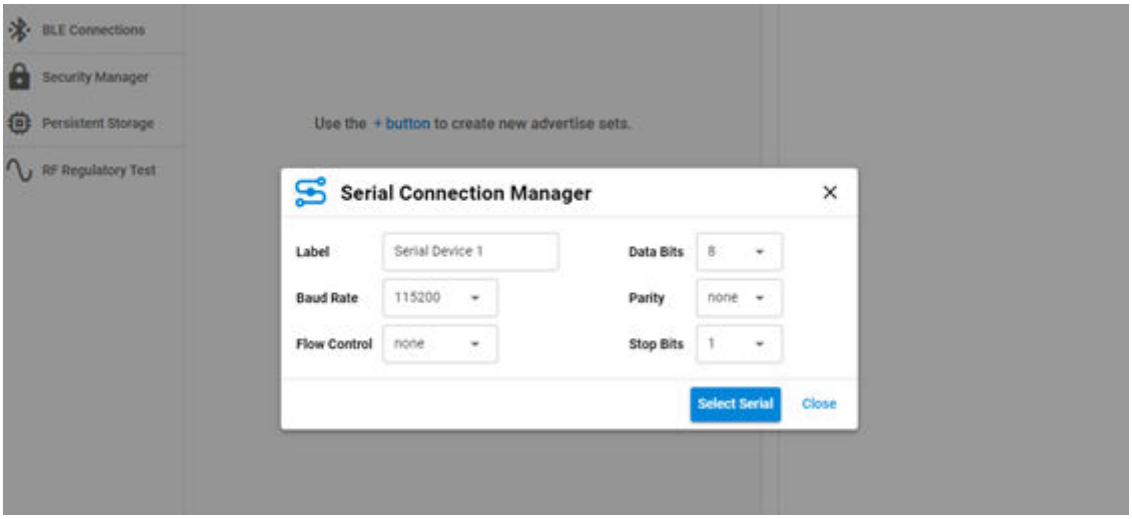


2. To open the standalone tool, navigate to C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\ncp_commander, and start NcpCommander.exe.

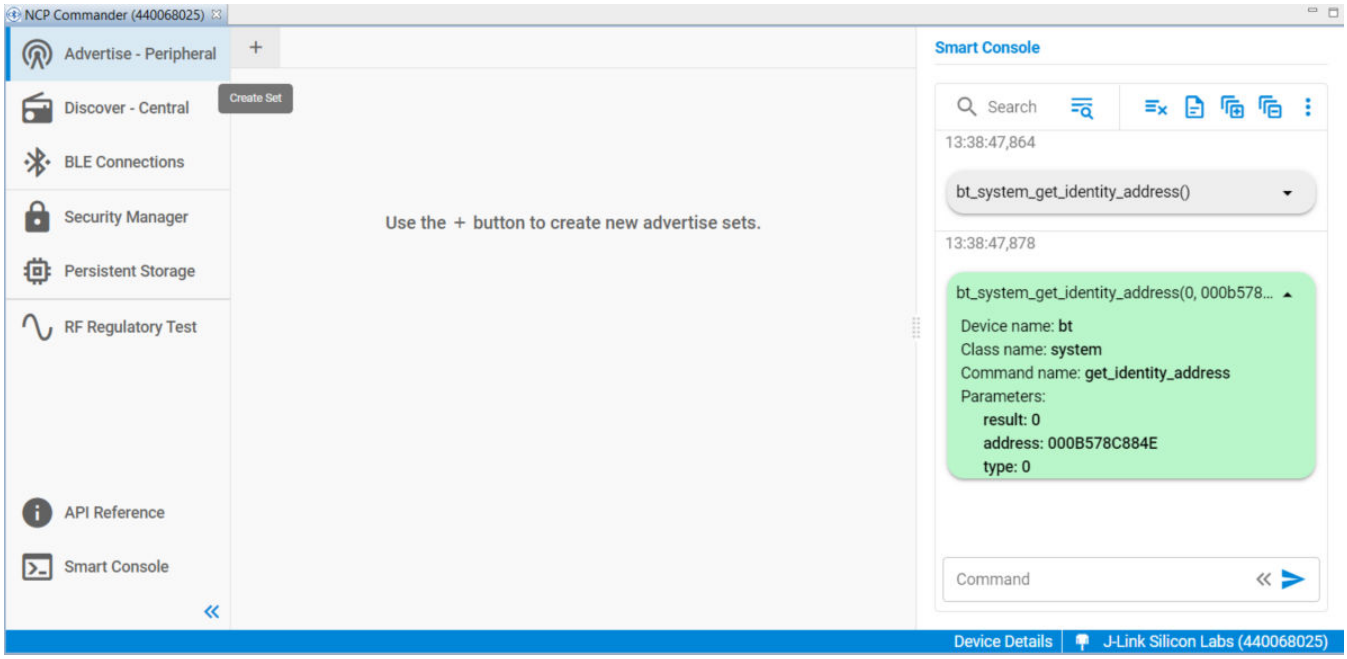
3. If you use the built-in version, select the target device, and click **[Connect]**.



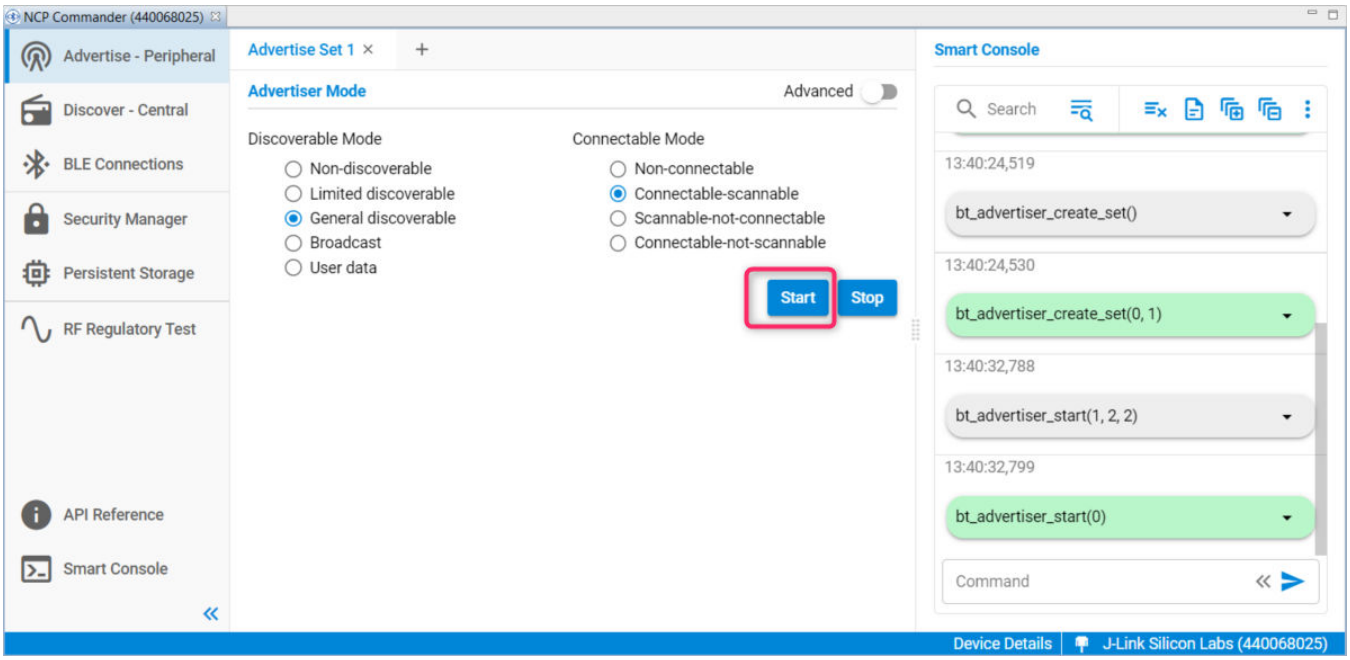
4. In the standalone tool, provide the UART interface settings, and then select the COM port on which the device can be accessed.



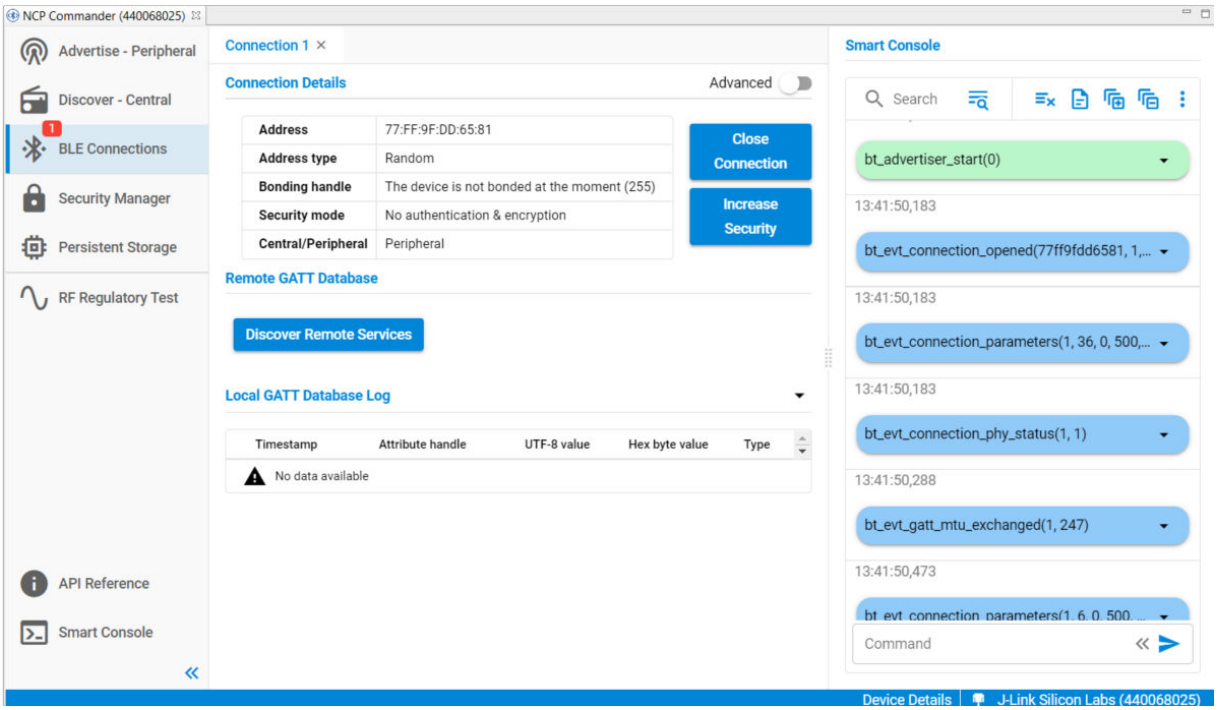
5. After the UART connection to the WSTK is established, an Interactive view opens, which you can use to issue BGAPI commands. Check the log for the NCP target response and status messages. To start advertising, click the "+" button under the "Advertise – Peripheral" view to create an advertiser set:



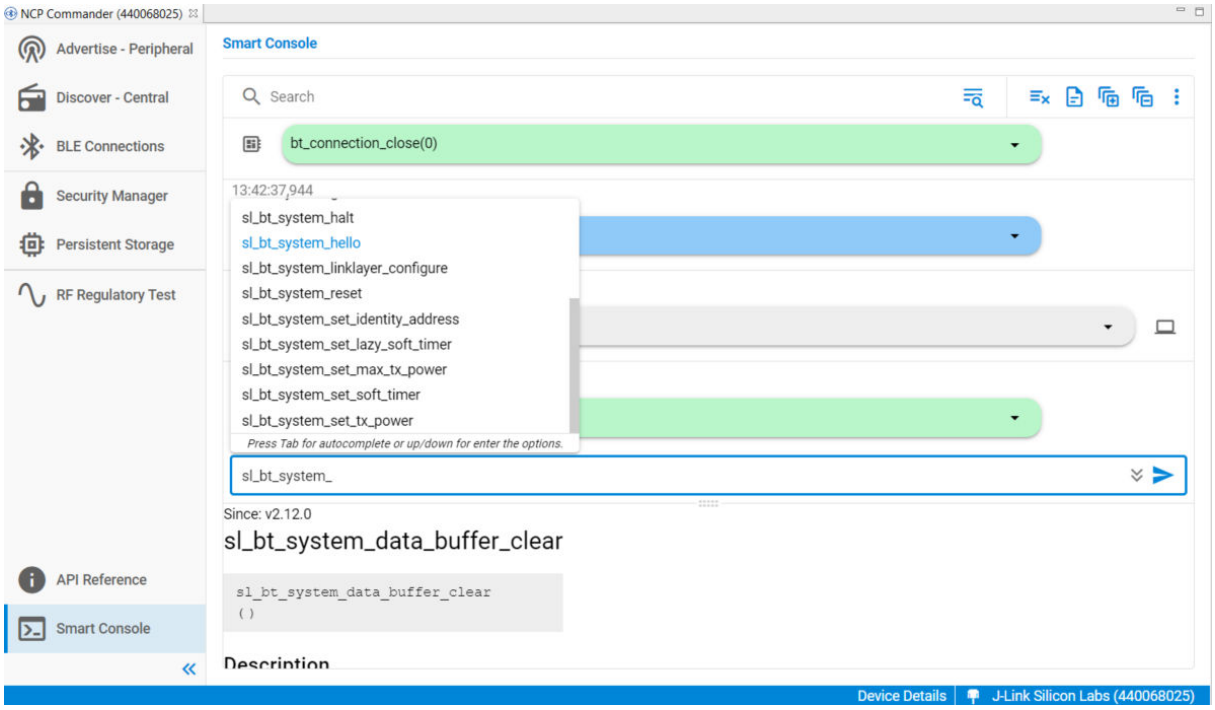
6. Select the desired advertising mode, and click **[Start]**.

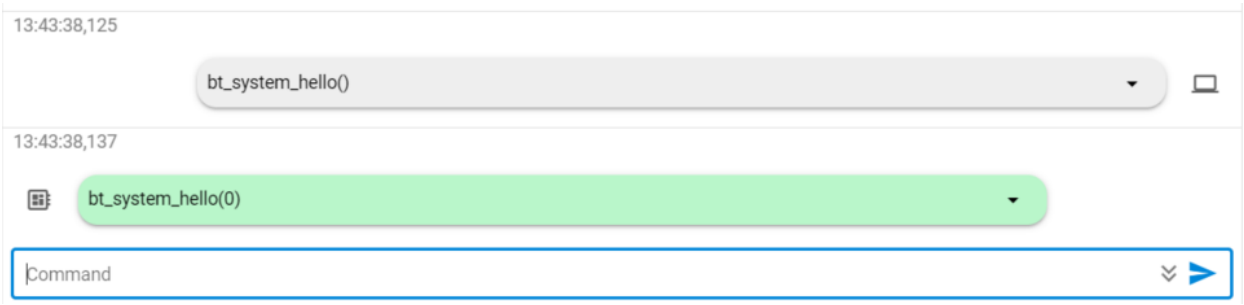


7. When advertising, the NCP target example accepts Bluetooth connections. If you connect to your WSTK or with another central device (for example with your phone), you can see the events and commands on the log.



8. You can also issue commands manually. For example, you can issue the 'system hello' command at any time to verify that communication between the host and the device is working. The Smart Console provides auto completion and documentation for the possible commands. To open/close the documentation, click the arrows at the right side of the input field.





3.2 Building the NCP Host Example on Windows

The Silicon Labs v3.x Bluetooth SDK contains a generic NCP Host example project for the PC. This example can be compiled on Windows or any POSIX OS. This section goes through the build process on Windows.

Note: The host example projects in the SDK use the dynamic GATT database feature. Therefore, they are to be used together with the Bluetooth – NCP target application not with Bluetooth – NCP Empty.

1. To build the examples properly, you need the MSYS2 development toolchain installed on your PC. Download MSYS2 at https://www.msys2.org/.
2. After MSYS2 is installed, update the package database as described at https://www.msys2.org/.
3. Start MSYS2 bash and install mingw-64 with the following command:

```
pacman -S make mingw-w64-x86_64-gcc
```

4. Close MSYS2 and start MSYS2 MinGW 64-bit.

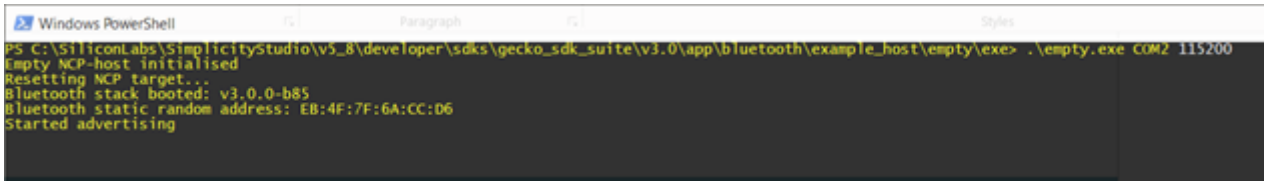
5. Change to the NCP Host example folder, where <version> varies by SDK version:

```
cd c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.x\app\bluetooth\example_host\empty\
```

6. Build the project with the command: `make`.
7. The build output is created in a new *exe* folder. Go to this folder with `cd exe`, and then run the *empty.exe*. The COM port and the baud rate are passed as command line parameters. The COM port should be the same as the one used by the JLink CDC UART Port, as shown in section 3. NCP Host Development. To see how to pass the different parameters, first run the exe with the -h (help) switch.

```
 .\empty.exe -h
```

8. Once the UART connection with the WSTK is established, you should see the following:



9. Now you can connect to the WSTK over Bluetooth.

3.3 Using Python for Host Side Development

You can also implement a host application using Python. A Python package is available at https://pypi.org/project/pybgapi/. This package parses the API description file of the Bluetooth SDK and makes it possible to issue BGAPI commands and get BGAPI events in the Python environment. See the referred website for further documentation.

4. Example Project Walkthrough

This chapter describes the structure of the example NCP Host and Target projects, and highlights the parts that can be important if you create your own project.

4.1 NCP Target

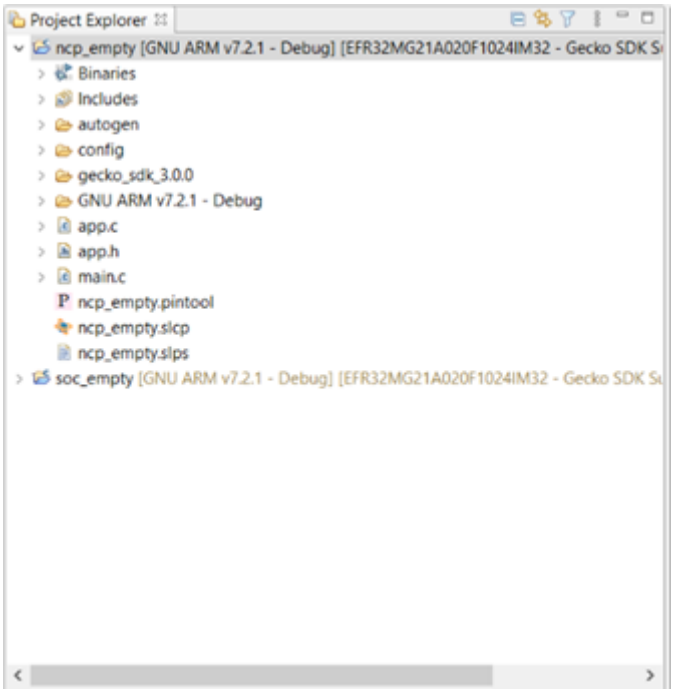
This section focuses on the NCP-specific part of the ncp / ncp-empty SSv5 projects. You can find a general project description in *UG434: Silicon Labs Bluetooth® C Application Developers Guide for SDK v3.x*. There are two sample projects for Target development: “Bluetooth - NCP” and “Bluetooth – NCP Empty”.

The “Bluetooth – NCP Empty” example contains a minimal GATT database. Use this with NCP Commander, or for any other use cases when building a database with the APIs is not a viable option.

The “Bluetooth - NCP” example does not contain a GATT database. The dynamic GATT API can be used for building it. This is recommended for most of the use cases because the target code does not need to be modified and synchronized with the Host code when the GATT database is updated.

4.1.1 Project File Structure

A common directory and file structure are used across all examples in the Bluetooth SDK v3.x. The following figure shows this layout.

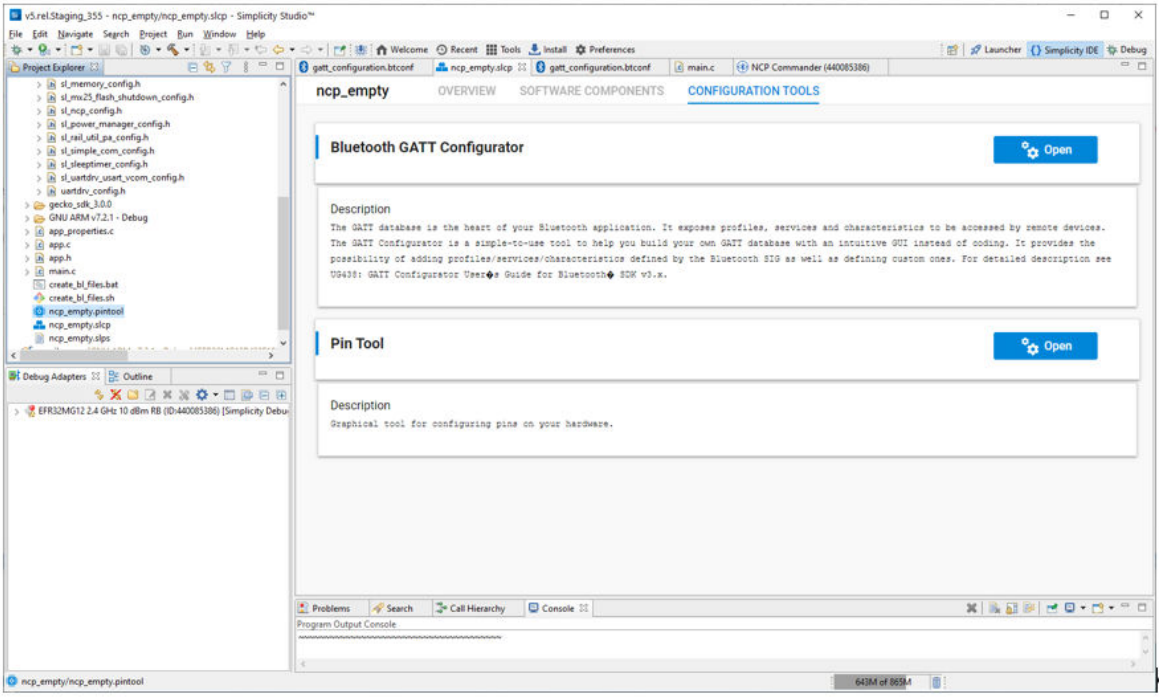


These files and directories are present in the root directory of the project:

- *main.c* and *app.c/h* – the C application code
- *ncp_empty.pintool* – the hardware configuration file and user interface
- *ncp_empty.slcp* – the component configuration and user interface
- *ncp_empty.slps* – the project properties XML file
- *GNU ARM v<X.Y.Z>* – the build directory
- *gecko.sdk_3.<X.Y>* – the Bluetooth SDK source code
- *config* – the C configuration files of the hardware and Bluetooth stack. This directory contains the output files of the Pintool and Component Manager. For an “NCP – empty” example, it also contains the Bluetooth GATT configuration *tool/user* interface file.
- autogen – the C configuration code of the application. This directory typically contains the stack definition and initialization C files, as well as the generated GATT database C declaration files (*gatt_db.c/h*).

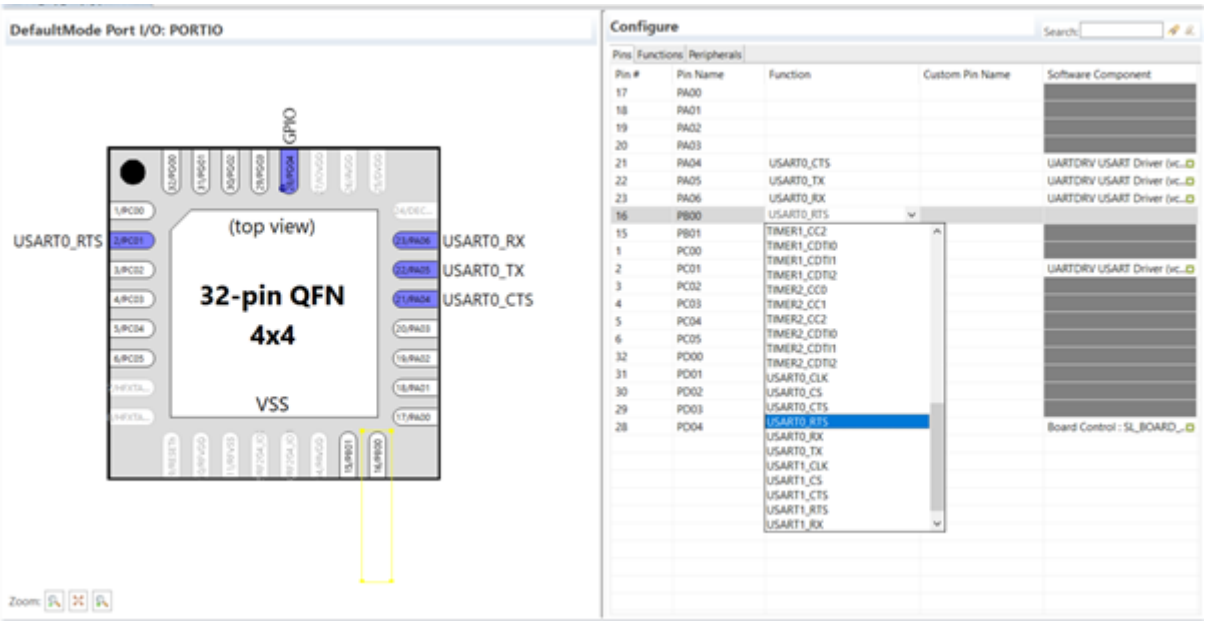
4.1.2 Pin Tool

1. Open the pin configuration tool (Pin Tool) on the project Configuration Tools tab.

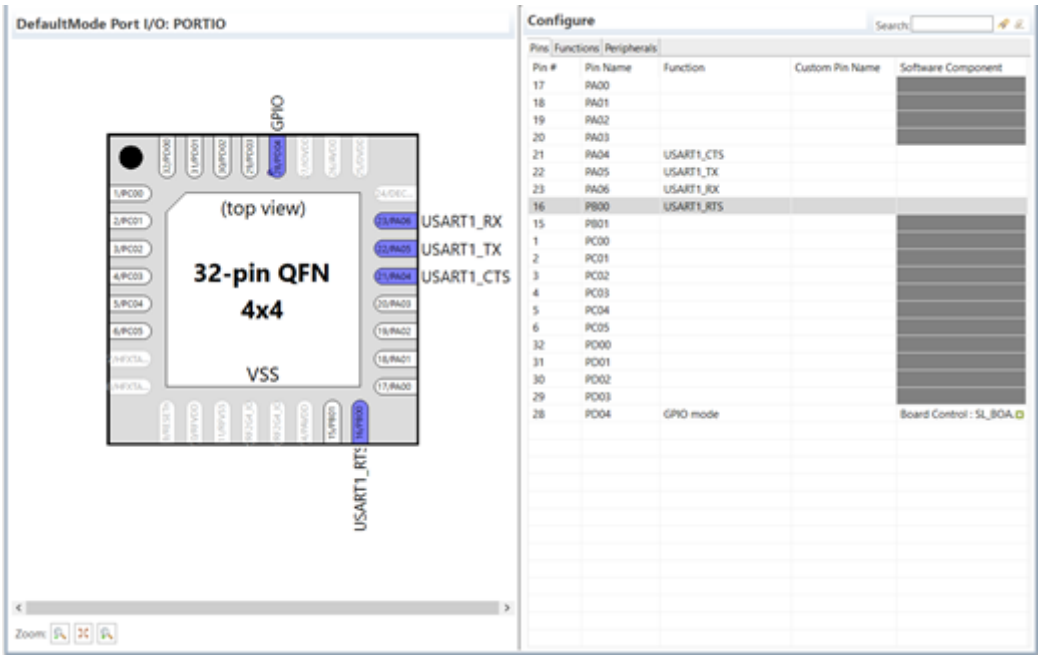


You can also double-click the <projectname>.pintool file in the Project Explorer view, shown highlighted in the figure above.

2. Use this tool to modify the pin configuration of the device, for example, you can reassign the pins used for USART communication to the appropriate layout for a custom board design. You do this by selecting the desired pin in the list and then selecting its functionality from the drop-down list.



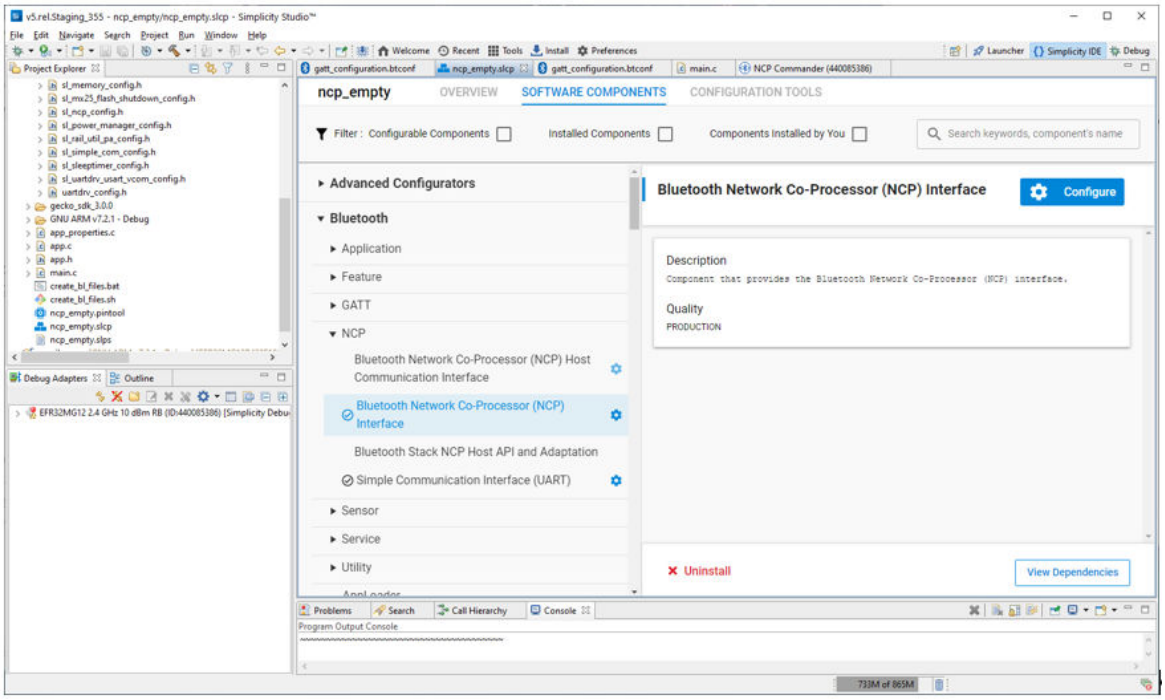
3. After clicking the selected item, the layout is updated. After saving the file, the configuration source codes are automatically generated.



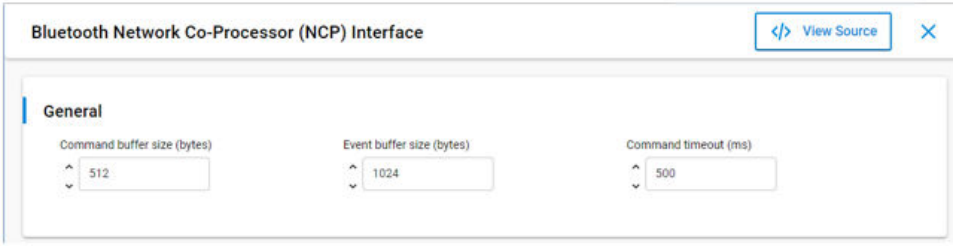
4.1.3 Project Configurator / Component Editor

You can install or uninstall components using the Project Configurator's Software Components tab. You can also configure installed components using the Component Editor. The following figures show how to change the NCP interface buffer sizes.

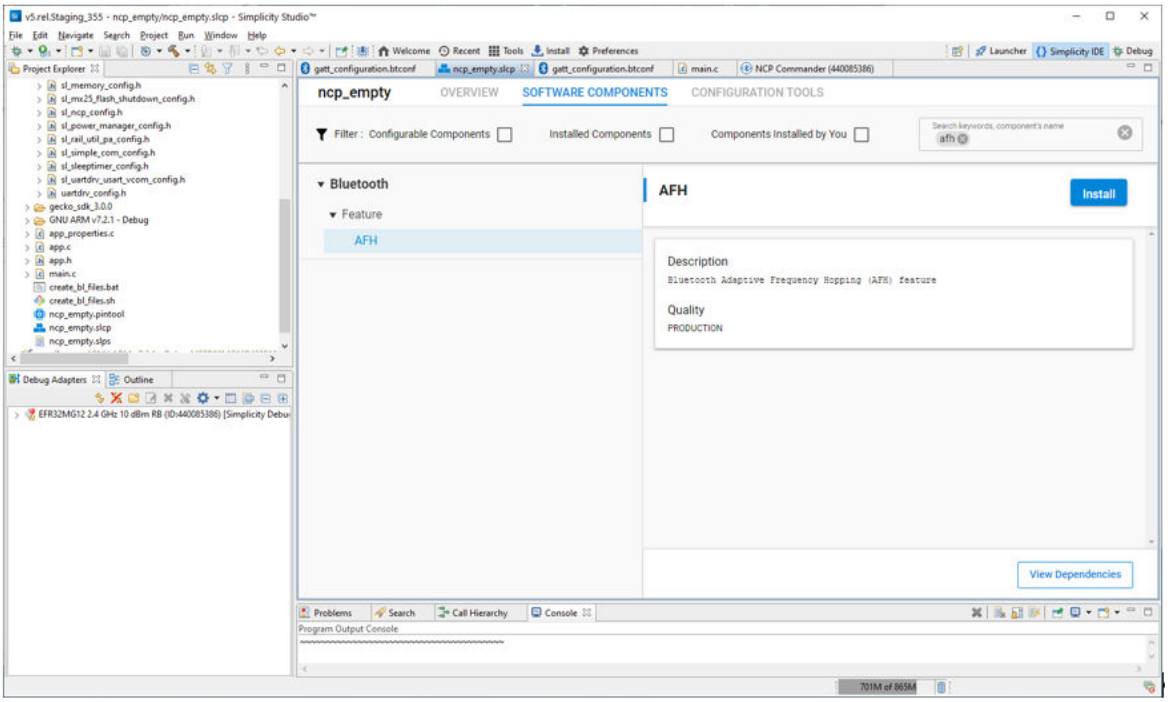
1. Select the component from the list and click **[Configure]**.



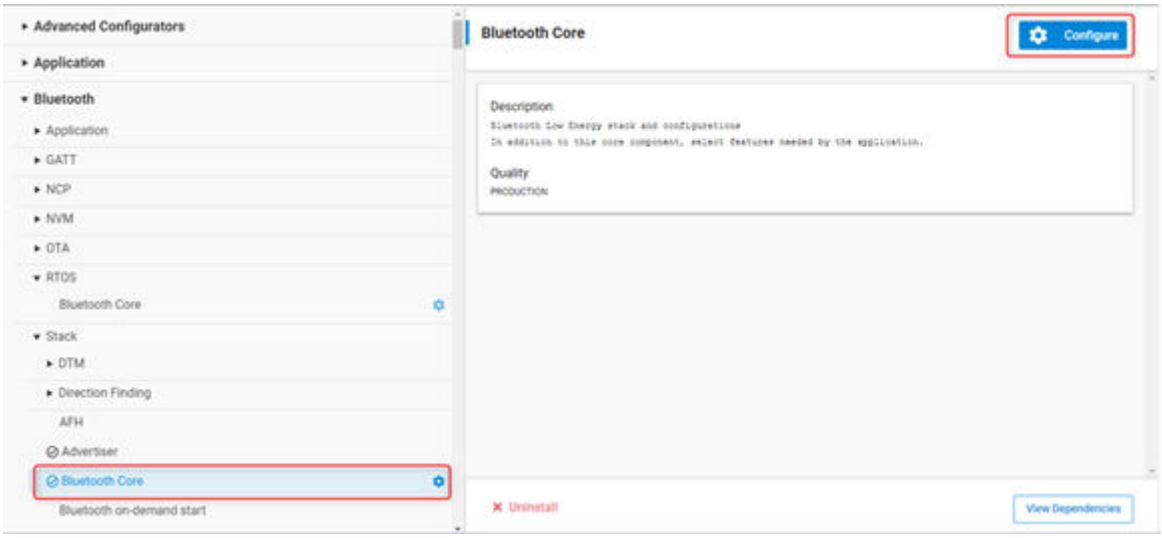
2. The Component Editor opens in a new tab with the possible configuration options. You can view the corresponding source code by clicking **[Open Source]**.



3. Apart from the application-specific NCP options, you can use the Project Configurator to configure the Bluetooth stack features that will be included in your project. Some advanced features are excluded from the stack by default, to save flash and memory. You can add the needed features, for example, the Adaptive Frequency Hopping (AFH) component, by clicking **[Install]**.



4. In many cases, you also need to change the default Bluetooth Core configuration, for example to enable more than four connections. To do so, browse for the Bluetooth Core component, and click **[Configure]**.



4.1.4 Main Walkthrough

This is a code snippet that corresponds to the `main` function. Because the Bluetooth stack and subsequent hardware are considered to be components, they are separated from the application processing that is entirely managed in *app.c/h*.

```
int main(void)
{
  // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
  // Note that if the kernel is present, processing task(s) will be created by
  // this call.
  sl_system_init();

  // Initialize the application. For example, create periodic timer(s) or
  // task(s) if the kernel is present.
  app_init();

#if defined(SL_CATALOG_KERNEL_PRESENT)
  // Start the kernel. Task(s) created in app_init() will start running.
  sl_system_kernel_start();
#else // SL_CATALOG_KERNEL_PRESENT
  while (1) {
    // Do not remove this call: Silicon Labs components process action routine
    // must be called from the super loop.
    sl_system_process_action();

    // Application process.
    app_process_action();

#if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
    // Let the CPU go to sleep if the system allows it.
    sl_power_manager_sleep();
#endif
  }
#endif // SL_CATALOG_KERNEL_PRESENT
   }
```

Once the USART and Bluetooth stack are initialized, the main loop continuously calls the component as well as the application state machine. The corresponding functions are `sl_system_process_action()` and `app_process_action()` respectively.

The `sl_system_process_action()` handles Silicon Labs tasks and routines. It must *not be removed* from the loop.

The default USART settings are mentioned in the Host example section. Make sure that the target and the host use the same configuration. The configuration can be adapted with the help of the Pin Tool and the Project Configurator.

4.1.5 Application Callback and Actions

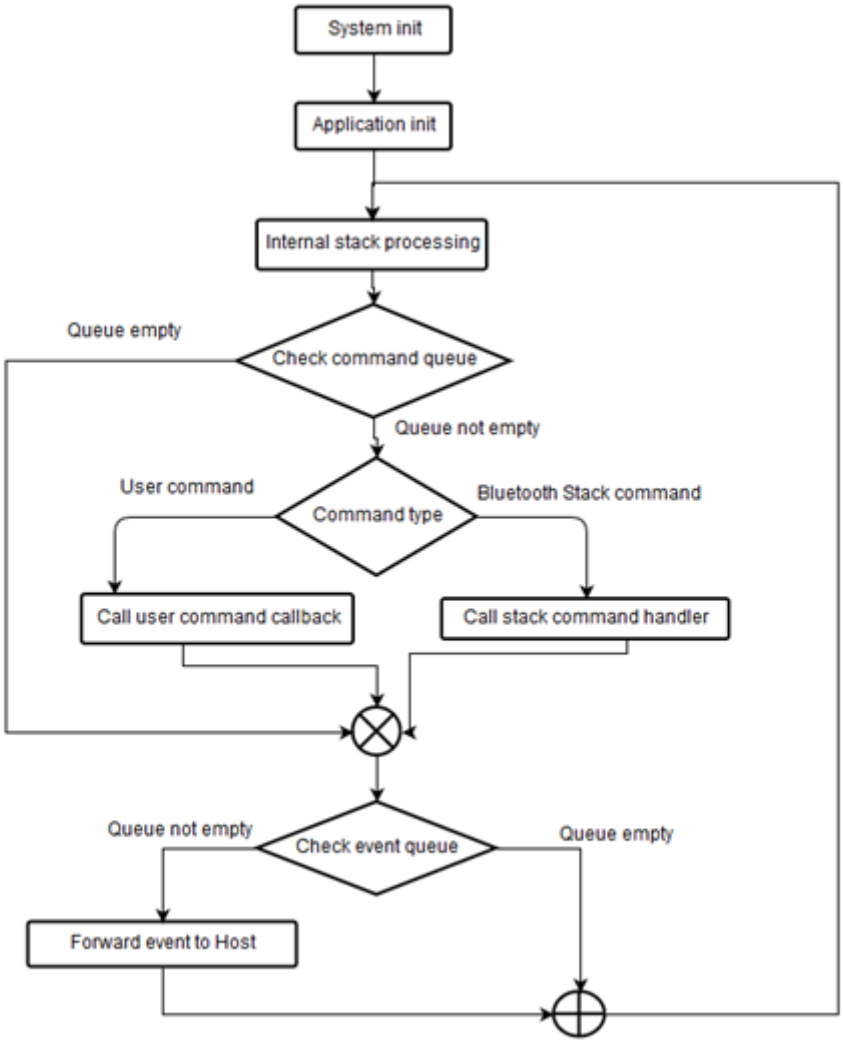
Use the `app_init()` function to call application-related initializations.

Use the `app_process_action()` function to call application-specific tasks and routines.

```
/**************************************************************************//**
 * Application initialisation.
 *****************************************************************************/
void app_init(void)
{
  ncp_init();

  ////////////////////////////////////////
  // Add your application init code here! //
  ////////////////////////////////////////
}

/**************************************************************************//**
 * Application process actions.
 *****************************************************************************/
void app_process_action(void)
{
  ncp_process_action();

  ////////////////////////////////////////
  // Add your application tick code here! //
  ////////////////////////////////////////
}
```

4.1.6 NCP Code Walkthrough

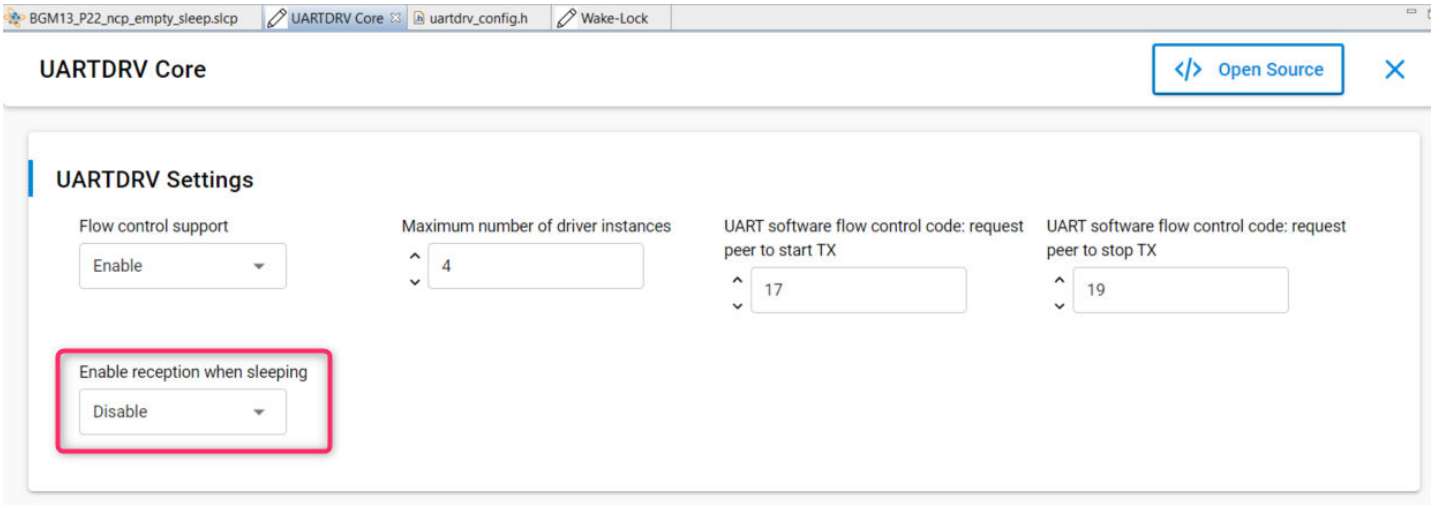
The USART communication handling is implemented in *ncp_usart.c*. Receiving any command from the Host generates an interrupt, and it will queue the received data in the command queue. Similarly, when a stack generates an event, it will be put into an event queue, which will be forwarded to the Host. These two queues will be processed in *ncp.c*, as described on the following figure.



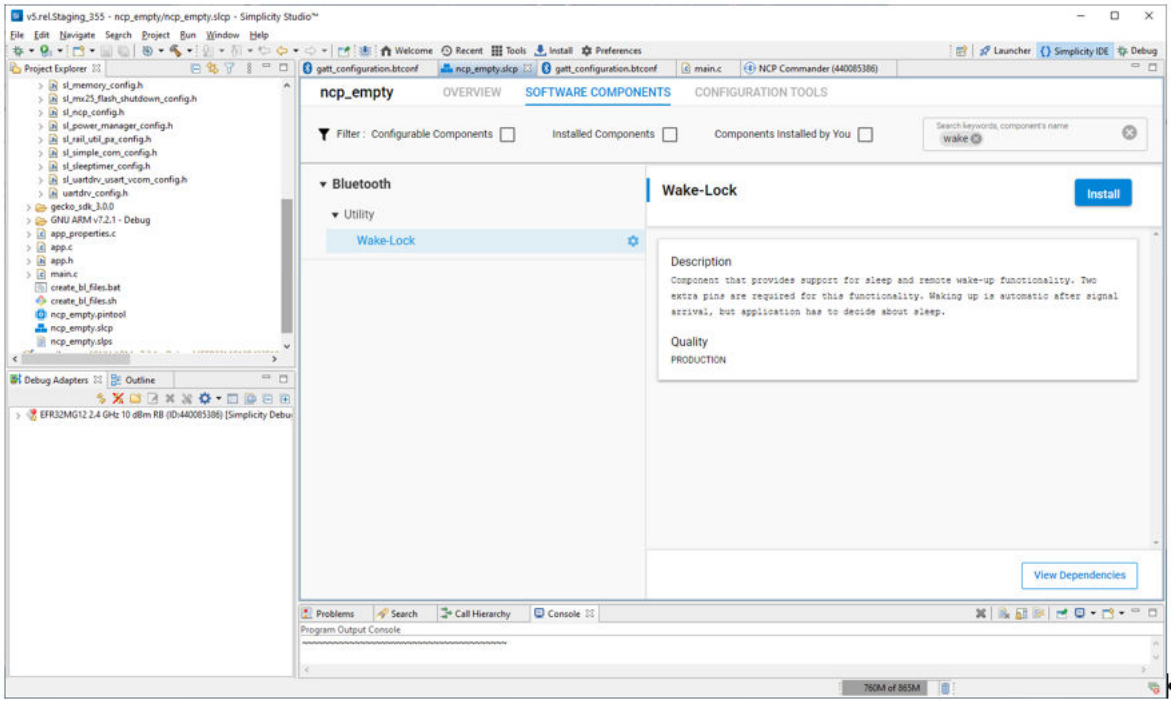
4.1.7 Sleep Modes

The NCP Empty example project does not enable deep sleep mode (EM2) by default, because UART needs EM1 or EM0 to be able to receive commands at any time. Deep sleep mode can be enabled, but in this case, it is essential to configure a wakeup pin so that the NCP Host can wake up the target before sending any BGAPI commands to it. Any available GPIO pin can be configured as a wakeup pin and the polarity is configurable. The following example shows how to configure pin PF6 as the wakeup pin using active-high polarity.

To enable deep sleep mode, the UARTDRV Core component's **Enable reception when sleeping** parameter must be disabled. Otherwise the UART driver will prevent the device from going into EM2 (deep sleep) and it will stay in EM1 (sleep):

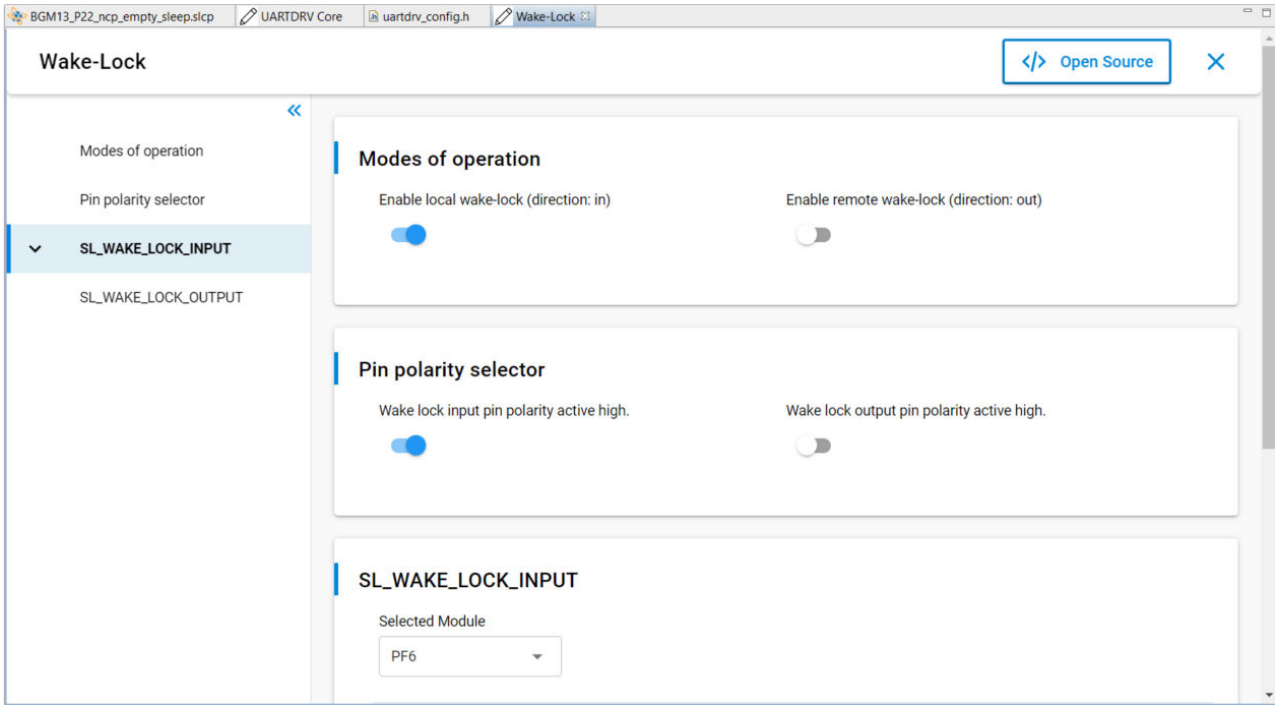


To define a wakeup pin, the Bluetooth > Utility > Wake Lock component must be added to the project:



Configure the Wake-Lock component as follows:

- Enable the wake-lock (direction in) functionality
- Set the polarity (active high in this case)
- Assign the GPIO pin (PF6 in this example)



When the Host sets the wakeup pin to the configured active value, the NCP device will wake up from deep sleep and send out the event `sl_bt_evt_system_awake` to indicate to the host that it has woken up. The host must wait for this event before sending any BGAPI commands, otherwise they might be partially or completely missed.

The remote wake-lock (direction: out) functionality can be used to wake up the host before the NCP target sends out an event. This way the host is also able to go into sleep mode, and it will be notified when it should wake up.

4.2 PC Host

The PC host application project that comes with the SDK is written in C. The host-side source files for this project are found in folders, where <version> varies by Gecko SDK Suite version:

c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\example_host\empty\

c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\example_host\common\

The projects comprise only a few source and header files. Note, however, that many other files are referenced from the SDK in the makefile.

4.2.1 BGAPI Support Files

While the files in the previous section contain all of the application logic, the actual BGLib implementation code containing the BGAPI parser and packet generation functions is found elsewhere, in other subfolders. <version> will vary by SDK version.

- c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\inc\sl_bt_ncp_host.h
- c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\src\sl_bt_ncp_host.c
- c:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\src\sl_bt_ncp_host_api.c

The SDK's specific arrangement of files is one possible way the BGAPI protocol can be used, but it is also possible to create your own library code that implements the protocol correctly with a different code architecture. The only requirement here is that the chosen implementation must be able to create BGAPI command packets correctly and send them to the module over UART. Similarly, it must be able to receive BGAPI response and event packets over UART and process them into whatever function calls are needed to trigger the desired application behavior.

The header files contain primarily #define'd compiler macros and named constants that correspond to all of the various API methods and enumerations you may need to use. The *sl_bt_ncp_host.h* file also contains function declarations for the basic packet reception, processing, and transmission functions.

The *sl_bt_ncp_host.c* file contains the implementation of the packet management functions. All functions defined here use only ANSI C code, to help ensure maximum cross-compatibility on different platforms.

Note: With structure packing, the SDK's BGLib implementation makes heavy use of direct mapping of packet payload structures onto contiguous blocks of memory, to avoid additional parsing and RAM usage. This is accomplished with the PACKSTRUCT macro used extensively in the BGLib header files. It is important to ensure than any ported version of BGLib also correctly packs structures together (no padding on multi-byte struct member variables) in order to achieve the correct operation.

With byte order, the BGAPI protocol uses little-endian byte ordering for all multi-byte integer values, which means directly-mapped structures will only work if the host platform also uses little-endian byte ordering. This covers most common platforms today, but some big-endian platforms exist and are actively used today (Motorola 6800, 68k, and so on).

4.2.2 Host Application Logic

1. Initialize BGLIB.

```
SL_BT_API_INITIALIZE_NONBLOCK(uart_tx_wrapper, uartRx, uartRxPeek);
```

2. Initialize UART.

```
if (serial_port_init(argc, argv, 100) < 0) {
  app_log("Non-blocking serial port init failure\n");
  exit(EXIT_FAILURE);
}
// Flush std output
fflush(stdout);
```

3. Reset NCP Target to ensure it gets into a defined state. Once the chip successfully boots, the `gecko_evt_system_boot_id` event should be received.

```
Sl_bt_system_reset(0);
```

4. The `sl_bt_step` function will be called from the main loop. It checks for any NCP Target events and forwards them to the handler function.

```
// Poll Bluetooth stack for an event and call event handler
static void sl_bt_step(void)
{
  sl_bt_msg_t evt;
  // Pop (non-blocking) a Bluetooth stack event from event queue.
  sl_status_t status = sl_bt_pop_event(&evt);
  if (status != SL_STATUS_OK) {
    return;
  }
  sl_bt_on_event(&evt);
} }
```

5. Process the incoming NCP target events. The example only handles the `gecko_evt_system_boot_id` and the `gecko_evt_le_connection_closed_id` events.

```
/* Handle events */
switch (SL_BT_MSG_ID(evt->header)) { case sl_bt_evt_system_boot_id:
// Print boot message.
      app_log("Bluetooth stack booted: v%d.%d.%d-b%d\n",
              evt->data.evt_system_boot.major,
              evt->data.evt_system_boot.minor,
              evt->data.evt_system_boot.patch,
              evt->data.evt_system_boot.build);
      sc = sl_bt_system_get_identity_address(&address, &address_type);
      app_assert(sc == SL_STATUS_OK,
                 "[E: 0x%04x] Failed to get Bluetooth address\n",
                 (int)sc);
      app_log("Bluetooth %s address: %02X:%02X:%02X:%02X:%02X:%02X\n",
              address_type ? "static random" : "public device",
              address.addr[5],
              address.addr[4],
              address.addr[3],
              address.addr[2],
              address.addr[1],
              address.addr[0]);

      // Create an advertising set.
      sc = sl_bt_advertiser_create_set(&advertising_set_handle);
      app_assert(sc == SL_STATUS_OK,
                 "[E: 0x%04x] Failed to create advertising set\n",
                 (int)sc);

      // Set advertising interval to 100ms.
      sc = sl_bt_advertiser_set_timing(
advertising_set_handle, // advertising set handle
        160, // min. adv. interval (milliseconds * 1.6)
        160, // max. adv. interval (milliseconds * 1.6)
        0,   // adv. duration
        0);  // max. num. adv. events
      app_assert(sc == SL_STATUS_OK,
                 "[E: 0x%04x] Failed to set advertising timing\n",
                 (int)sc);
      // Start general advertising and enable connections.
      sc = sl_bt_advertiser_start(
        advertising_set_handle,            // advertising set handle
        advertiser_general_discoverable,   // discoverable mode
        advertiser_connectable_scannable); // connectable mode
      app_assert(sc == SL_STATUS_OK,
                 "[E: 0x%04x] Failed to start advertising\n",
                 (int)sc);
      app_log("Started advertising\n");
      break;
}
```

```
case sl_bt_evt_connection_closed_id:
      app_log("Connection closed\n");
      // Check if need to boot to OTA DFU mode.
      if (boot_to_dfu) {
        // Enter to OTA DFU mode.
        sl_bt_system_reset(2);
      } else {
        // Restart advertising after client has disconnected.
        sc = sl_bt_advertiser_start(
          advertising_set_handle,            // advertising set handle
          advertiser_general_discoverable,   // discoverable mode
          advertiser_connectable_scannable); // connectable mode
        app_assert(sc == SL_STATUS_OK,
                   "[E: 0x%04x] Failed to start advertising\n",
                   (int)sc);
        app_log("Started advertising\n");
      }
      break;}
```

5. Custom API Support

This chapter introduces how to implement a custom binary protocol between an NCP target and host using specific features of the BGAPI. The Silicon Labs Bluetooth SDK provides the following commands and events for that purpose:

- `cmd_user_message_to_target`
- `evt_user_message_to_host`

The command and event details are documented in the API reference manual.

The `cmd_user_message_to_target` command can be used by an NCP host to send a message to the target application on a device. To send a custom message with this API command, the host must send the byte sequence specified below to the target. Byte 4..255 can be the custom message itself.

Table 5.1. Command Byte Sequence

Byte Number	Value/Type	Description
0	0x20	Message type: Command
1	payload length	The size of the uint8array struct including its length and payload members.
2	0xFF	Message class: User messaging
3	0x00	Message ID
4..255	uint8array	The user message. The first byte is the length of the message. The next bytes are the message bytes

Once the target receives this byte sequence, it must response with the byte sequence specified below. Byte 6 to 255 can be used for the custom response.

Table 5.2. Response Byte Sequence

Byte Number	Value/Type	Description
0	0x20	Message type: Command
1	payload length	The size of the uint8array struct including its length and payload members.
2	0xFF	Message class: User messaging
3	0x00	Message ID
4-5	uint16	Result code: 0: Success / Non-0: An error occurred
6..255	uint8array	The user message. The first byte is the length of the message. The next bytes are the response message bytes.

Additionally, the `evt_user_message_to_host event` can be used by the target to send a message to NCP host. The target must send the byte sequence specified below. Byte 4..255 can be the custom message itself.

Table 5.3. Event Byte Sequence

Byte Number	Value/Type	Description
0	0xA0	Message type: Event
1	payload length	The size of the uint8array struct including its length and payload members.
2	0xFF	Message class: User messaging
3	0x00	Message ID

Byte Number	Value/Type	Description
4..255	uint8array	The user message. The first byte is the length of the message. The next bytes are the message bytes.

5.1 ncp_user_command_cb

The NCP Target calls `ncp_user_command_cb` if a command ID equals to `sl_bt_cmd_user_message_to_target_id`.

You can find the default implementation of `ncp_user_command_cb` in the *app.c* file and the declaration it in the *ncp.h* file.

In the first case, the Target echoes back the command to the Host, as a reply for the `USER_CMD_1` command. Also, it sends back the same as an event, to demonstrate how events can be sent to the Host.

It is also possible to initiate the communication from the Target. For the second user command, `USER_CMD_2` starts a timer, and when it expires, it will send a user event to the Host, using the function `sl_bt_send_evt_user_message_to_host`.

```
/**************************************************************************//**
 * User command handler.
 *****************************************************************************/
void ncp_user_command_cb(void* data)
{
  uint8array* cmd = (uint8array*)data;
  user_cmd_t* user_cmd = (user_cmd_t*)cmd->data;

  switch (user_cmd->hdr) {
    // User command 1.
    case USER_CMD_1_ID:
      ////////////////////////////////////////////////
      // Add your user command handler code here! //
      ////////////////////////////////////////////////

      // Example: respond to user command and send user event back too.
      // Sending back received command both as response and event.
      ncp_user_command_rsp(SL_STATUS_OK, cmd->len, cmd->data);
      ncp_user_evt(cmd->len, cmd->data);
      break;

    // User command 2.
    case USER_CMD_2_ID:
      ////////////////////////////////////////////////
      // Add your user command handler code here! //
      ////////////////////////////////////////////////
      sl_bt_system_set_soft_timer(2*SECOND, USER_EVENT_ID, 1)
      break;

    // Unknown user command.
    default:
      // Send error response back to host.
      ncp_user_command_rsp(SL_STATUS_FAIL, 0, NULL);
      break;
  }
}

…
/* Sending target-initiated message from the timer event handler */

  /* soft timer fired event */
   case sl_bt_evt_system_soft_timer_id:
   /* user event can be sent out to HOST */
   /* send custom message from the Target */
    sl_bt_send_evt_user_message_to_host(data_len, data)
  break;
```

6. Firmware Update

The ability to update the firmware of units already deployed in the field is a common requirement for many products. For example, it may be necessary to add new features to products after the first version has been launched. If a software bug or some unanticipated compatibility issue is identified after the product has been shipped, it is essential to provide a firmware update that fixes the problem, without the need to recall units or for the customer to take them to service for reprogramming.

Before Bluetooth SDK version 2.7.0, legacy OTA and Legacy UART DFU bootloader methods were supported for some devices. These legacy methods were deprecated in version 2.6.0, and the software was removed in version 2.7.0. Only the Gecko Bootloader is supported by Wireless Gecko devices using Network Co-Processor Mode.

The Gecko Bootloader was developed to unify the firmware update methods across different Silicon Labs SDKs, stacks, MCUs, and WMCUs. Key features of the Gecko Bootloader are:

- Useable across families (MCU and WMCU)
- Supports image verification and encryption for:
  - Integrity
  - Authenticity
  - Confidentiality
- In-field updateable
- Configurable

The Gecko Bootloader has a two-stage design, where a minimal first stage bootloader is used to update the main bootloader. This allows for in-field updates of the main bootloader, including adding new capabilities, changing communication protocols, adding new security features and fixes, and so on.

The Gecko Bootloader can be configured to function as a standalone bootloader or an application bootloader, depending on the plugin configuration. To function as a standalone bootloader, a plugin providing a communication interface such as UART has to be configured. To function as an application bootloader, a plugin providing a bootloader storage implementation must be configured. Plugins can be enabled and configured through the Simplicity Studio IDE.

For more information about the Gecko Bootloader and its use with the Bluetooth SDK, see UG266: Silicon Labs Gecko Bootloader User's Guide and AN1086: Using the Gecko Bootloader with Silicon Labs Bluetooth® Applications.

7. Application Loader

A Bluetooth application developed with the Silicon Labs Bluetooth SDK comprises two parts: an application loader called AppLoader and the user application. AppLoader is a small standalone application that is required to support in-place OTA updates. AppLoader can run independently of the user application. It contains a minimal version of Bluetooth stack, including only those features that are necessary to perform the OTA update. Any Bluetooth features that are not necessary to support OTA updates are disabled in AppLoader to minimize the flash footprint.

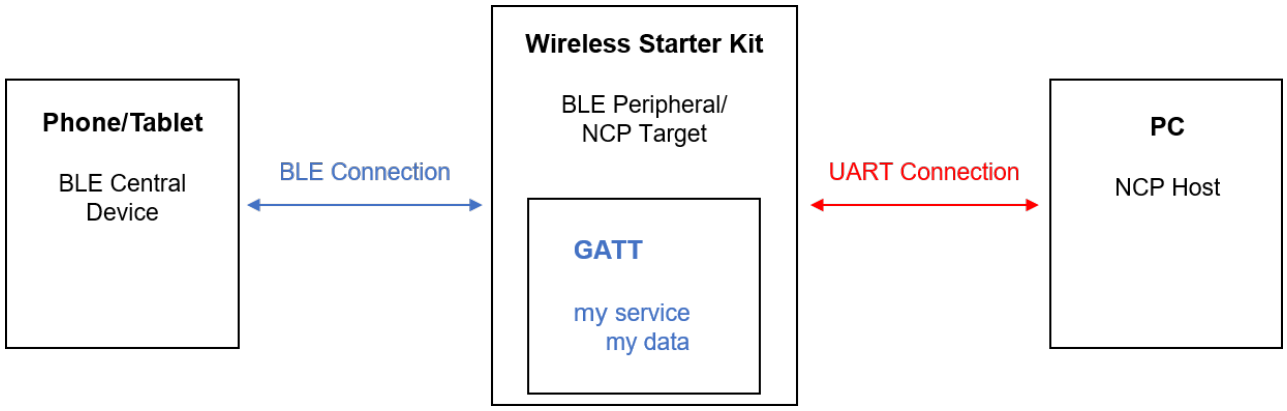
The AppLoader features and limitations are summarized below:

- Enables OTA updating of the user application.
- The AppLoader itself can also be updated.
- Only one Bluetooth connection is supported, GATT server role only.
- Encryption and other security features (bonding and so on) are not supported.

For more information, see AN1086: Using the Gecko Bootloader with Silicon Labs Bluetooth® Applications.

8. Adding a New Service to the NCP Empty Example

This chapter describes how to add a custom Bluetooth service to the **NCP empty** example. The service will have a characteristic to receive data. When the central device (tablet/phone) writes this characteristic, the peripheral (WSTK – NCP Target) will forward this data to the NCP Host. The NCP Host will print out the actual data to the PC console.



To implement this application, you need to make these changes:

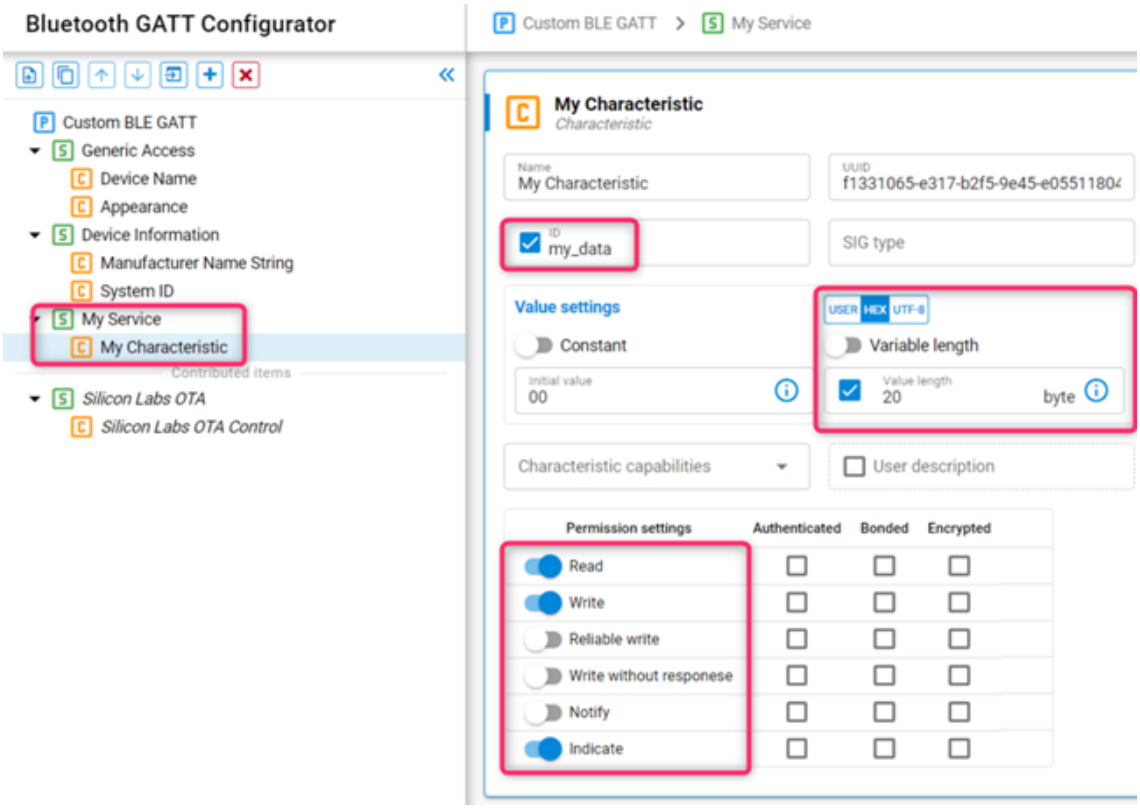
- Add the new service and characteristics to the (static) GATT database.
- Add the new GATT both to the Host and Target project.
- Handle GATT change event (sl_bt_gatt_server_attribute_value) in the Host project.

If you want to build the GATT database dynamically from the host software (preferred method), see the next section.

8.1 Update the Target Project

The GATT database can be modified using the visual GATT editor in Simplicity Studio. For more details, refer to *QSG169: Bluetooth® SDK v3.x Quick Start Guide*.

Add a new service definition after the last service as shown in the following figure.



Once the GATT is configured, save the file, and check the folder named **autogen**. You should see a new handle for the `my_data` characteristic in the newly generated *gatt_db.h*.

Once generation is completed you can rebuild the project. Click the debug button to flash the WSTK with the new firmware.

Note: After saving any changes to the GATT content, the *gatt_db.c* and *gatt_db.h* files will be automatically re-generated Then you have to re-compile the project.

8.2 Update the Host Project

1. In order to use the GATT handles on the NCP Host side, you need to add *gatt_db.h* to the Host project. *gatt_db.h* is already generated to your workspace in the procedure in section 8.1 Update the Target Project.
2. Copy *gatt_db.h* to the host project folder:

   ..\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\examples_ncp_host\empty\
3. Optional: If you want to be sure that the Host and Target GATT are always in sync, you can modify the makefile to include the *gatt_db.h* from the folder of your Target project. Add the path to your project workspace.

```
INCLUDEPATHS += \
-I../../../../<your project path>/autogen \
-I../../../../protocol/bluetooth/ble_stack/inc/common \
-I../../../../protocol/bluetooth/ble_stack/inc/host
```

4. In the following step, you have to modify the file *app.c* (shown in section 6. Firmware Update). First include *gatt_db.h* to *app.c*.

```
/* include GATT handles*/
#include "gatt_db.h"
```

5. Then add the callback function that reacts to the GATT change. In this case, it will print out the content of the characteristic.

```
void AttrValueChanged_my_data(uint8array *value)
{
  uint8_t i;
  for (i = 0; i < value->len; i++){
      app_log("my_data[%d] = 0x%x \r\n",i,value->data[i]);
  }
  app_log("\r\n");
}
```

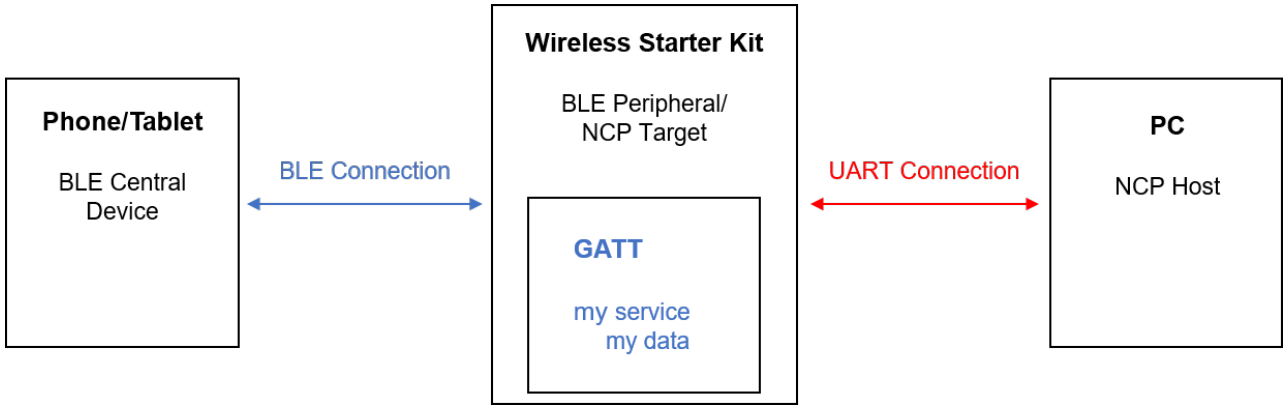
6. Now add the `sl_bt_evt_gatt_server_attribute_value_id` event to the switch case.

```
case sl_bt_evt_gatt_server_attribute_value_id:
  // Check if the event is because of the my_data changed by the remote GATT client
  if ( gattdb_my_data == evt->data.evt_gatt_server_attribute_value.attribute ){
   // Call my handler
   AttrValueChanged_my_data(&(evt->data.evt_gatt_server_attribute_value.value));
   }
break;
```

7. Now you can rebuild the host application. See the build process with MinGW in section 3.2 Building the NCP Host Example on Windows. The test setup is described in section 9.2 Testing.

9. Adding a New Service to the NCP Example with the Dynamic GATT API

This chapter describes how to add a custom Bluetooth service to the **NCP** example using the dynamic GATT API. The service has a characteristic to receive data. When the central device (tablet/phone) writes this characteristic, the peripheral (WSTK – NCP Target) forwards this data to the NCP Host. The NCP Host prints out the actual data to the PC console. With this method, the Target project does not need to be modified.



To implement this application, you need to make these changes:

- Create a GATT database from the Host project
- Handle the GATT change event (sl_bt_evt_gatt_server_attribute_value) in the Host project

9.1 Update the Host Project

The GATT database will be built with the APIs provided by the Dynamic GATT Configurator component. See the Bluetooth API reference manual on docs.silabs.com, section "GATT Database", for more details.

In the code snippet below, the custom service and characteristic is added to the database. The service will be a primary service, defined with a 16-byte long UUID, and it will be advertised.

The characteristic has the following properties:

- Read, Write, Indicate
- Value length: 20 bytes
- Value max length: 20 bytes
- 16-byte long UUID

1. Add this to the boot event.

```
uint8_t uuid_service[16] = {…} //define your 128bit service UUID, you can use a random number
uint8_t uuid_characteristic[16] = {…} //define your 128bit characteristic UUID

//create a session for the database update
sl_bt_gattdb_new_session(&session);
//add our service to the database, as an advertised primary service
sl_bt_gattdb_add_service(session, sl_bt_gattdb_primary_service, SL_BT_GATTDB_ADVERTISED_SERVICE, 16,
                         uuid_service, &service);
//define the following properties: read, write, indicate
property = (SL_BT_GATTDB_CHARACTERISTIC_READ | SL_BT_GATTDB_CHARACTERISTIC_INDICATE |
            SL_BT_GATTDB_CHARACTERISTIC_WRITE);
//add our characteristic to the service
sl_bt_gattdb_add_uuid128_characteristic(session, service, property, 0, 0, uuid_characteristic,
                                        sl_bt_gattdb_fixed_length_value, 20, 20, &value, &characteristic);
//activate the new service
sl_bt_gattdb_start_service(session, service);
//activate the new characteristic
sl_bt_gattdb_start_characteristic(session, characteristic);
//store the handle of the characteristic for future reference
gattdb_my_data = characteristic;
//save changes and close the database editing session
sl_bt_gattdb_commit(session);
```

2. Add the callback function that reacts to the GATT change. In this case, it prints out the content of the characteristic.

```
void AttrValueChanged_my_data(uint8array *value)
{
  uint8_t i;
  for (i = 0; i < value->len; i++){
      app_log("my_data[%d] = 0x%x \r\n",i,value->data[i]);
  }
  app_log("\r\n");
}
```

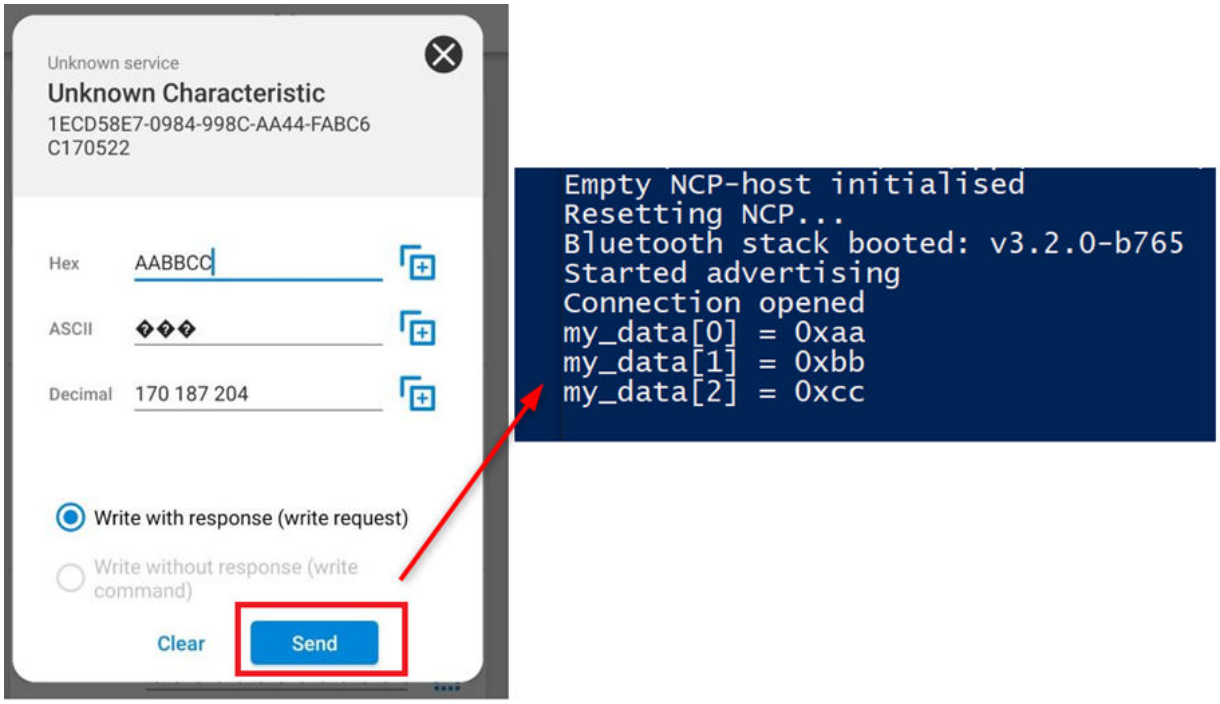
3. Add the `sl_bt_evt_gatt_server_attribute_value_id` event to the switch case.

```
case sl_bt_evt_gatt_server_attribute_value_id:
  // Check if the event is because of the my_data changed by the remote GATT client
  if ( gattdb_my_data == evt->data.evt_gatt_server_attribute_value.attribute ){
   // Call my handler
   AttrValueChanged_my_data(&(evt->data.evt_gatt_server_attribute_value.value));
   }
break;
```

Now you can rebuild the host application. See the build process with MinGW in 3.2 Building the NCP Host Example on Windows.

9.2 Testing

1. Start the host application from the *\exe* folder.
2. Once the PC is connected to WSTK (via UART), the WSTK starts advertising on Bluetooth.
3. If you connect via tablet/phone you can write the newly created `my_data` characteristic in the GATT. For this, you can use the EFR Connect app provided by Silicon Labs.
4. Browse to the `my_data` characteristic and write something to it. The data will be printed by the host application.



Smart. Connected. Energy-Friendly.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com