AN1042: Using the v2.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 Bluetooth Stack in Network Co-Processor (NCP) mode. The document covers the C language application development flow and the use of BGTool and Simplicity Studio, then 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
- Describes customizing the built-in examples

1. Introduction

The 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 is an ANSI C reference implementation of the BGAPI protocol, which can be used in the NCP Host Application.

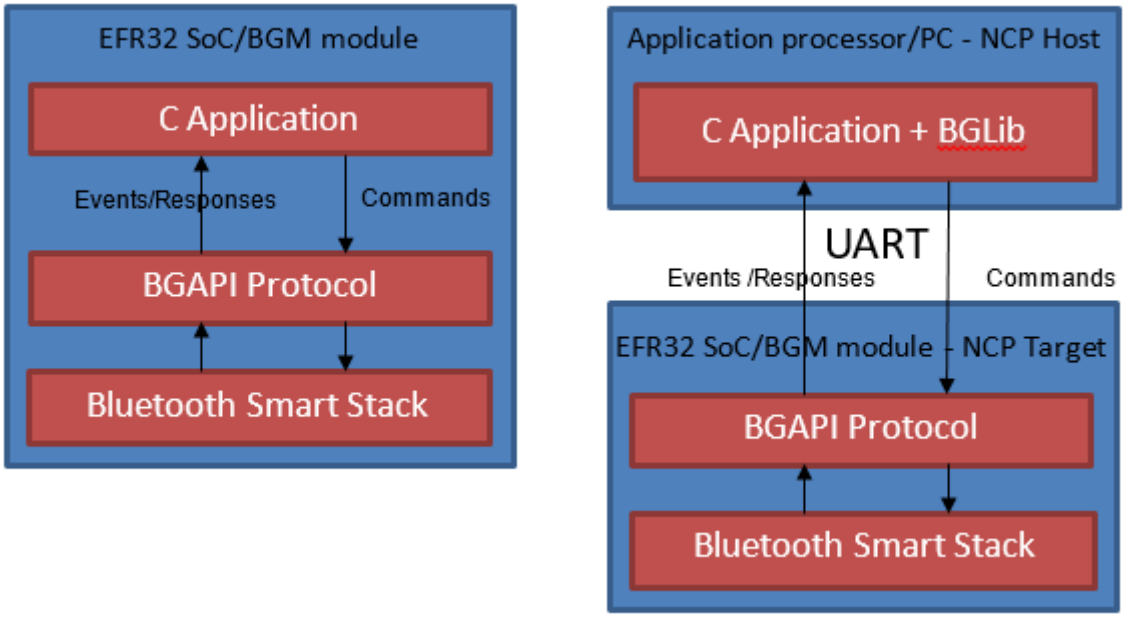


Figure 1.1. SoC vs NCP System Models

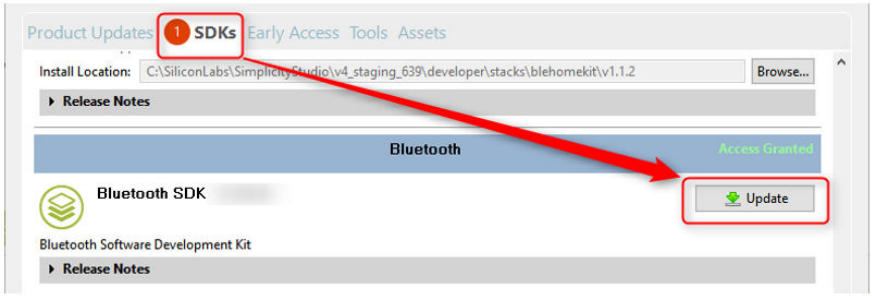
2. NCP Target Development

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. You can download it from the Silicon Labs website: http://www.silabs.com/simplicity

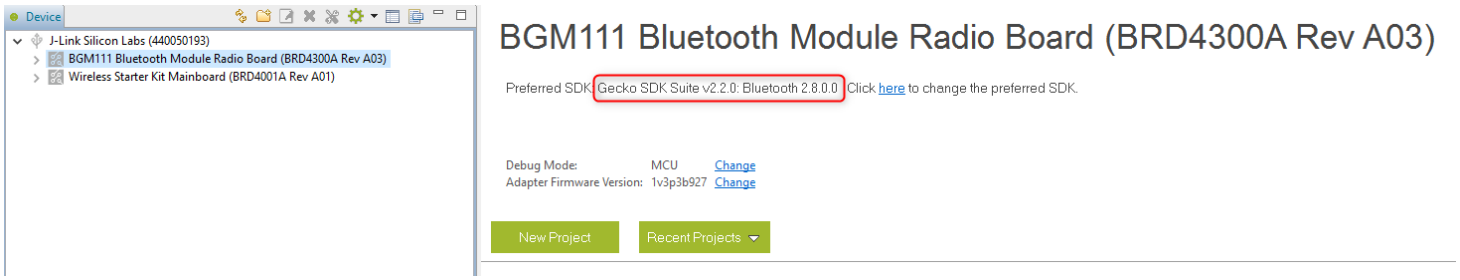
Before installing Simplicity Studio, connect the WSTK and your PC with a USB cable. The white switch located on left side of the WSTK must be in the AEM position. See *QSG139: Bluetooth Development with Simplicity Studio* for details on installing Studio and the Bluetooth SDK.

If you have already installed Simplicity Studio you can download the Bluetooth SDK through the Simplicity Studio Package Manager. To open it, click the arrow icon, then go to the Stacks tab. Finally, click **[Install]** next to the Bluetooth SDK. .

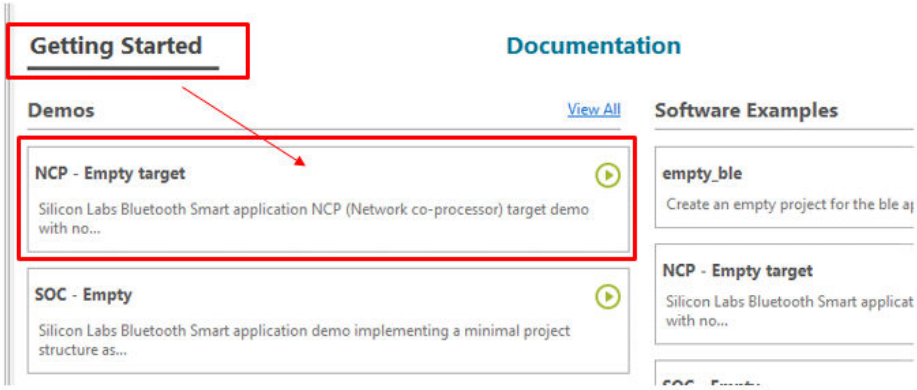


To develop in C you need not only Simplicity Studio but also a supported compiler. *UG136: Silicon Labs Bluetooth® C Application Developer's Guide* lists the supported compilers.

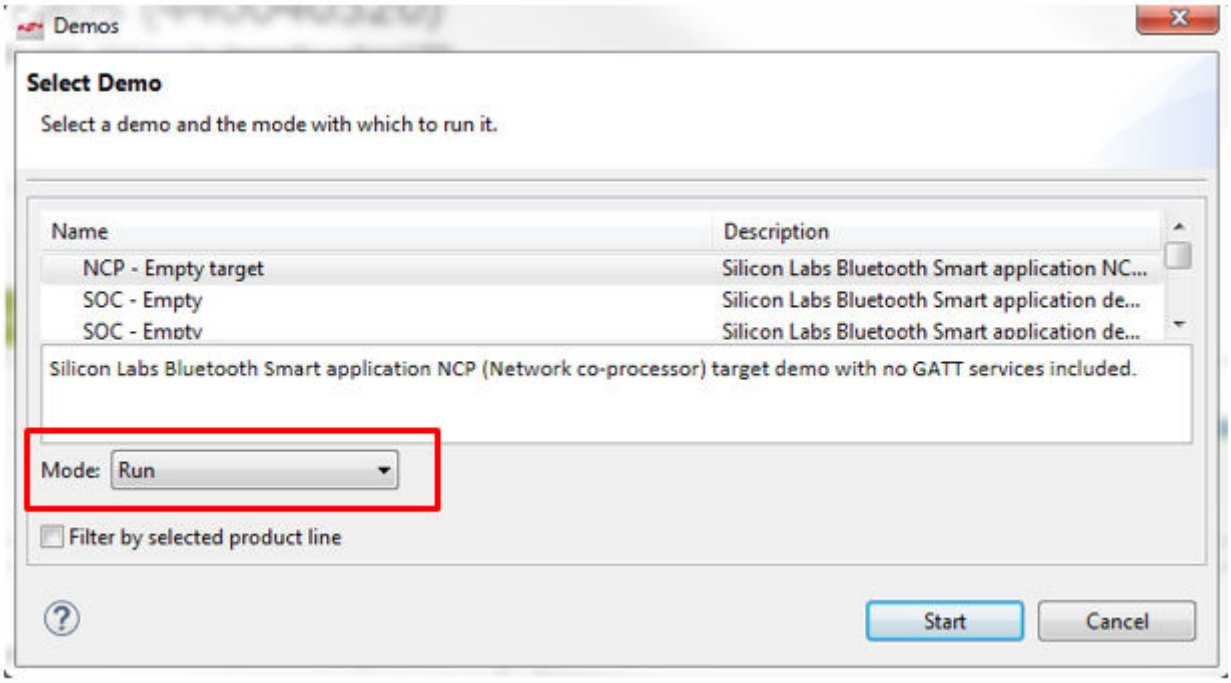
The NCP target firmware comes with the Bluetooth SDK. It is available in a precompiled binary format and as a project file you can build. The following procedures describe how to install the precompiled binary image and how to build and install the example project. Note that Simplicity Studio only shows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



1. To download the prebuilt NCP target firmware, click the **NCP - Empty target** project under Demos.



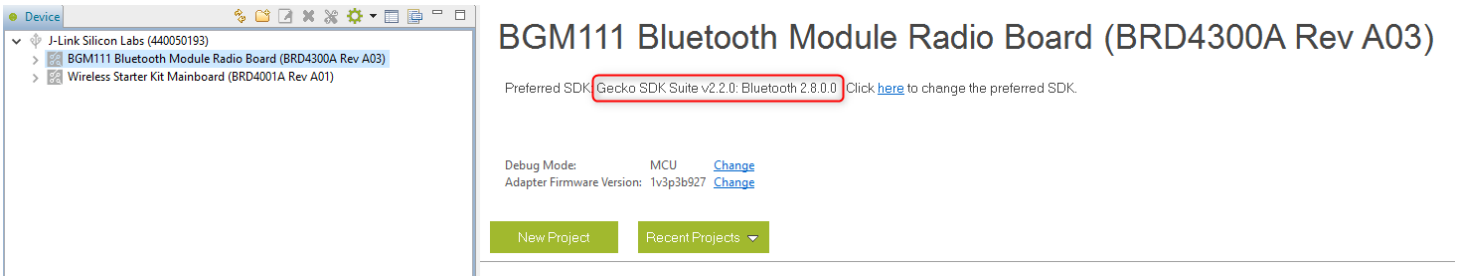
2. By default, all demos run under Energy Profiler, but you can only flash and run the firmware by choosing the Mode *Run* as shown below. Click [**Start**].



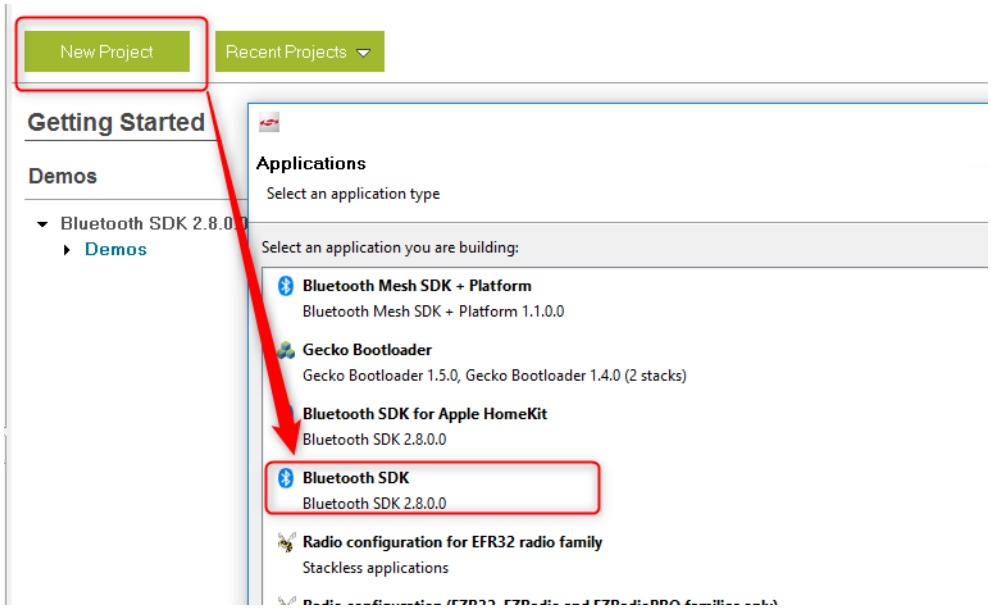
The following procedure describes how to build and load the example code. This procedure assumes you have already loaded a Gecko Bootloader in one of the following three ways:

- Loading the Gecko Bootloader precompiled binary from the list of Demos. For an NCP application you should load the BGAPI UART DFU bootloader.
- Loading the “NCP – Empty Target” precompiled binary as described in the previous procedure, which will also load the UART DFU Gecko Bootoader.
- Building and loading your own Gecko Bootloader combined image called **<projectname>-combined.s37**, as described in chapter 4 of *UG266: Silicon Labs Gecko Bootloader User’s Guide*.

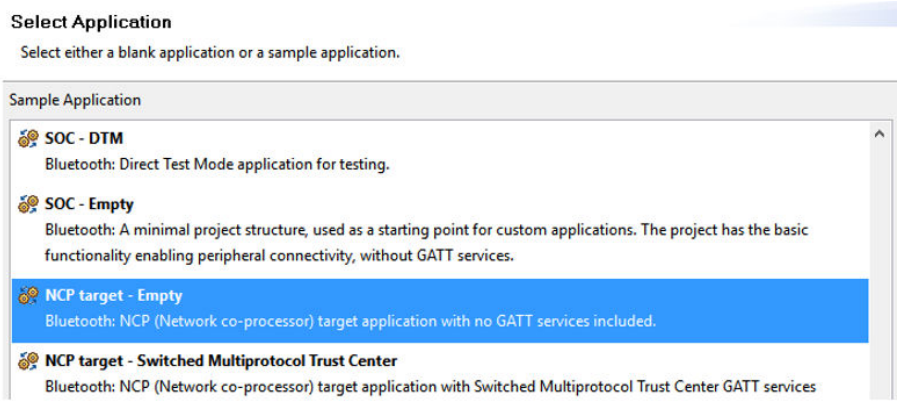
1. Make sure that the preferred SDK is *Gecko SDK Suite: Bluetooth*.



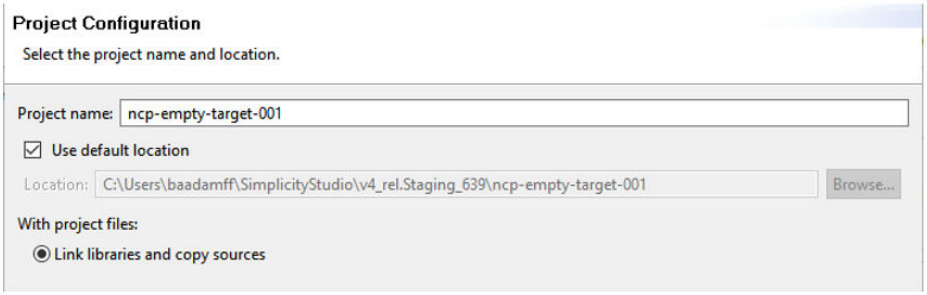
2. Click [**New Project**], and then Bluetooth SDK.



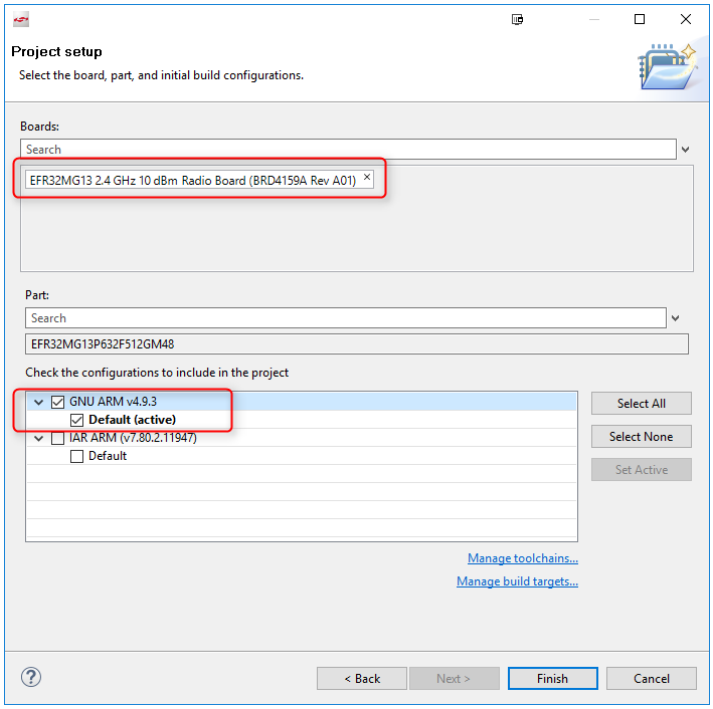
3. Select the **NCP target – Empty** example application, then click [**Next**].



4. Name your project, then click [**Next**].



5. Make sure that the board shown on the dialog is the one you have connected. Select a compiler and click [**Finish**].



6. Now your project is ready to build and flash. Click  (Debug) 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  , click the project .isc file in the Project Explorer view. It may not be fully selected.

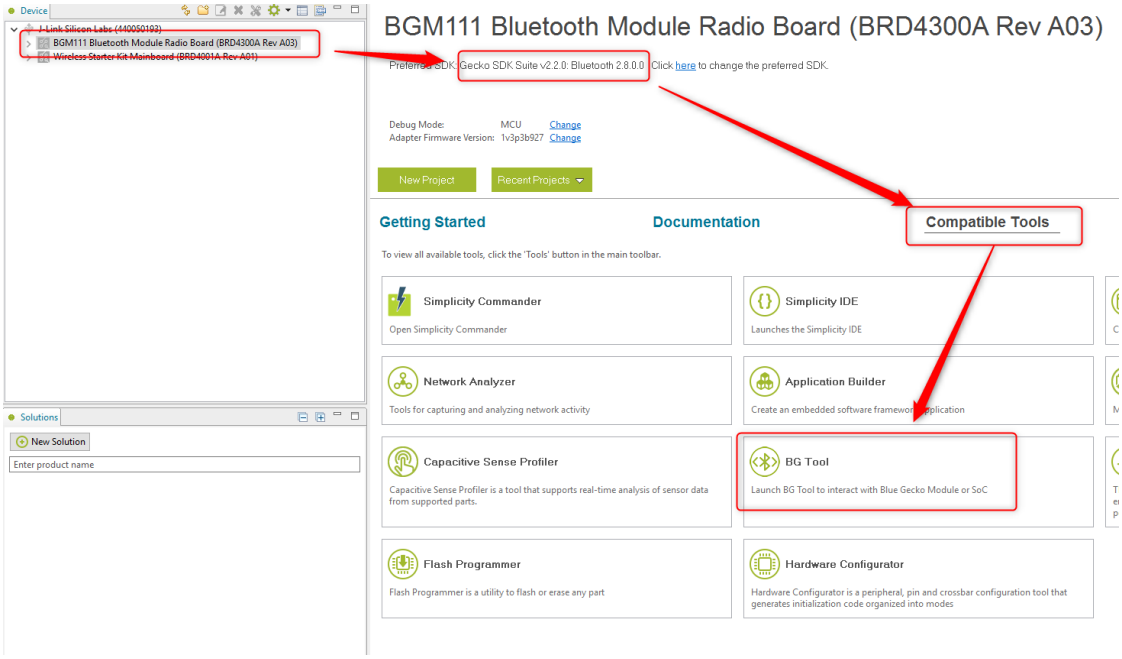
3. NCP Host Development

This chapter introduces the BGTool Interactive View which can be used to send BGAPI commands from a GUI, then walks through on the building process of the PC Host example provided by the SDK.

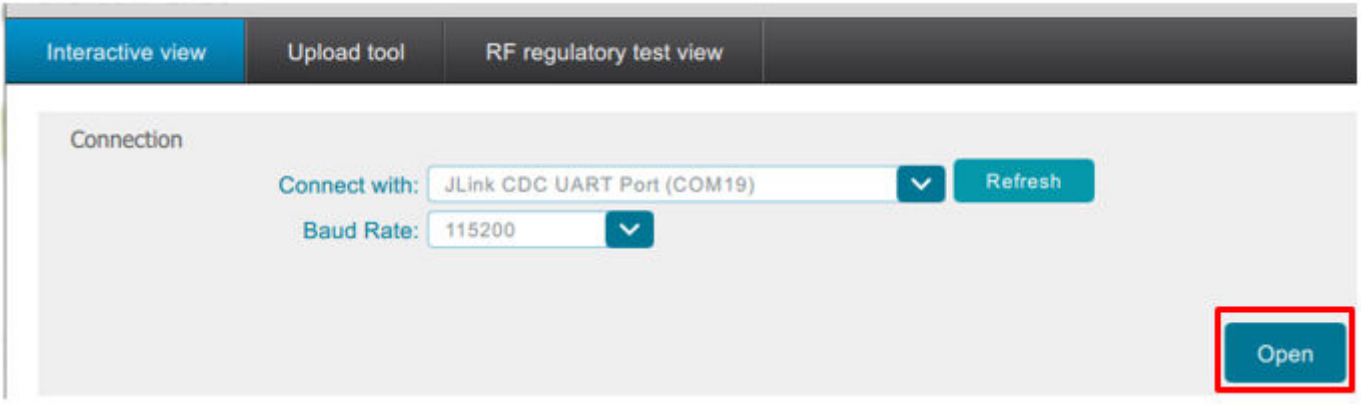
3.1 Host – BGTool Interactive View

You can use the BGTool UI to issue BGAPI commands.

1. To open BGTool, make sure that the correct board selected and the preferred SDK is *Gecko SDK Suite:Bluetooth.* Then click the **Compatible Tools** tab, scroll down if necessary, and click **BG Tool**.

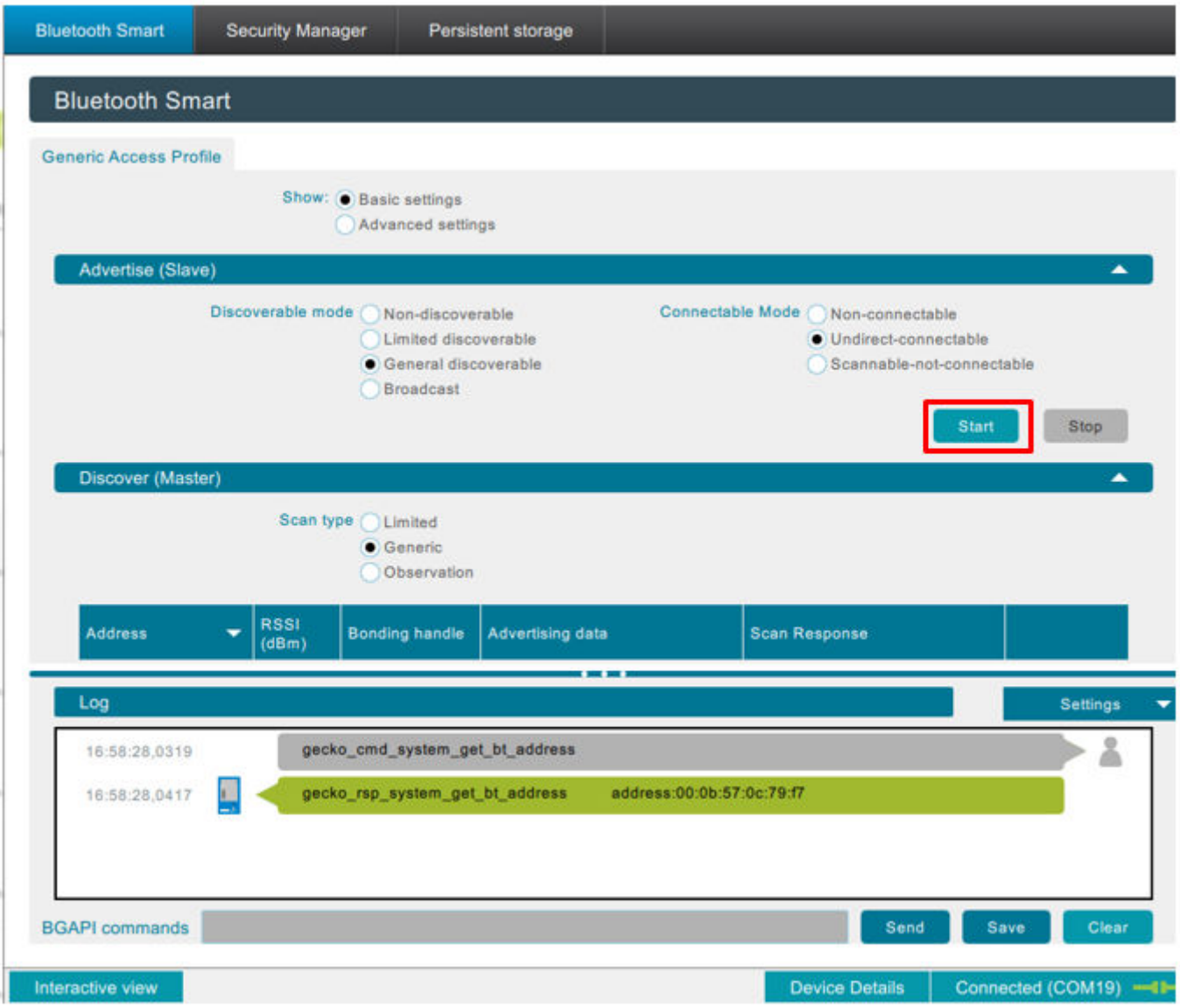


2. Now you need to set up the correct COM port and baud rate in BGTool. Select *JLink CDC UART Port* @ 115200 baud rate, then click [**Open**].

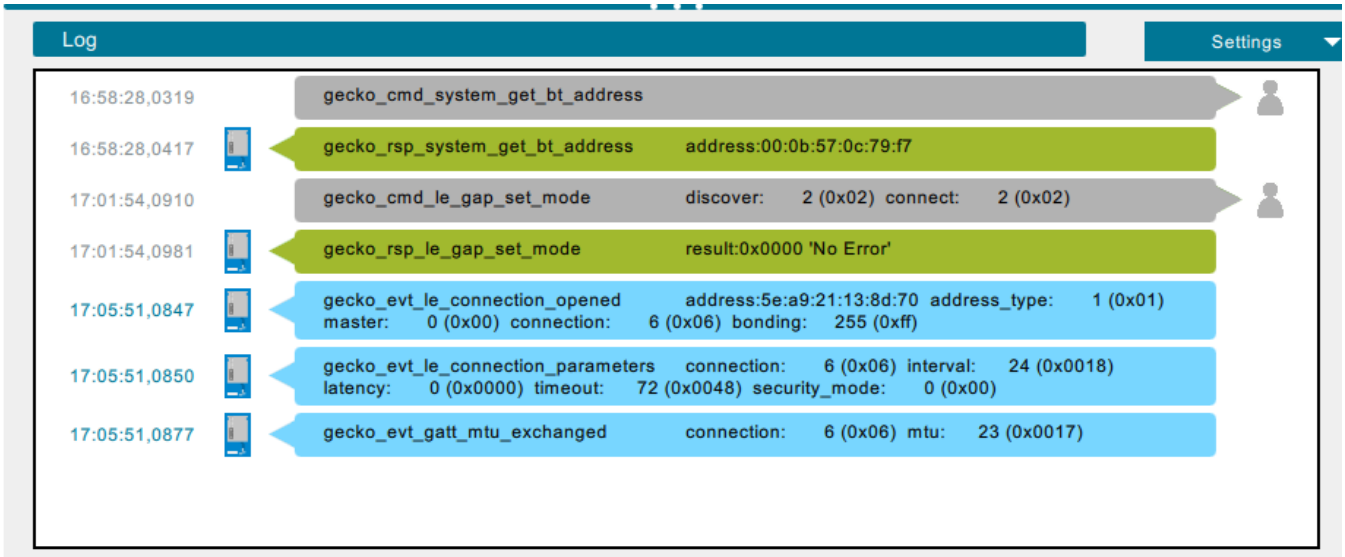


3. Once the UART connection to the WSTK is established, an Interactive view opens, which you can use to issue BGAPI commands. Check the log for the NCP target response and status messages.

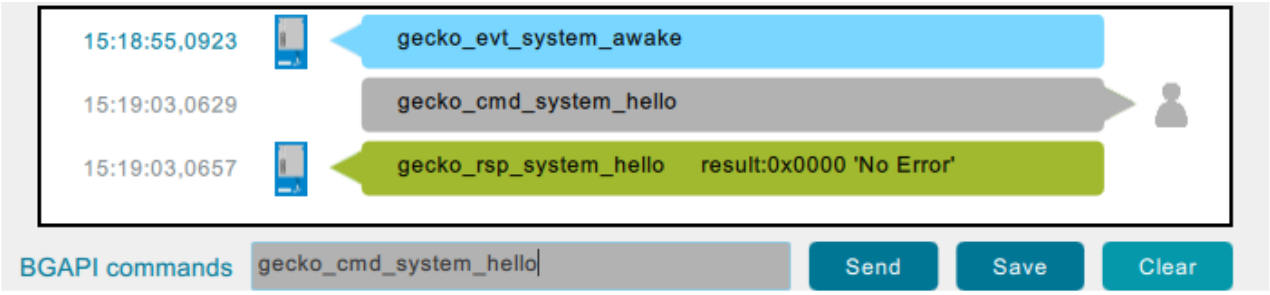
4. Click [**Start**] to start advertising.



5. After you click [**Start**], the NCP target accepts Bluetooth connections. If you connect to your WSTK with another a master device (for example with your phone), you can see BGAPI communication in the log.



6. You can also issue commands manually. For example, the “system hello” command can be used at any time to verify that communication between the host and the device is working.



3.2 Building the NCP Host Example on Windows

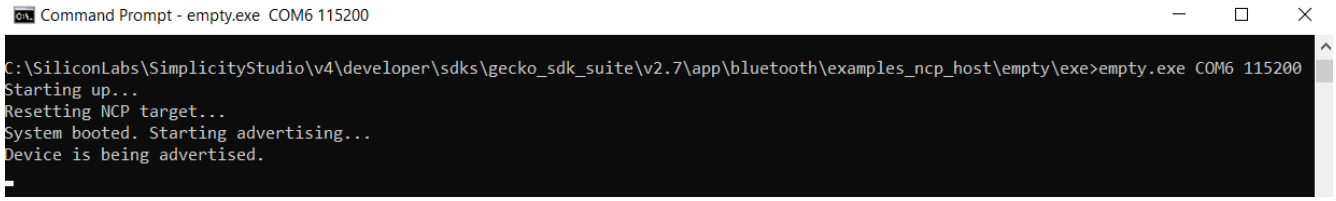
The Bluetooth SDK contains a generic NCP Host example project for the PC. This example can be compiled on Windows or any POSIX OS. This section goes through the build process on Windows.

1. You can use GCC to build the example. First install MinGW, which is a complete runtime environment for GCC. You can download it from here :

   http://www.mingw.org/
2. Make sure that the C:\MinGW\bin folder is added to your Path environmental variable.
3. Change to the NCP Host example folder, where <version> varies by SDK version:

```
cd                c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth
\examples_ncp_host\empty\
```

4. Build the project with the command: `mingw32-make`
5. The build output is created in a new **exe** folder. Go to this folder with `cd exe`, 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.
6. Once the UART connection with the WSTK is established you should see the following:



Now you can connect to the WSTK over Bluetooth.

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-empty-target* Simplicity Studio project. You can find a general project description in *UG136: Silicon Labs Bluetooth® C Application Developer's Guide*.

The main differences from an SoC project are:

- UART configuration
- Stack initialization
- Main loop/event handling
- Sleep and wake up management

4.1.1 UART Configuration

The UART must be configured, as this is the communication interface between the NCP Target and Host. It is recommended to use 115200 baud rate with 8N1 and hardware flow control. This initialization is implemented by the `ncp_usart_init` function in the *ncp_usart.c* file.

The UART settings can be configured with #defines in *hal-config-board.h* and *hal-config.h*. The *hal-config.h* contains the generic UART options like baud rate, flow control, and so on. The *hal-config-board.h* defines the board-specific configuration, such as which peripheral to use (USART0/USART1) and which pins to use.

On the target, the stack registers callbacks that allow it to receive and transmit messages to the application running on the host. When the NCP UART module is initialized, it configures the 8 bit USART timer for RX timeout detection and sets it to 0x30 by default. The 8 bit timer is decremented at a frequency of HAL_UARTNCP_BAUD_RATE bauds, that is (1/HAL_UARTNCP_BAUD_RATE) seconds.

Additionally, a software timeout counter corresponding to the number of RX timeout detection retries is also defined. The RX timeout detection retry counter is set by default in the code to (HAL_UARTNCP_BAUD_RATE/64).

The following formula indicates the overall timeout value using the default USART timer value of 0x30 (48):

UART_RX_TIMEOUT = (48 * (1/HAL_UARTNCP_BAUD_RATE) * (HAL_UARTNCP_BAUD_RATE/64)) = 48/64 which result in 750 ms.

The timeout value should be constant given the value used by default by the software timer (HAL_UARTNCP_BAUD_RATE /64)

When the software RX timeout detection retry counter reaches 0 in the NCP_USART_IRQ_NAME interrupt service routine, an NCP_USART_TIMEOUT_SIGNAL is signaled to the stack, which sends a system_error event to the host device with the `bg_err_command_incomplete` error code.

Important notice: This formula is invalid for the BG1 chip family. See "USART_E202 — Incorrect 8-bit Timer Operation in Asynchronous Mode" in the document *Blue Gecko Bluetooth® Low Energy SoCBlue Gecko SoC (EFR32BG1) Errata* (https://www.silabs.com/documents/login/errata/efr32bg1-errata.pdf) for more detail.

The configuration options in *hal-config.h* are the following:

```
#define HAL_UARTNCP_BAUD_RATE
```

This define sets the baud rate of the UART. Use the default 115200 value if you use VCOM.

```
#define HAL_VCOM_ENABLE
```

This define enables/disables forwarding the NCP communication to the J-Link CDC Virtual COM port on the WSTK. If it is set to 1, VCOM is enabled. If it is set to 0, VCOM is disabled. If you use a custom board without a WSTK, do not define this symbol.

```
#define HAL_UARTNCP_FLOW_CONTROL
```

This define selects the possible flow control option. It is recommended to use hardware flow control as it set by default. However the following flow control options are supported:

`HAL_USART_FLOW_CONTROL_NONE` - No flow control used.

`HAL_USART_FLOW_CONTROL_HWUART` - RTS and CTS pins are controlled by the hardware.

`#define NCP_USART_STOPBITS`

This define sets the number of stop bits. Use the *USART_Stopbits_TypeDef* options. The default is *usartStopbits1*.

`#define NCP_USART_PARITY`

This define is set the parity. Use the *USART_Parity_TypeDef* options. The default is *usartNoParity*.

`#define NCP_USART_OVS`

This define selects the oversampling mode. Use the *USART_OVS_TypeDef* options. The default is *usartOVS16*.

`#define NCP_USART_MVDIS`

This define enables the UART majority vote feature. This is disabled by default

`#define NCP_USART_FLOW_CONTROL_ENABLED`

This define enables flow control. This is enabled by default.

`#define NCP_DEEP_SLEEP_ENABLED`

This define enables the deep sleep function of the stack. This is disabled by default. When deep sleep mode is enabled a wake-up pin must be configured.

`#define NCP_WAKEUP_PIN`

The wake-up pin is an input pin for the NCP target so the NCP host can wake the target up. If deep sleep is enabled the wake-up pin number must be defined.

`#define NCP_WAKEUP_PORT`

If deep sleep is enabled the wake-up pin's port must be defined. Use *GPIO_Port_TypeDef* to select the port.

`#define NCP_WAKEUP_POLARITY`

If this is defined as 0, then the wake-up signal must be driven low to wake up the NCP target. If it is defined as 1, the wake-up signal must be driven high to wake up the NCP target.

`#define NCP_HOST_WAKEUP_ENABLED`

This option enables the NCP target's ability to wake up the NCP host before sending any BGAPI commands to the NCP host. This signal is an output for the NCP target. Any available GPIO pin can be configured as a host wake-up pin. The polarity is also configurable. This feature is disabled by default.

`#define NCP_HOST_WAKEUP_PORT`

If the NCP host wake-up feature is enabled, the host wake-up pin's port must be defined. Use *GPIO_Port_TypeDef* to define the port.

`#define NCP_HOST_WAKEUP_PIN`

This define sets the pin number of the host wake up pin. The NCP host wake-up pin is an output pin for the NCP target so the NCP target can wake the host up.

`#define NCP_HOST_WAKEUP_POLARITY`

If this is defined as 0, then the wake up signal goes low to wake up the NCP host.

The configuration options in *hal-config-board.h* are the following:

`#define BSP_VCOM_ENABLE_PIN`

This define sets the pin of the virtual COM control switch. This switch can connect the USART RX/TX lines with the board controller to enable serial communication over USB.

`#define BSP_VCOM_ENABLE_PORT`

This define sets the port of the virtual COM control switch.

`#define BSP_UARTNCP_USART_PORT`

This define sets the USART0 or USART1 as the peripheral to be used. Typically it is `HAL_SERIAL_PORT_USART0` or `HAL_SERIAL_PORT_USART1`.

`#define BSP_UARTNCP_XXX_PIN`

`#define BSP_UARTNCP_XXX_PORT`

`#define BSP_UARTNCP_XXX_LOC`

These defines set the pin, port, and location for the Rx, Tx, RTS, and CTS pins. Use the *GPIO_Port_TypeDef* to select the port. Use a simple pin number to define the pin within the port. The location number for each pin can be found in the device's datasheet.

4.1.2 Stack Initialization

To run the Bluetooth stack and an application on a Wireless Gecko, the stack has to be initialized using the `gecko_init()` function. In the case of the NCP target, the configuration looks like this:

```
// Gecko configuration parameters (see gecko_configuration.h)
static const gecko_configuration_t config = {
  .config_flags = 0,
#if defined(NCP_DEEP_SLEEP_ENABLED)
  .sleep.flags = SLEEP_FLAGS_DEEP_SLEEP_ENABLE,
#endif
  .bluetooth.max_connections = MAX_CONNECTIONS,
  .bluetooth.max_advertisers = MAX_ADVERTISERS,
  .bluetooth.heap = bluetooth_stack_heap,
  .bluetooth.heap_size = sizeof(bluetooth_stack_heap),
  .bluetooth.sleep_clock_accuracy = 100, // ppm
  .gattdb = &bg_gattdb_data,
#if (HAL_PA_ENABLE) && defined(FEATURE_PA_HIGH_POWER)
  .pa.config_enable = 1, // Enable high power PA
  .pa.input = GECKO_RADIO_PA_INPUT_VBAT, // Configure PA input to VBAT
#endif // (HAL_PA_ENABLE) && defined(FEATURE_PA_HIGH_POWER)
};
```

The options are documented in *UG136: Silicon Labs Bluetooth® C Application Developer's Guide*.

4.1.3 Main Loop/Event Handling

Once the USART and Bluetooth stack are initialized, the main loop starts with forwarding the incoming BGPAPI commands from the UART to the stack by the `ncp_handle_commands()` function. Then the `gecko_peek_event()` checks for stack events to be processed in the `while (evt)` loop.

As a next step, the `ncp_handle_event(evt)` and `local_handle_event(evt)` functions are called.

The `ncp_handle_event(evt)` function is used to handle the wake-up and timeout external signals.

The `local_handle_event(evt)` function can be used to offload some tasks from the host, so the target can handle stack events here. But by default, this function does not handle any stack events. The prototype of stack events and commands can be found in ncp_gecko.h.

If an event is not handled by these functions locally it goes to an event queue by calling the `ncp_transmit_enqueue(evt)` function. At the end of the loop the `ncp_transmit()` function sends all the events from the queue to the NCP host to be processed there.

```
  while (1) {
    struct gecko_cmd_packet *evt;
    ncp_handle_command();
    /* Check for stack event. */
    evt = gecko_peek_event();
    while (evt) {
      if (!ncp_handle_event(evt) && !local_handle_event(evt)) {
        // send out the event if not handled either by NCP or locally
        ncp_transmit_enqueue(evt);
      }
      evt = gecko_peek_event();
    }
    ncp_transmit();

    CORE_DECLARE_IRQ_STATE;
    CORE_ENTER_ATOMIC();
    gecko_sleep_for_ms(gecko_can_sleep_ms());
    CORE_EXIT_ATOMIC();
  }
```

4.1.4 Enabling Sleep

The ncp-empty-target example project does not enable sleep mode by default. When sleep mode is enabled it is also 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-

First enable the deep sleep function in *hal_conf.h*

```
#define NCP_DEEP_SLEEP_ENABLED 1
```

Then set the wake pin and port.

```
#define NCP_WAKEUP_PIN 6
#define NCP_WAKEUP_PORT gpioPortF
```

Then define the polarity. Here it is set active high.

```
#define NCP_WAKEUP_POLARITY 1
```

This completes the configuration. GPIO and interrupt settings are done by *ncp_usart.c.*

Note: When sleep is enabled the NCP device will send out the event `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.

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\v4\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\examples_ncp_host\empty\

c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\examples_ncp_host\common\

The files comprise only a few source and header files.

4.2.1 Project Files

makefile

Standard makefile containing a set of directives that can be used with the make build automation tool.

main.c

Main application logic to control the Blue Gecko module, and BGLib initialization and setup code.

app.c/app.h

Event handling and application code.

uart.h

Function declarations for UART port initialization and RX/TX data transfers common to most platforms and compatible with BGLib methods and callbacks.

uart_win.c

Implementations for the functions defined in uart.h that are appropriate for a Windows environment.

uart_posix

Implementations for the functions defined in uart.h that are appropriate for a POSIX OS environment.

infrastructure.h

Macros for different type conversions and helper functions.

4.2.2 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\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\ble_stack\inc\host\
- c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\ble_stack\inc\common\
- c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\ble_stack\src\host\

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 gecko_bglib.h file also contains function declarations for the basic packet reception, processing, and transmission functions.

The gecko_bglib.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).

4.2.3 Host Application Logic

1. Initialize BGLIB.

```
BGLIB_INITIALIZE_NONBLOCK(on_message_send, uartRx, uartRxPeek);
```

2. Initialize UART.

```
if (appSerialPortInit(argc, argv, 100) < 0) {
  printf("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.

```
gecko_cmd_system_reset(0);
```

4. Go to the main loop and wait for the NCP Target events.

```
while (1) {
  /* Check for stack event. */
  evt = gecko_peek_event();
  /* Run application and event handler. */
  appHandleEvents(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 (BGLIB_MSG_ID(evt->header)) {
  case gecko_evt_system_boot_id:

    appBooted = true;
    printf("System booted. Starting advertising... \n");

    /* Set advertising parameters. 100ms advertisement interval. All channels used.
     * The first two parameters are minimum and maximum advertising interval, both
     * in units of (milliseconds * 1.6). The third parameter '7' sets advertising
     * on all channels. */
    gecko_cmd_le_gap_set_adv_parameters(160, 160, 7);

    /* Start general advertising and enable connections. */
    gecko_cmd_le_gap_set_mode(le_gap_general_discoverable, le_gap_undirected_connectable);

    printf("Device is being advertised.\n");

    break;

  case gecko_evt_le_connection_closed_id:

    /* Restart general advertising and reenable connections after disconnection. */
    gecko_cmd_le_gap_set_mode(le_gap_general_discoverable, le_gap_undirected_connectable);

    break;

  default:
    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. Beginning with Bluetooth SDK version 2.6.0 the stack 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 fisrt 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 fisrt 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

Byte Number	Value/Type	Description
3	0x00	Message ID
4..255	uint8array	The user message. The fisrt byte is the length of the message. The next bytes are the message bytes

5.1 handle_user_command

The ncp target calls the *handle_user_command* function if a command ID is *gecko_cmd_user_message_to_target_id*.

By default *handle_user_command()* only sends back a *bg_err_not_implemented* error code but the user can edit the function in *handle_user_command.c* to send custom payload to the ncp host. See the example in the next chapter.

```
/**************************************************************************//**
 * @brief
 * Called when a user command (Message ID: gecko_cmd_user_message_to_target_id)
 * is received.
 *
 * @details
 * Implement this function if the BGAPI protocol is extended at application layer
 * between the host and target for data exchange. At the end of command handling,
 * a response to this command must be sent to the host. Use
 * gecko_send_rsp_user_message_to_target(uint16_t, uint8_t, uint8_t*) to send the
 * response.
 *
 * The target can also initiate the communication by sending event messages
 * (Message ID: gecko_evt_user_message_to_host_id) to the host by using API
 * gecko_send_evt_user_message_to_host(uint8_t, uint8_t*).
 *
 * Notice that events should not be sent in the context of user command handling.
 *
 * @param[in] payload
 *   the data payload in the command message
 *****************************************************************************/
void handle_user_command(const uint8array *payload);
```

The default implementation of *handle_user_command()* is in user_command.c:

```
# void handle_user_command(const uint8array *payload)
{
  gecko_send_rsp_user_message_to_target(bg_err_not_implemented, 0, NULL);
}
```

5.2 Custom Implementation of the *handle_user_command()* Function

The code snippet below immediately echoes back the payload received from the host. Receiving the user message starts a 2-second one-shot timer.

```
/**
 * handle for receiving message from the HOST
 */
void handle_user_command(const uint8array *payload)
{
  user_event_data_len = payload->len > USER_EVENT_DATA_MAX_LEN ? USER_EVENT_DATA_MAX_LEN : payload->len;
  memcpy(user_event_data, payload->data, user_event_data_len);

  /* Start one shot timer for generating a user event to demonstrate HOST to TARGET communication */
  gecko_cmd_hardware_set_soft_timer(2*SECOND, USER_EVENT_ID, 1);

  /* Loop back payload as a response */
  gecko_send_rsp_user_message_to_target(bg_err_success, payload->len, payload->data);
}
```

Note that in this case the communication is initiated by the NCP host. To initiate the communication by the target the *gecko_send_evt_user_message_to_host* API can be used as shown below. In this example, when *gecko_evt_hardware_soft_timer_id* event happens the target sends a user-defined payload.

```
/* soft timer fired event */
case gecko_evt_hardware_soft_timer_id:
   /* check which softtimer fired */
   if (evt->data.evt_hardware_soft_timer.handle == USER_EVENT_ID) {
   /* user event can be sent out to HOST */
   /* echo the last custom message from the HOST */
      gecko_send_evt_user_message_to_host(user_event_data_len, user_event_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.

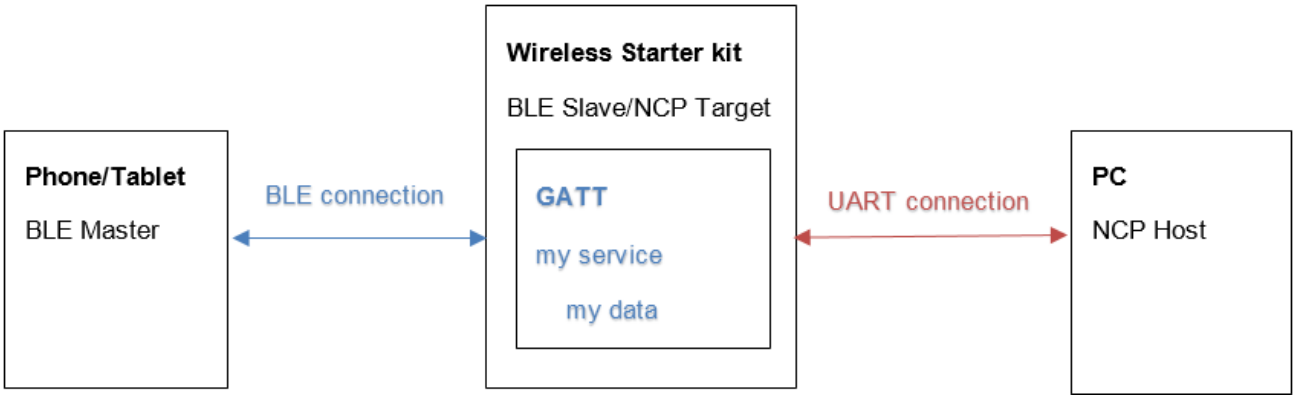
The AppLoader features and limitations are summarized below:

- 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 master device (tablet/phone) writes this characteristic, the slave (WSTK – NCP Target) will forward this data to the NCP Host. The NCP Host will print out the actual data to the PC console.



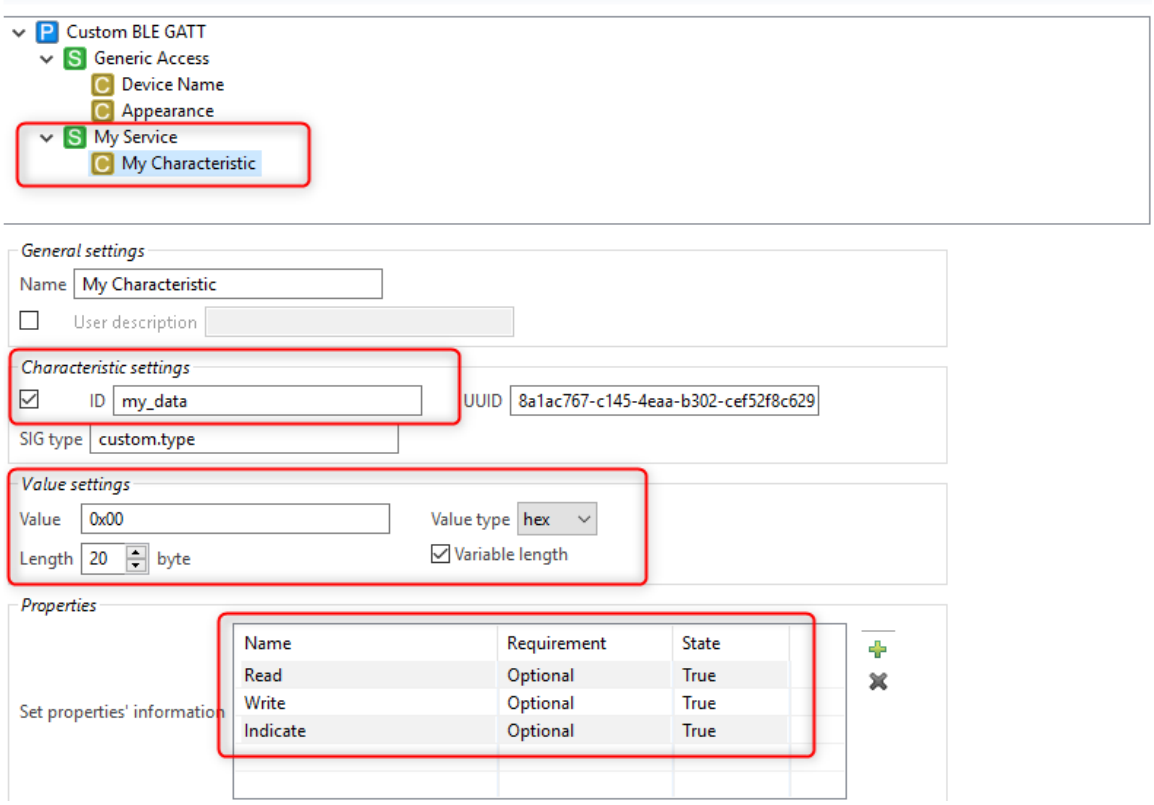
To implement this application, you need these changes:

- Add the new service and characteristics to the GATT
- Add the new GATT both to the Host and Target project
- Handle GATT change event (`gecko_evt_gatt_server_attribute_value`) in the Host project

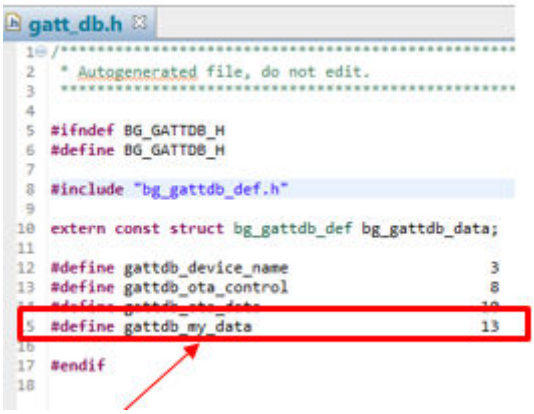
8.1 Update the Target Project

In projects using SDK 2.1.0 or later, the GATT database can be modified using the visual GATT editor in Simplicity Studio. For more details, refer to *QSG139: Bluetooth® Development with Simplicity Studio*.

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



Once the GATT is configured, save the .isc file and click [**Generate**]. 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: Please note that after making any changes to the GATT content you need to click [**Generate**] to update the *gatt_db.c* and *gatt_db.h* files. Then the project has to be re-compiled.

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:

   c:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.x\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../../../../app/bluetooth/examples_bgscript/wstk_bgapi \
-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)
{
  int i;
  for (i = 0; i < value->len; i++)
  {
    printf("my_data[%d] = 0x%x \r\n",i,value->data[i]);
  }
  printf("\r\n");
}
```

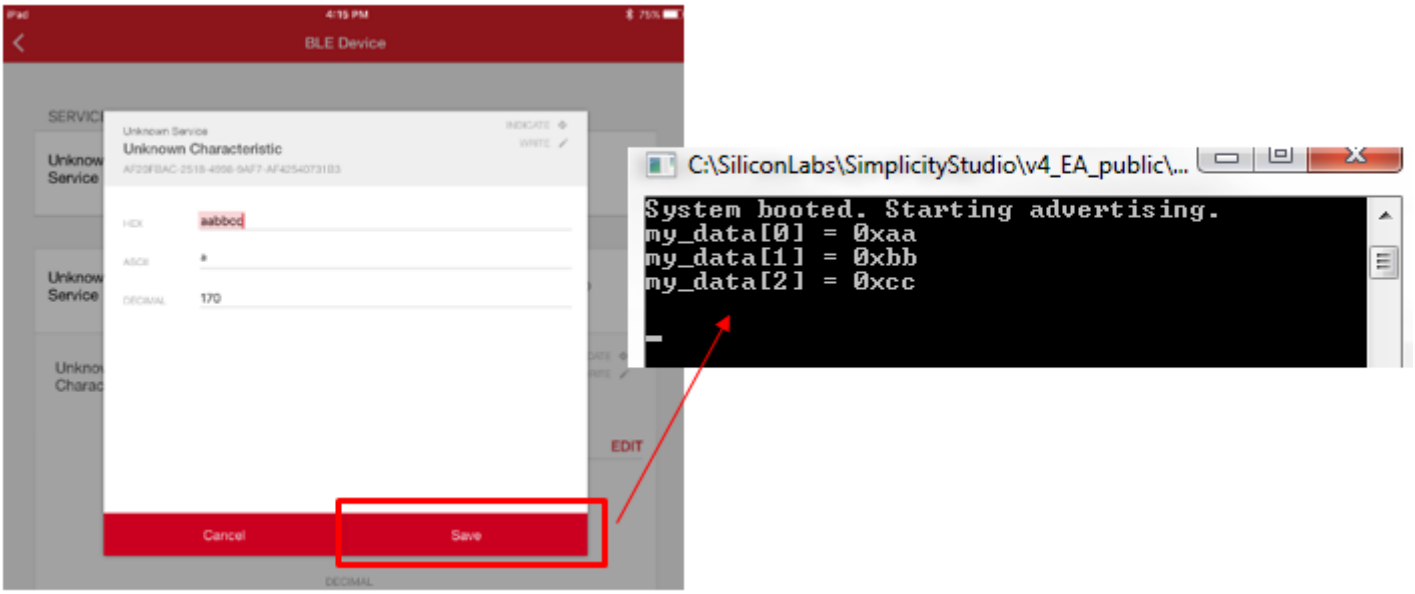
6. Now add the *gecko_evt_gatt_server_attribute* event to the switch case.

```
case gecko_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.

8.3 Testing

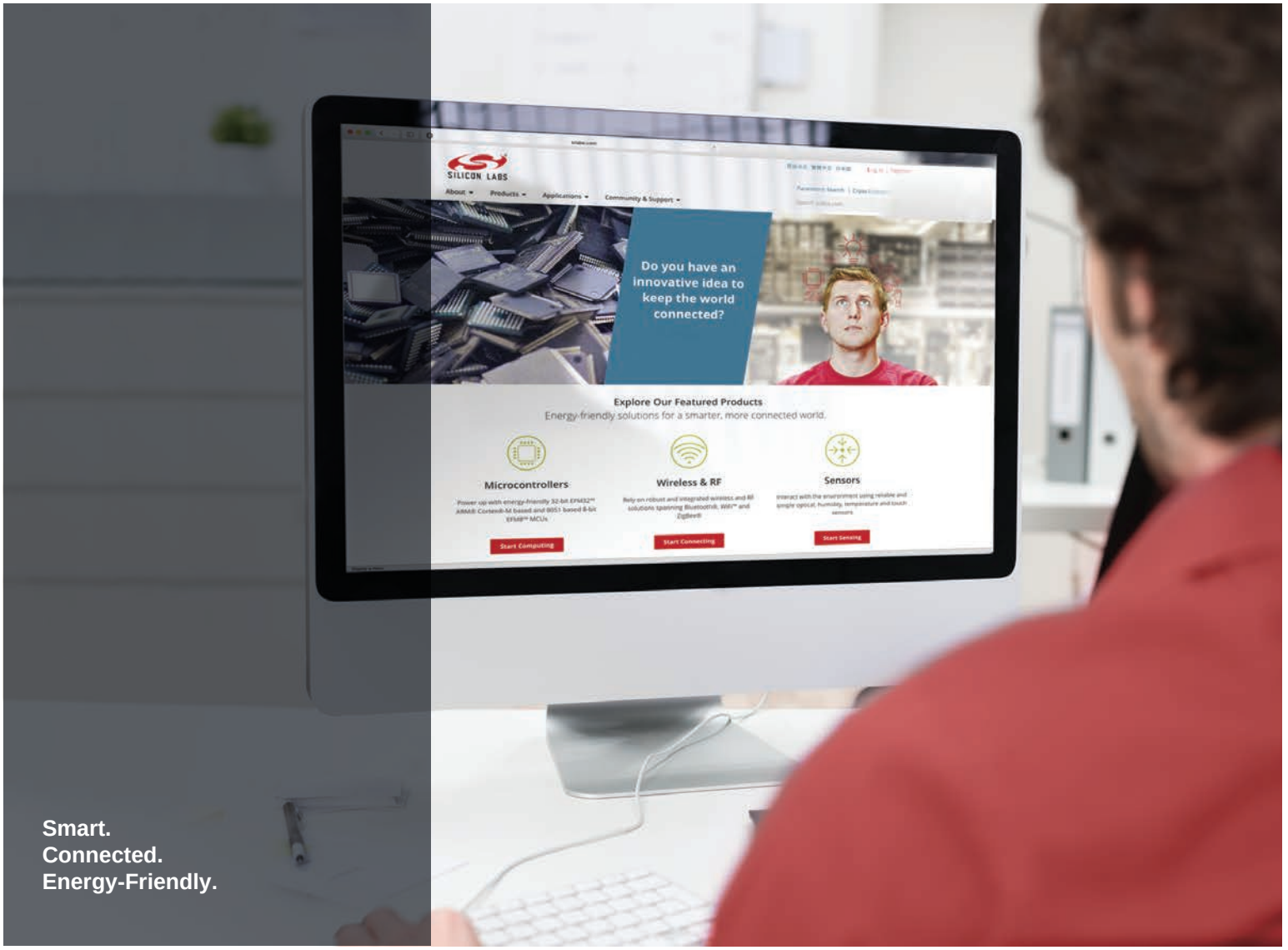
1. Start the host application from the *\exe* folder.
2. Once the PC is connected to WSTK (via UART), the WSTK starts advertising on Bluetooth.
3. If you connect via tablet/phone you can write the newly created *my_data* characteristic in the GATT. For this you can use the Blue Gecko app provided by Silicon Labs.
4. Browse to the *my_data* characteristic (*af20fbac-2518-4998-9af7-af42540731b3*) and write something to it. The data will be printed out by the host application.



9. Appendix: Empty NCP Host Application Example Source Code

```
/**************************************************************************************************//**
 * \file   app.c
 * \brief  Event handling and application code for Empty NCP Host application example
 ******************************************************************************************************
 * <b> (C) Copyright 2016 Silicon Labs, http://www.silabs.com</b>
 ******************************************************************************************************
 * This file is licensed under the Silabs License Agreement. See the file
 * "Silabs_License_Agreement.txt" for details. Before using this software for
 * any purpose, you must agree to the terms of that agreement.
 ******************************************************************************************************/

/* standard library headers */
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <stdbool.h>

/* BG stack headers */
#include "bg_types.h"
#include "gecko_bglib.h"

/* include the GATT handles*/
#include "gatt_db.h"
/* Own header */
#include "app.h"

void AttrValueChanged_my_data(uint8array *value);

/**************************************************************************************************//**
 *  \brief  Event handler function.
 *  \param[in] evt Event pointer.
 ******************************************************************************************************/
void appHandleEvents(struct gecko_cmd_packet *evt)
{
    if (NULL == evt)
    {
        return;
    }

    /* Handle events */
    switch (BGLIB_MSG_ID(evt->header))
    {
        case gecko_evt_system_boot_id:

            printf("System booted. Starting advertising. \n");

            /* Set advertising parameters. 100ms advertisement interval. All channels used.
            * The first two parameters are minimum and maximum advertising interval, both in
            * units of (milliseconds * 1.6). The third parameter '7' sets advertising on all channels. */
            gecko_cmd_le_gap_set_adv_parameters(160, 160, 7);

            /* Start general advertising and enable connections. */
            gecko_cmd_le_gap_set_mode(le_gap_general_discoverable, le_gap_undirected_connectable);

            break;
```

```
        case gecko_evt_le_connection_closed_id:

            /* Restart general advertising and re-enable connections after disconnection. */
            gecko_cmd_le_gap_set_mode(le_gap_general_discoverable, le_gap_undirected_connectable);

            break;
            /* Value of attribute changed from the local database by remote GATT client */

        case gecko_evt_gatt_server_attribute_value_id:
            /* Check if the event is because "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;

        default:
            break;
  }
}

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



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com