

# QSG169: *Bluetooth® SDK v3.x Quick-Start Guide*

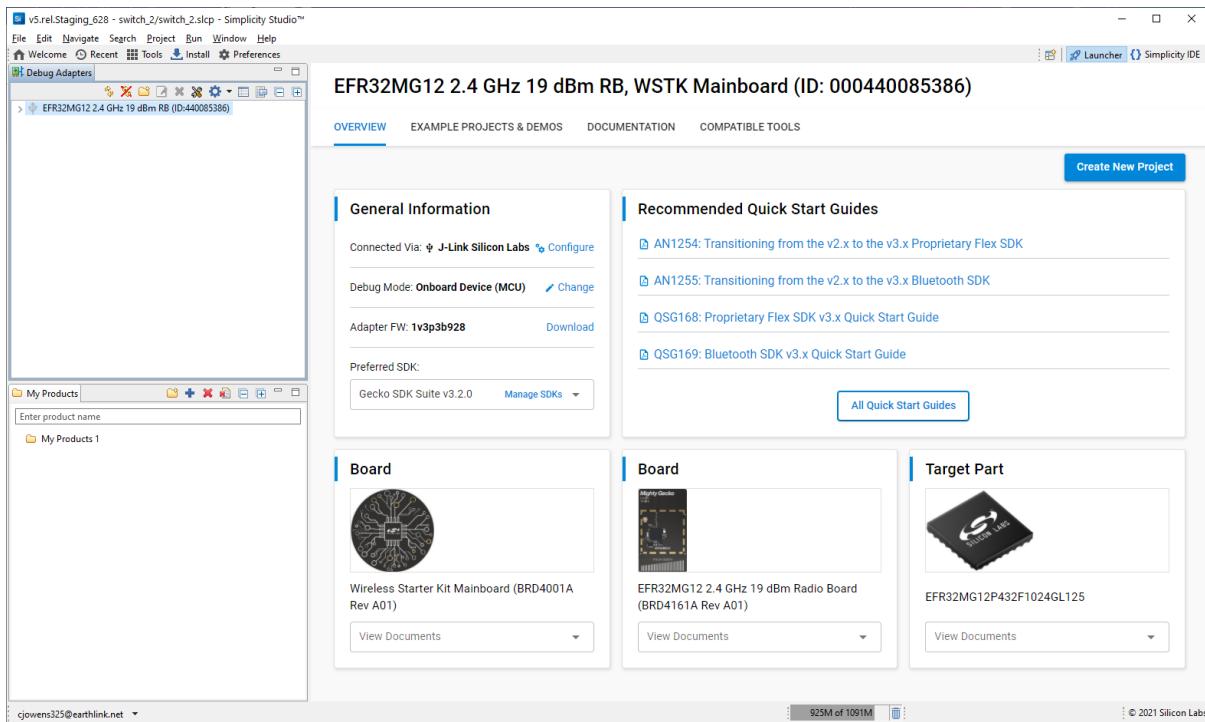


This document describes how to get started with Bluetooth development using the Bluetooth software development kit (SDK) v3.x and Simplicity Studio® 5 with a compatible wireless starter kit (WSTK). If you have purchased an EFR32BG WSTK you can first experiment with precompiled demos and an Android or iOS smartphone app before continuing with your own application development.

If you use Simplicity Studio 4 with Bluetooth SDK v2.x, find corresponding content in [QSG139: Bluetooth® SDK v2.x Quick Start Guide](#).

## KEY POINTS

- Introducing the Bluetooth development environment.
- Using the WSTK demos and Android or iOS smartphone app to demonstrate Bluetooth features.
- Starting application development for Bluetooth devices with Simplicity Studio.



# 1 Introduction

This document describes how to get started with Bluetooth development using Silicon Labs products. It introduces the features of the Silicon Labs Bluetooth stack v3.x and the resources available to help with development. Application development is started using the Silicon Labs development environment Simplicity Studio 5 and the Bluetooth Software Development Kit (SDK) v3.x. The SDK comes with a number of example application that you can then modify to create your own applications. If you are developing with an EFR32BG device and have purchased a EFR32BG Wireless Starter Kit (WSTK), you can use precompiled demos and an Android or iOS smartphone app to demonstrate Bluetooth software features.

This document describes the following:

- Bluetooth Stack features and components (see section [2 About the Bluetooth Stack](#))
- A description of the precompiled demos and example code available in the SDK (see section [3 About Demos and Examples](#))
- How to test prebuilt demo software with either an iOS or Android smartphone app (see section [4 Getting Started with Bluetooth Demo Software](#))
- How to develop your own applications in Simplicity Studio (see section [6 Starting Application Development](#))
- A description of other tools that are useful in the development process (see section [6 Development Tools](#))

## 1.1 Prerequisites

Before beginning application development you should have:

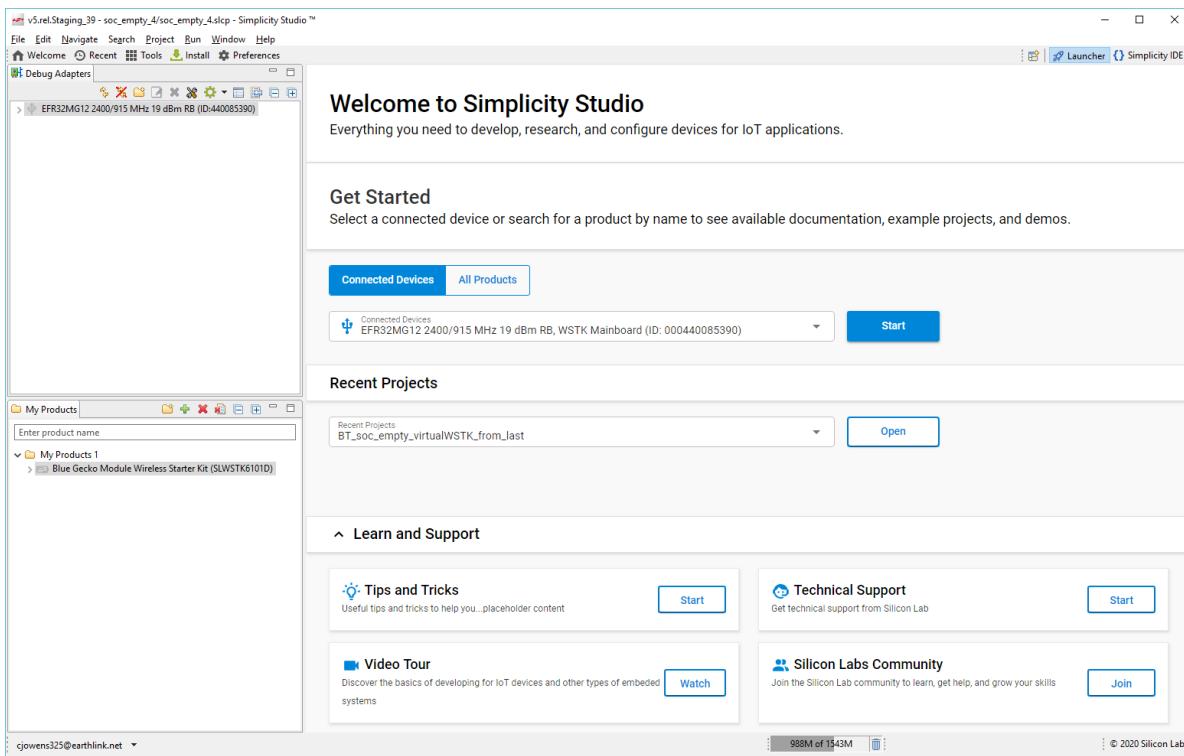
- Acquired a basic understanding of Bluetooth technology and terminology. *UG103.14: Bluetooth LE Fundamentals* provides a good starting point if you have not yet learned about Bluetooth.
- Purchased an EFR32BG Wireless Starter Kit or other compatible target hardware.
- Created an account at Silicon Labs. This is required in order to download the Silicon Labs Bluetooth SDK.
  - You can register at [https://siliconlabs.force.com/apex/SL\\_CommunitiesSelfReg?form=short](https://siliconlabs.force.com/apex/SL_CommunitiesSelfReg?form=short).
- Downloaded Simplicity Studio 5 and the Silicon Labs Bluetooth SDK and become generally familiar with the SSv5 Launcher perspective. SSv5 installation and getting started instructions along with a set of detailed references can be found in the online *Simplicity Studio 5 User's Guide*, available on <https://docs.silabs.com/> and through the SSv5 help menu.
- Obtained a compatible compiler (See the Bluetooth SDK's release notes for the compatible versions):
  - Simplicity Studio comes with a free GCC C-compiler.
  - IAR Embedded Workbench for ARM (IAR-EWARM) can also be used as the compiler for Silicon Labs Bluetooth projects. Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM.

To get a 30-day evaluation license for IAR-EWARM:

- Go to the Silicon Labs support portal at <https://www.silabs.com/support>.
- Scroll down to the bottom of the page, and click **Contact Support**
- If you are not already signed in, sign in.
- Click the Software Releases tab. In the View list select **Development Tools**. Click **Go**. In the results is a link to the IAR-EWARM version named in the release notes.
- Download the IAR package (takes approximately 1 hour).
- Install IAR.
- In the IAR License Wizard, click **Register with IAR Systems to get an evaluation license**.
- Complete the registration and IAR will provide a 30-day evaluation license.
- Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM.

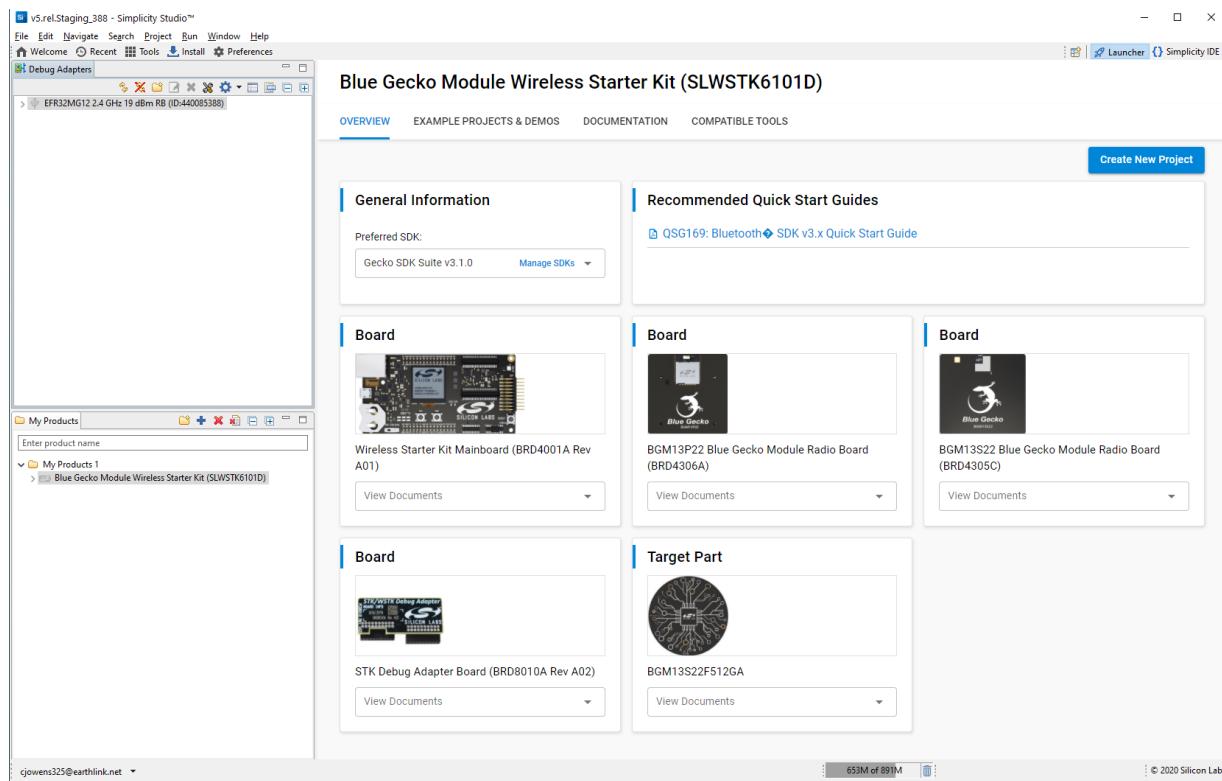
## 1.2 Support

You can access the Silicon Labs support portal at <https://www.silabs.com/support> through Simplicity Studio 5's Welcome view under Learn and Support. Use the support portal to contact Customer Support for any questions you might have during the development process.

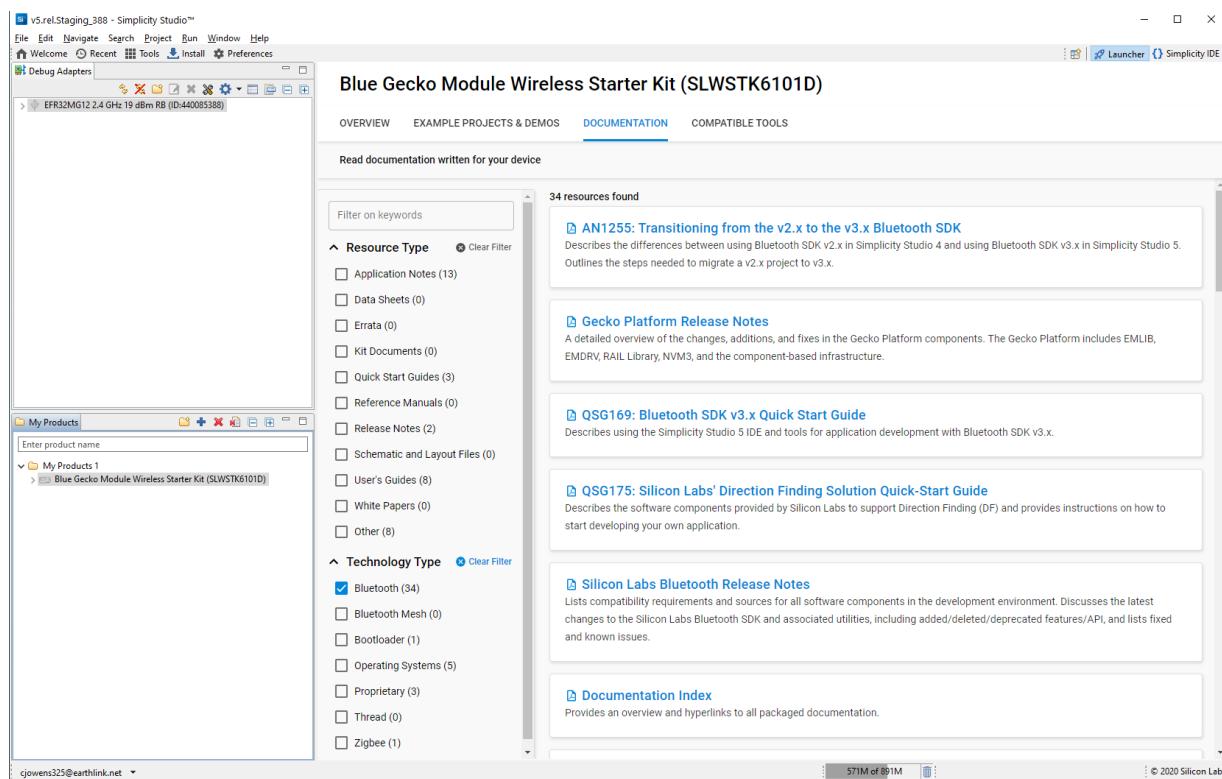


## 1.3 Documentation

Hardware-specific documentation may be accessed through links on the part Overview tab in Simplicity Studio 5.



SDK documentation, User's Guides, and other references are available through the Documentation tab.



The Bluetooth API reference is available online along with further useful documentation and code examples: <https://docs.silabs.com/bluetooth/latest/>

The screenshot shows a web browser window displaying the 'General Overview - v3.0 - Bluetooth' page from [docs.silabs.com/bluetooth/latest/](https://docs.silabs.com/bluetooth/latest/). The page title is 'General Overview'. The left sidebar contains a navigation menu with sections like 'Bluetooth', 'Firmware Upgrade', 'NCP', 'System & Performance', 'API Reference', 'Lab Manuals', and 'Code Examples'. The main content area includes a 'Release Notes | Downloads' link and a section titled 'About the Bluetooth Stack' with descriptive text and a diagram. The diagram, titled 'Wireless Gecko', illustrates the stack structure with layers: Application, BGAPI, GAP, SM, GATT, and ATT.

You are viewing documentation for version: 3.0 (latest) | 2.13 | Version History

## General Overview

[Release Notes](#) | [Downloads](#)

### About the Bluetooth Stack

The Silicon Labs Bluetooth stack is an advanced Bluetooth 5-compliant protocol stack implementing the Bluetooth low energy standard. It supports multiple connections, concurrent central, peripheral, broadcaster, and observer roles. The Silicon Labs Bluetooth stack is meant for Silicon Labs Wireless Gecko SoCs and modules.

The Silicon Labs Bluetooth stack provides multiple APIs for the developer to access the Bluetooth functionality. Two modes are supported:

1. Standalone mode, where both the Bluetooth stack and the application run in a Wireless Geckos SoC or module. The application can be developed with C programming language.

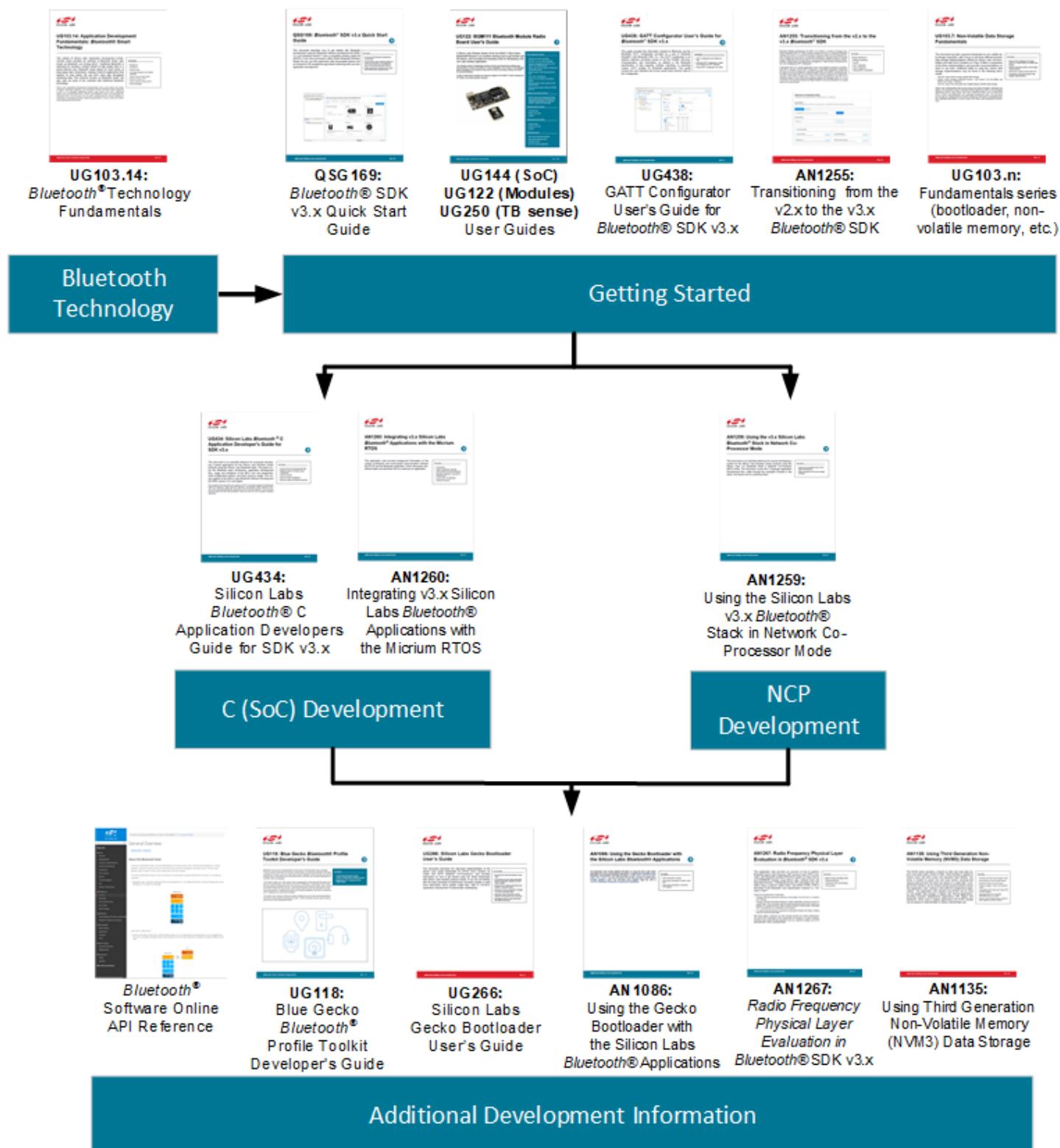
**Wireless Gecko**

```
graph TD; Application[Application] --- BGAPI[BGAPI]; BGAPI --- GAP[GAP]; BGAPI --- SM[SM]; BGAPI --- GATT[GATT]; BGAPI --- ATT[ATT];
```

The diagram shows the 'Wireless Gecko' stack structure. At the top is the 'Application' layer, followed by the 'BGAPI' layer. Below the BGAPI layer are four protocol layers: 'GAP', 'SM', 'GATT', and 'ATT', arranged vertically.

Training materials are available on the Silicon Labs website at <https://www.silabs.com/support/training/bluetooth>.

Key documentation for the Bluetooth SDK is summarized in the following figure.



#### 1.4 Gecko Platform

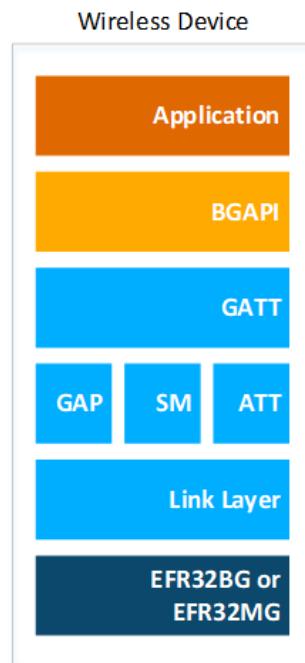
The Gecko Platform is a set of drivers and other lower layer features that interact directly with Silicon Labs chips and modules. Gecko Platform components include EMLIB, EMDRV, RAIL Library, NVM3, and mbedTLS. For more information about Gecko Platform, see release notes that can be found in Simplicity Studio's Documentation tab.

## 2 About the Bluetooth Stack

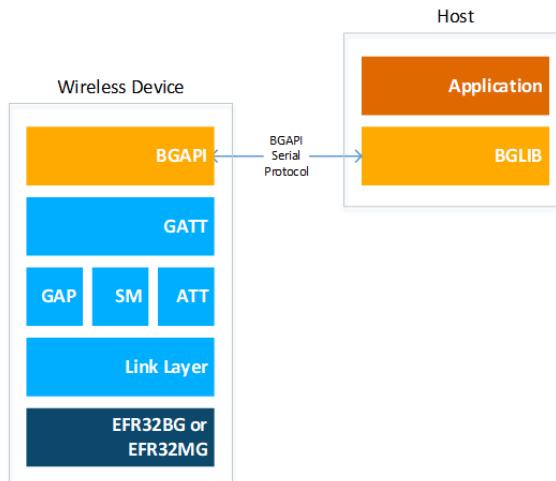
The v3.x Silicon Labs Bluetooth stack is an advanced Bluetooth 5-compliant protocol stack implementing the Bluetooth low energy standard. It supports multiple connections, concurrent central, peripheral, broadcaster, and observer roles. The v3.x Silicon Labs Bluetooth stack is meant for Silicon Labs EFR32 SoCs and modules.

The Silicon Labs Bluetooth stack provides multiple APIs for the developer to access the Bluetooth functionality. Three modes are supported:

1. Standalone mode, where both the Bluetooth stack and the application run in an EFR32SoC or module. The application can be developed with C programming language.

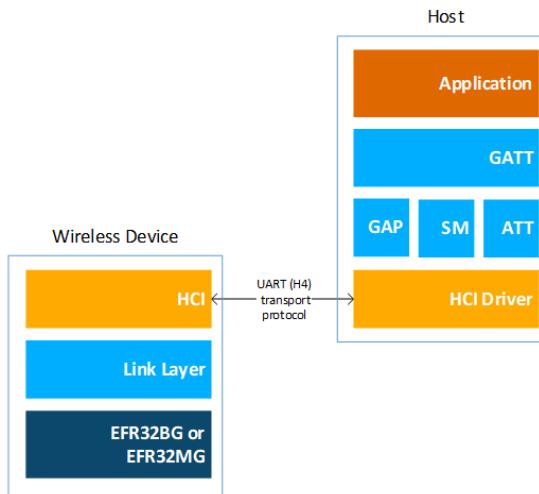


2. Network Co-Processor (NCP) mode, where the Bluetooth stack runs in an EFR32 and the application runs on a separate host MCU. For this use case, the Bluetooth stack can be configured into NCP mode where the API is exposed over a serial interface such as UART.



3. Radio Co-Processor (RCP) mode, where only the Link Layer of the Bluetooth stack runs on the EFR32, and the Host Layer of the stack, as well as the application, runs on a separate host MCU or PC. In this use case, the Host Layer is developed by a third party, since Silicon Labs' Bluetooth stack is only built for EFR32 SoCs / modules. The Link Layer and the host layer communicate via HCI

(Host-Controller Interface), which is a standard interface between the two layers. The HCI can be accessed via UART following the Bluetooth SIG's UART (H4) transport protocol.



## 2.1 Bluetooth Stack Features

The features of the Silicon Labs Bluetooth stack are listed in the following table.

Feature	Value and comments
Bluetooth version	Bluetooth 5.2
Bluetooth features	Bluetooth 5.2 GATT caching
"	Bluetooth 5 2M PHY (EFR32[B M]G12, EFR32[B M]G13, EFR32[B M]G21, and EFR32[B M]G22)
"	Bluetooth 5 LE Long Range (EFR32[B M]G13, EFR32[B M]G21, and EFR32[B M]G22)
"	Bluetooth 5 advertisement sets and scan event reporting
"	Bluetooth 5 extended advertisements (EFR32[B M]G12, EFR32[B M]G13, EFR32[B M]G21, and EFR32[B M]G22): Anonymous advertisement, Periodic advertisement, and Extended advertisement packet size: up to 1650 B
"	Concurrent central, peripheral, broadcaster and observer modes LE secure connections
"	LE Privacy 1.2 (peripheral) LE packet length extensions LE dual topology
"	Whitelisting (central side only)
"	LE Power Control
Simultaneous connections	Up to 32 simultaneous connections regardless of role (master or slave)
Maximum throughput	1300 kbps over 2M PHY
"	700 kbps over 1M PHY
Encryption	AES-128
Pairing modes	Just works
"	Man-in-the-Middle with numeric comparison and passkey Out-Of-Band
Number of simultaneous bondings	EFR32[B M]G1, EFR32[B M]G12 and EFR32[B M]G13: Up to 13 when using PS, up to 32 with NVM3
"	EFR32[B M]G21, EFR32[B M]G22: Up to 32
Link Layer packet size	Up to 251 B

Feature	Value and comments
ATT protocol packet size	Up to 250 B
Supported Bluetooth profiles and services	All GATT based profiles and services are supported
Apple HomeKit	Apple HomeKit R15-compliant implementation (EFR32[B M]12, EFR32[B M]13, EFR32[B M]21, and EFR32[B M]G22)
"	Implements all Apple HomeKit profiles and services Available separately for Apple MFi licensees
Host (NCP/RCP) interfaces	4-wire UART with RTS/CTS control or 2-wire UART without RTS/CTS
"	GPIOs for sleep and wake-up management
Wi-Fi Coexistence	Using Packet Trace Arbitration (PTA)
Bootloaders	Secure Gecko Bootloader supporting authenticated and encrypted updates over OTA or UART and Secure Boot. The Gecko Bootloader also supports flash partitioning and both internal and external (SPI) flash.
Non-volatile memory	EFR32[B M]G1, EFR32[B M]G12, EFR32[B M]G13: NVM3 or Persistent Store (PS)*
"	EFR32[B M]G21, EFR32[B M]G22: NVM3

\* Example applications in the SDK that are generated for these platforms will use PS by default.

## 2.2 Bluetooth Qualification

All products using Bluetooth technology must go through the Bluetooth SIG's Qualification Process, even if the product does not have the Bluetooth logo or Bluetooth is not mentioned in the packaging and the documentation. In practice this means that, before you can sell a Bluetooth-enabled product to the market, the product must be qualified as an End Product through the Bluetooth SIG. The qualification listing has a [Declaration Fee](#). There are online resources to learn more about the [Bluetooth Qualification Process](#) as well as tutorials on the [Launch Studio](#), which is the online tool used to complete the Bluetooth Qualification Process. If you need assistance to qualify your device consider reaching out to your nearest [Bluetooth Qualification Consultant](#).

When qualifying your end-product based on the Silicon Labs Bluetooth stack, you will integrate the pre-qualified components listed in the table below, depending on which SDK version was used to build your application.

Bluetooth SDK version	Component	QDID
v2.10.x up to 2.12.x	Link Layer (Bluetooth 5.1)	Launch Studio Listing Details (2)
"	Host stack (Bluetooth 5.1) (1)	Launch Studio Listing Details (3)
v2.13.x and above; v3.0.x and above	Link Layer (Bluetooth 5.2)	Launch Studio Listing Details (4)
"	Host stack (Bluetooth 5.2)	Launch Studio Listing Details (5)

(1) Launch Studio: For end-product qualifications based on this component, you must check the items 21/19 and 34/15 and uncheck the items 21/17 and 34/13 in the "Summary ICS" and upload the Erratum 11838 Test Evidences, which can be requested through the support portal.

(2) [127618](#)

(3) [126252](#)

(4) [147971](#)

(5) [146950](#)

**Note:** According to Bluetooth SIG Qualification [Program Reference Document \(PRD\)](#), the Assessment Date of the tested Component must be less than three years old at the time it is being imported into a Launch Studio project for a new End Product Listing (EPLs). After the expiration of a Component QDID (Qualified Design ID), a newer SDK version than the one used for the outdated QDID should be used in order to qualify your product. There can be also newer QDIDs than the ones listed in the table above if there are newer Component versions. You can browse our valid Qualified Components and their Assessment Date by inserting Silicon Laboratories in the search bar of [Launch Studio](#). Contact the technical support in case there is a need to use an older SDK version.

The above software-based pre-qualified components are two out of the three components to integrate when proceeding with the "[Qualification Process with Required Testing](#)". Despite the "Required Testing", customers do not need to do any additional testing, given that the test reports are embedded in the pre-qualified components for the SIG to review.

In addition to these two software components you must also have integrated a qualified RF-PHY component in your end-product listing. If you are designing with one of the Silicon Labs Bluetooth modules then refer to the module datasheet for the appropriate component QDID to use. If you are designing with an SoC then you may need to obtain your own RF-PHY qualification with the Bluetooth SIG, depending on your hardware design. In the latter case, consult your nearest [Bluetooth Qualification Consultant](#), or Silicon Labs through the support portal, to understand if an existing Silicon Labs RF-PHY pre-qualification could be used.

Silicon Labs does not provide prequalified profiles. Customers must provide these with their end applications that implement the functionality as per the SIG profile specification.

## 2.3 The Bluetooth Stack APIs

This section briefly describes the different software APIs available for the developer when developing a Bluetooth application either in SoC or NCP mode. In RCP mode the standard HCI is used, which is defined in the Bluetooth Core Specification and therefore is not discussed here.

### 2.3.1 The BGAPI Bluetooth API

The BGAPI is the Bluetooth API provided by the Silicon Labs Bluetooth stack. It provides access to all the Bluetooth functionality implemented by the Bluetooth stack, such as: the Generic Access Profile (GAP), connection manager, the security manager (SM), and GATT client and server.

In addition to the Bluetooth APIs, the BGAPI also provides access to a few other functions like the Direct Test Mode (DTM) API for RF testing purposes, the NVM (Non-Volatile Memory) API for reading and writing settings to and from the devices flash memory, the DFU (Device Firmware Update) API for field firmware updates, and the System API for various system level functions.

### 2.3.2 CMSIS and emlib

The Cortex Microcontroller Software Interface Standard (CMSIS) is a common coding standard for all ARM Cortex devices. The CMSIS library provided by Silicon Labs contains header files, defines (for peripherals, registers and bitfields), and startup files for all devices. In addition, CMSIS includes functions that are common to all Cortex devices, like interrupt handling, intrinsic functions, etc. Although it is possible to write to registers using hard-coded address and data values, it is recommended to use the defines to ensure portability and readability of the code.

To simplify programming Wireless Geckos, Silicon Labs developed and maintains a complete C function library called emlib that provides efficient, clear, and robust access to and control of all peripherals and core functions in the device. This library resides within the em\_xxx.c (for example, em\_dac.c) and em\_xxx.h files in the SDK.

The emlib documentation is available on <https://docs.silabs.com>.

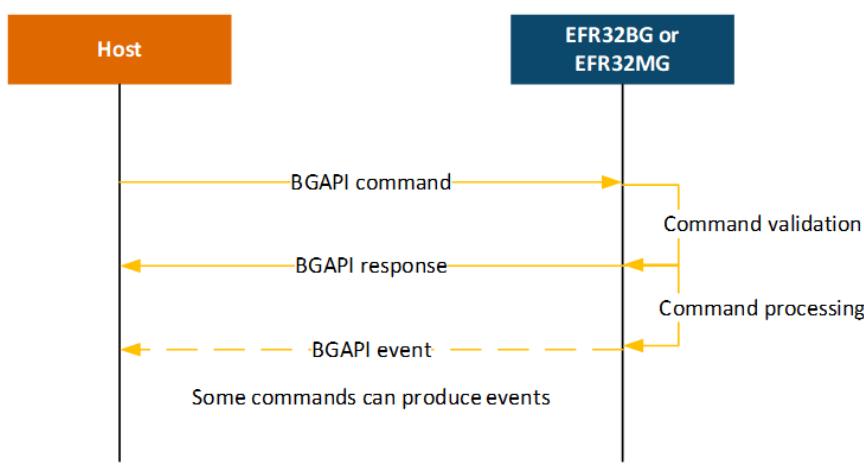
### 2.3.3 The BGAPI Serial Protocol and BGLIB Host API

When configured in NCP (network co-processor) mode, the Bluetooth stack also implements the BGAPI serial protocol. This allows the Bluetooth stack to be controlled over a serial interface such as UART from a separate host like an EFM32 microcontroller. The BGAPI serial protocol provides exactly the same Bluetooth APIs over UART as the BGAPI API when used in a standalone mode. Additionally, an extra command and an event are reserved for user messaging in case the interface should be extended with custom commands.

The BGAPI serial protocol is a lightweight, binary protocol that carries the BGAPI commands from the host to the Bluetooth stack and responses and events from the Bluetooth stack back to the host.

The Bluetooth SDK delivers a ready-made BGAPI serial protocol parser implementation, called BGLIB. It implements the serial protocol parser and C language function and events for all the APIs provided by the Bluetooth stack. The host code developed on top of BGLIB can be written to be identical to the code for the Wireless Gecko, which allows easy porting of the application code from the Wireless Gecko to a separate host or vice versa.

A Python based BGAPI serial protocol parser is also available here: <https://pypi.org/project/pybgapi/>

**Figure 2.1. BGAPI Serial Protocol Message Exchange**

The BGAPI serial protocol packet structure is described in the following table.

**Table 2-1. BGAPI Packet Structure**

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4-255
<b>Explanation</b>	Message type	Minimum payload length	Message class	Message ID	Payload
<b>Values</b>	0x20: command	0x00 - 0xFF	0x00 - 0xFF	0x00 - 0xFF	Specific to command, response, or event
"	0x20: response	0x00 - 0xFF	0x00 - 0xFF	0x00 - 0xFF	Specific to command, response, or event
"	0xA0: event	0x00 - 0xFF	0x00 - 0xFF	0x00 - 0xFF	Specific to command, response, or event

### 2.3.4 GATT Configuration

Bluetooth applications usually need a GATT database. The structure of the GATT database can be defined in the Bluetooth application. The Silicon Labs' Bluetooth SDK provides two ways to define the GATT database:

- A static GATT database can be defined in compile time with the appropriate tools provided by the Bluetooth SDK. In this case the database structure is stored in the ROM, which means faster start-up time and lower memory usage.
- A dynamic GATT database can be defined in runtime with the appropriate BGAPI commands. In this case the database structure is stored in the RAM, which makes it more flexible. This is recommended in the NCP use case to avoid re-building the target code that runs on the Wireless Gecko.

#### The Bluetooth Profile Toolkit GATT Builder

The Bluetooth Profile Toolkit is a simple XML-based API and description language used to describe (static) GATT-based services and characteristics easily without the need to write them in code. The XML files can be easily written by hand based on the information contained in *UG118: Blue Gecko Bluetooth® Profile Toolkit Developer Guide*. Use the Profile Toolkit GATT Builder if you are developing outside of Simplicity Studio, and follow the instructions of *UG118: Blue Gecko Bluetooth® Profile Toolkit Developer Guide* to convert your GATT database into C code.

#### The GATT Configurator

Simplicity Studio includes the GATT Configurator, a tool that allows building the (static) GATT database in a visual way, without hand editing the XML file. It also automatically converts the database structure into C code upon saving. See section [6.1, The GATT Configurator](#) for summary information, and *UG438: GATT Configurator User's Guide for Bluetooth SDK v3.x* for details. Open the GATT

Configurator in Simplicity Studio through the Project Configurator, Configuration Tools tab. Click **Open** and the GATT Configurator tool opens the file *gatt\_configuration.btconf* in a new tab.

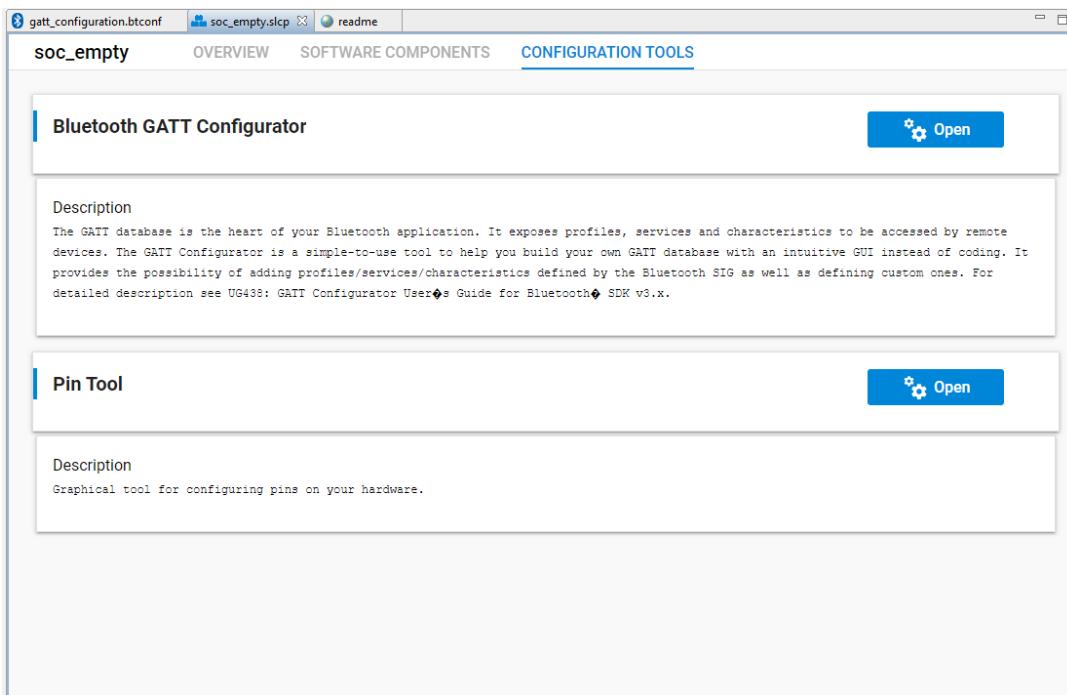


Figure 2.2. Opening the Bluetooth GATT Configurator

*gatt\_configuration.btconf* gives the trunk of the GATT database. It is located inside the *config > btconf* directory of your project. This file can be edited using the GATT Configurator.

The contents of the additional xml files in the *config > btconf* folder will appear as *Contributed Items* in the GATT Configurator UI. See for example the *ota-dfu.xml* file provided by the OTA DFU software component. These files cannot be edited by the GATT Configurator.

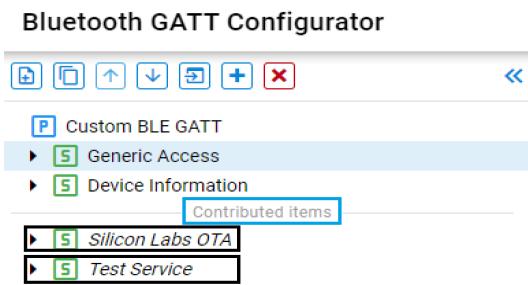


Figure 2.3. Contributed Items in the GATT Configurator UI

Upon saving `gatt_configuration.btconf`, the GATT database developed with the GATT Configurator is converted to a .c file and an .h file and included in the application project as a pre-build step when the firmware is compiled. Then the GATT can be accessed with the Bluetooth stack GATT APIs or by a remote Bluetooth device.

```
<gatt>

    <service uuid="1800">
        <description>Generic Access Service</description>

        <characteristic uuid="2a00">
            <properties read="true" const="true" />
            <value>Blue Gecko BGMI11</value>
        </characteristic>

        <characteristic uuid="2a01">
            <properties read="true" const="true" />
            <value type="hex">0768</value>
        </characteristic>

    </service>

</gatt>
```

**Figure 2.4. Example of a Generic Access Service**

### Building a Dynamic GATT database

The GATT database can also be built dynamically from the application using the GATT database API class of the Bluetooth API if the Dynamic GATT Database software component is installed in your project, or if this API class is explicitly initialized. For more information see the Bluetooth API reference manual at <https://docs.silabs.com/bluetooth/latest/> and the corresponding section of *UG438: GATT Configurator User's Guide for Bluetooth SDK v3.x*.

## 2.4 About the Bluetooth SDK

The Bluetooth SDK is a full software development kit that enables you to develop applications on top of the Bluetooth stack using C programming language. The SDK also supports making standalone applications, where the Bluetooth stack and the application both run in the Wireless Gecko, or the network co-processor (NCP) architecture, where the application runs on an external host and the Bluetooth stack runs in the Wireless Gecko. SDK contents and folder structure are described in the following sections.

### 2.4.1 Libraries

The following libraries are delivered with the Bluetooth SDK and must be included in C application projects.

Library	Explanation	Mandatory
<code>libbluetooth.a</code>	Bluetooth stack library	Yes
<code>libraill_efr32xg1_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG1 platform
<code>libraill_efr32xg12_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG12 platform
<code>libraill_efr32xg13_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG13 platform
<code>libraill_efr32xg14_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG14 platform
<code>libraill_efr32xg21_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG21 platform
<code>libraill_efr32xg22_gcc_release.a</code>	RAIL library for GCC	Yes for GCC projects on EFR32xG22 platform
<code>libraill_efr32xg1_iar_release.a</code>	RAIL library for IAR	Yes for IAR projects on EFR32xG1 platform
<code>libraill_efr32xg12_iar_release.a</code>	RAIL library for IAR	Yes for IAR projects on EFR32xG12 platform
<code>libraill_efr32xg13_iar_release.a</code>	RAIL library for IAR	Yes for IAR projects on EFR32xG13 platform

Library	Explanation	Mandatory
libraill_efr32xg14_iar_release.a	RAIL library for IAR	Yes for IAR projects on EFR32xG14 platform
libraill_efr32xg21_iar_release.a	RAIL library for IAR	Yes for IAR projects on EFR32xG21 platform
libraill_efr32xg22_iar_release.a	RAIL library for IAR	Yes for IAR projects on EFR32xG22 platform
libpsstore.a	PSStore library	Yes, on series 1
binapploader.o	Apploader for OTA updates	No
libcoex.a	Wi-Fi and Bluetooth coexistence	No
libnvm3_CM33_gcc.a / libnvm3_CM33_iar.a	-	Yes, on series 2

## 2.4.2 Include Files

The following files are delivered with the Bluetooth SDK and must be included in C application projects.

Library	Explanation	When needed
bg_gattdb_def.h	Bluetooth GATT database structure definition.	Included automatically.
sl_bt_version.h	Bluetooth stack version in plain text. The boot event reports the same version values as this file has.	For convenient access to Bluetooth SDK version Information. Not mandatory for application development.
sl_bt_ll_config.h	Bluetooth Link Layer configuration data type definitions. Included by sl_bt_stack_config.h.	Included automatically.
sl_bt_stack_config.h	Bluetooth stack configuration data type definitions. Included by sl_bluetooth_config.h.	Included automatically.
sl_bluetooth_config.h	Bluetooth configuration.	Included automatically if application is generated with the Project Configurator.
sl_bt_types.h	Bluetooth API data type definitions.	Included automatically.
sl_bt_stack_init.h	Bluetooth feature and API initialization functions on SoC.	Included automatically if application is generated with the Project Configurator.
sl_bt_api.h	Bluetooth API declarations with comprehensive documentation. This is the single file for Bluetooth API in SoC or NCP mode.	Included automatically if application is generated with the Project Configurator.
sl_bt_api.h	Bluetooth API library in plain source code for NCP host applications.	Included automatically if application is generated with the Project Configurator.
sl_bt_ncp_host_api.c	Bluetooth API library in plain source code for NCP host applications.	Included automatically if application is generated with the Project Configurator.
sl_bt_ncp_host.h	An adaptation layer between host application and Bluetooth API serial protocol.	Included automatically if application is generated with the Project Configurator.
sl_bt_ncp_host.c	An adaptation layer between host application and Bluetooth API serial protocol.	Included automatically if application is generated with the Project Configurator.
sl_bt_rtos_adaptation.h	An adaptation layer for running Bluetooth in Micrium OS on SoC.	Included automatically if application is generated with the Project Configurator.
sl_bt_rtos_adaptation.c	An adaptation layer for running Bluetooth in Micrium OS on SoC.	Included automatically if application is generated with the Project Configurator.

### 2.4.3 Platform Components

The following components are delivered with the Bluetooth SDK. The platform components are under the platform folder.

Folder	Explanation
bootloader	Gecko Bootloader source code and project files.
CMSIS	Silicon Laboratories CMSIS-CORE device headers. (1)
common	Silicon Labs status codes
Device	EFR32BG and EFR32MG device files. (2)
emdrv	A set of function-specific high-performance drivers for EFR32 on-chip peripherals. Drivers are typically DMA based and are using all available low-energy features. For most drivers, the API offers both synchronous and asynchronous functions. (3)
emlib	A low-level peripheral support library that provides a unified API for all EFM32, EZR32 and EFR32 MCUs and SoCs from Silicon Laboratories. (4)
Halconfig	Peripheral configuration
Hwconf_data	Gather chip-specific hardware configuration
micrium_os	Micrium OS
middleware	Display driver for WSTK development kits (5)
radio	Silicon Labs RAIL (Radio Abstraction Interface Layer) library
service	Sleep timer driver and configuration file. Used by the Bluetooth LE stack.

(1) <https://docs.silabs.com/mcu/latest/bgm21/group-Parts>

(2) <https://docs.silabs.com/mcu/latest/bgm21/group-Parts>

(3) <https://docs.silabs.com/mcu/latest/bgm21/group-emdrv>

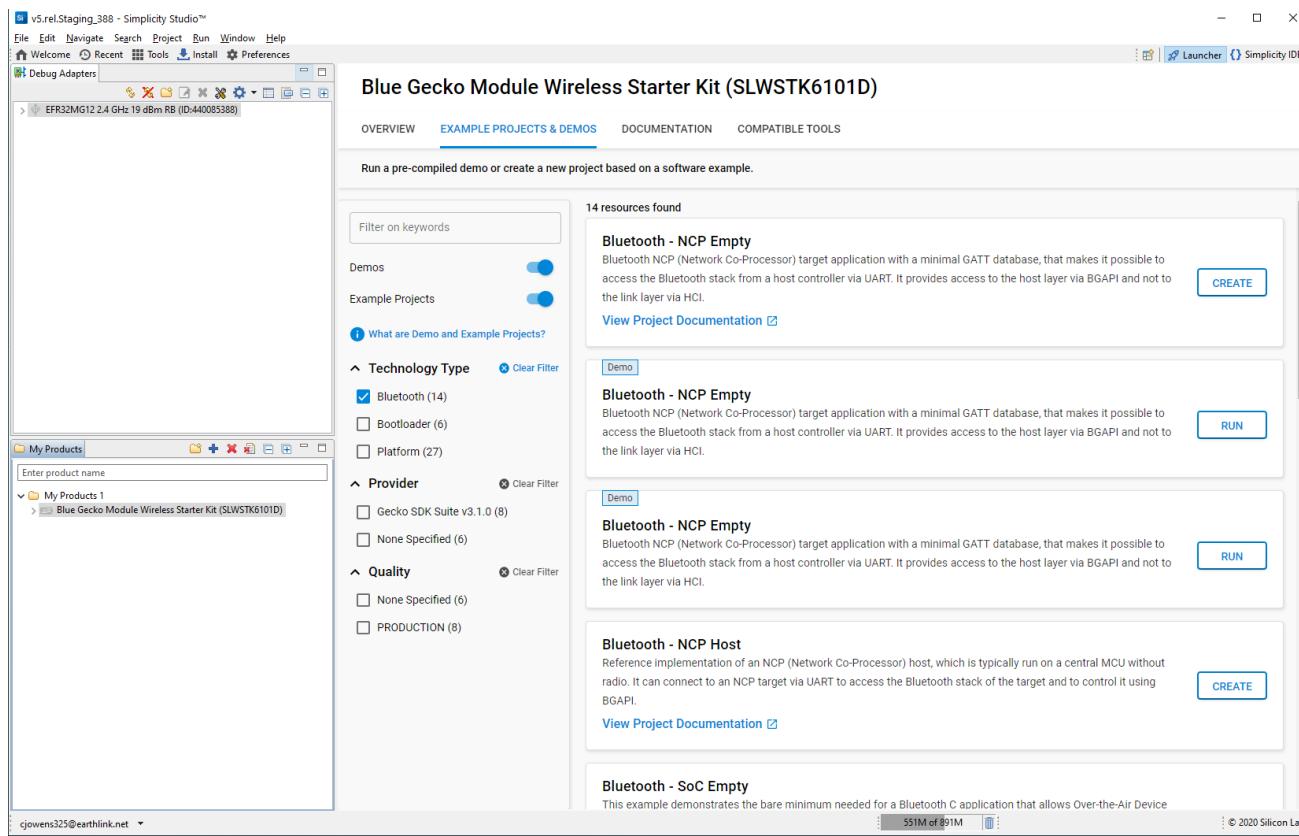
(4) <https://docs.silabs.com/mcu/latest/bgm21/group-emlib>

(5) <https://docs.silabs.com/mcu/latest/bgm21/group-emlib>

### 3 About Demos and Examples

Because starting application development from scratch is difficult, the Bluetooth SDK comes with a number of built-in demos and examples covering the most frequent use cases, as shown in the following figure. Demos are pre-built application images that you can run immediately. Software examples (example projects) can be modified before building the application image. Demos with the same name as Software examples are built from their respective example. Click **View Project Documentation** to see additional information about some examples. This is also displayed on a **readme** tab when you create a project based on the example.

Use the **Demos** and **Example Projects** switches to filter on only examples or only demos. Demos are also noted by the blue Demo tag in the upper left of the card.



**Note:** The demos and examples you see are determined by the part selected. If you are using a custom solution with more than one part, click on the part you are working with to see only the items applicable to that part.

To download and run a demo on your device, click **RUN** on the desired demo card. The demo automatically downloads to the selected device. See section [4 Getting Started with Bluetooth Demo Software](#) for more information about testing the demos.

Software examples include a configurable GATT database and configurable components. See section [5 Starting Application Development](#).

#### 3.1 Demo/Example Descriptions

The following examples are provided. Examples with (\*) in their names have a matching pre-built demo.

- **Silicon Labs Gecko Bootloader examples** (see [UG266: Silicon Labs Gecko Bootloader User Guide](#) and [AN1086: Using the Gecko Bootloader with Silicon Labs Bluetooth Applications](#))
- **Bluetooth Examples**
  - **Bluetooth – RCP**: Radio Co-Processor (RCP) target application. Runs the Bluetooth Controller (i.e. the Link Layer only) and provides access to it using the standard HCI (Host-Controller Interface) over a UART connection.
  - **Bluetooth – NCP(\*)**: Network Co-Processor (NCP) target application. Runs the full Bluetooth stack and provides access to it by exposing the Bluetooth API (BGAPI) via UART connection. NCP mode makes it possible to run your application on a host controller

or PC. This example does not have a GATT database, but makes it possible to build one from the application using the Dynamic GATT API.

- **Bluetooth – NCP Empty(\*)**: Network Co-Processor (NCP) target application. Runs the full Bluetooth stack and provides access to it by exposing the Bluetooth API (BGAPI) via UART connection. NCP mode makes it possible to run your application on a host controller or PC. This example contains a minimal GATT database, and cannot be used with host applications that use the Dynamic GATT API.
- **Bluetooth – NCP Host**: Reference implementation of an NCP (Network Co-Processor) host, which typically runs on a central MCU without radio. It can connect to an NCP target via UART to access the Bluetooth stack of the target and to control it using BGAPI. This example uses the Dynamic GATT feature.
- **Bluetooth – NCP AoA Locator(\*)**: Network Co-Processor (NCP) target application extended with CTE Receiver support. It enables Angle of Arrival (AoA) calculation. Use this application with Direction Finding host examples.
- **Bluetooth – SoC AoA Asset Tag(\*)**: Demonstrates a CTE (Constant Tone Extension) transmitter that can be used as an asset tag in a Direction Finding setup estimating Angle of Arrival (AoA).
- **Bluetooth – SoC Blinky(\*)**: The classic blinky example using Bluetooth communication. From the EFR Connect mobile app, the LED controller button toggles LED0 on the board. In addition, pressing or releasing PB0 on the mainboard notifies the app. This is a demonstration of a simple two-way data exchange over GATT.
- **Bluetooth – SoC DTM**: This example implements the direct test mode (DTM) application for radio testing. DTM commands can be called via UART. See [AN1267: Radio Frequency Physical Layer Evaluation in Bluetooth® SDK v3.x](#) for more information.
- **Bluetooth – SoC Empty**: A minimal project structure that serves as a starting point for custom Bluetooth applications. It contains a minimal GATT database that can be expanded to fit your application requirements. The application starts advertising after boot and restarts advertising after a connection is closed. It also supports Over-the-Air Device Firmware Upgrade (OTA DFU)
- **Bluetooth – SoC Interoperability Test**: A test procedure containing several test cases for Bluetooth Low Energy communication. This demo is meant to be used with the EFR Connect mobile app, through the "Interoperability Test" tile on the Develop view of the app.
- **Bluetooth – SoC Thermometer(\*)**: Implements a GATT Server with the Health Thermometer Profile, which enables a Client device to connect and get temperature data. Temperature is read from the Si7021 digital relative humidity and temperature sensor of the WSTK or of the Thunderboard.
- **Bluetooth – SoC Thermometer Client**: Implements a GATT Client that discovers and connects with up to four Bluetooth LE devices advertising themselves as Thermometer Servers. It displays the discovery process and the temperature values received via UART.

**Note:** Some radio boards will exhibit random pixels in the display when this example is running because they have a shared pin for sensor- and display-enabled signals.

- **Bluetooth – SoC Thermometer FreeRTOS**: Demonstrates the integration of FreeRTOS into Bluetooth applications. RTOS is added to the Bluetooth - SoC Thermometer sample app.
- **Bluetooth – SoC Thermometer Micrium OS**: Demonstrates the integration of Micrium RTOS into Bluetooth applications. RTOS is added to the Bluetooth - SoC Thermometer sample app.
- **Bluetooth – SoC Throughput(\*)**: Tests the throughput capabilities of the device and can be used to measure throughput between two EFR32 devices, as well as between a device and a smartphone using the EFR Connect mobile app, through the Throughput demo tile.
- **Bluetooth – SoC Voice(\*)**: Voice over Bluetooth Low Energy sample application. It is supported by Thunderboard Sense 2 and Thunderboard EFR32BG22 boards and demonstrates how to send voice data over GATT, which is acquired from the on-board microphones.
- **Bluetooth – SoC iBeacon(\*)**: An iBeacon device implementation that sends non-connectable advertisements in iBeacon format. The iBeacon Service gives Bluetooth accessories a simple and convenient way to send iBeacons to smartphones. This example can be tested together with the EFR Connect mobile app.
- **Bluetooth – SoC Thunderboard Sense 2(\*)**, and **Thunderboard EFR32BG22(\*)**: Demonstrate the features of the Thunderboard Kit. These can be tested with the Thunderboard mobile app.
- **Dynamic Multiprotocol Examples** (see [AN1134: Dynamic Multiprotocol Development with Bluetooth and Proprietary Protocols on RAIL](#) for more information)
  - **Bluetooth – SoC Empty RAIL DMP FreeRTOS**: A minimal project structure, used as a starting point for custom Bluetooth + Proprietary DMP (Dynamic Multiprotocol) applications. It runs on top of FreeRTOS and multiprotocol RAIL.
  - **Bluetooth – SoC Empty RAIL DMP Micrium OS**: A minimal project structure, used as a starting point for custom Bluetooth + Proprietary DMP (Dynamic Multiprotocol) applications. It runs on top of Micrium OS and multiprotocol RAIL.
  - **Bluetooth – SoC Empty Standard DMP FreeRTOS**: A minimal project structure, used as a starting point for custom Bluetooth + Standard DMP (Dynamic Multiprotocol) applications. It runs on top of FreeRTOS and multiprotocol RAIL utilizing IEE802.15.4 standard protocol.
  - **Bluetooth – SoC Empty Standard DMP Micrium OS**: A minimal project structure, used as a starting point for custom Bluetooth + Standard DMP (Dynamic Multiprotocol) applications. It runs on top of Micrium OS and multiprotocol RAIL, utilizing IEE802.15.4 standard protocol.

- **Bluetooth – SoC Light RAIL DMP FreeRTOS(\*)**: A Dynamic Multiprotocol reference application demonstrating a light bulb that can be switched both via Bluetooth and via a Proprietary protocol. To switch it via Bluetooth use the Wireless Gecko smartphone app. To switch it via Proprietary protocol use the Flex (RAIL) - Switch sample app.
- **Bluetooth – SoC Light RAIL DMP Micrium OS**: A Dynamic Multiprotocol reference application demonstrating a light bulb that can be switched both via Bluetooth and via a Proprietary protocol. To switch it via Bluetooth use the Wireless Gecko smartphone app. To switch it via Proprietary protocol use the Flex (RAIL) - Switch sample app.
- **Bluetooth – SoC Light Standard DMP FreeRTOS(\*)**: A Dynamic Multiprotocol reference application demonstrating a light bulb that can be switched both via Bluetooth and via a standard protocol. To switch it via Bluetooth use the Wireless Gecko smartphone app. To switch it via standard protocol use the "Flex (RAIL) - Switch Standards" sample app.
- **Bluetooth – SoC Light Standard DMP Micrium OS(\*)**: A Dynamic Multiprotocol reference application demonstrating a light bulb that can be switched both via Bluetooth and via a standard protocol. To switch it via Bluetooth use the Wireless Gecko smartphone app. To switch it via standard protocol use the "Flex (RAIL) - Switch Standards" sample app.
- **NCP Host Examples** (located in C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko\_sdk\_suite\<version>\app\bluetooth\example\_host)
  - **Empty**: Minimal host-side project structure, used as a starting point for NCP host applications. Use it with the **Bluetooth – NCP** target application flashed to the radio board.
  - **ota-dfu**: Demonstrates how to perform an OTA DFU on a Silicon Labs Bluetooth Device. It requires a WSTK with a radio board flashed with NCP firmware to be used as the GATT client that performs the OTA.
  - **uart-dfu**: Demonstrates how to perform a UART DFU on a Silicon Labs Bluetooth Device running NCP firmware.
  - **voice**: On a WSTK programmed with NCP firmware, it connects to the **Bluetooth – SoC Voice** example, sets the correct configuration on it, receives audio via Bluetooth, and stores audio data into a file.
  - **aoa\_locator**: A locator host sample app that works together with a **Bluetooth – NCP AoA Locator** target app. It receives IQ samples from the target and estimates the Angle of Arrival (AoA). For more information see *AN1296: Application Development with Silicon Labs' RTL Library*.
  - **aoa\_multilocator**: Connects to multiple **aoa\_locator** sample apps (via MQTT) and estimates a position from Angles of Arrival (AoA). For more information, see *AN1296: Application Development with Silicon Labs' RTL Library*.
  - **aoa\_multilocator\_gui**: Connects to the **aoa\_multilocator** sample app (via MQTT), reads out the position estimations and displays the tags and locators on a 3D GUI. This sample app is python based. For more information, see *AN1296: Application Development with Silicon Labs' RTL Library*.
  - **aoa\_compass**: Demo application with GUI showcasing the angle estimation capabilities of the RTL library. This is now replaced with the AoA Analyzer tool as described in QSG175: *Silicon Labs Direction Finding Solution Quick-Start Guide*.
  - **throughput**: Tests the throughput capabilities of the device in NCP mode and can be used to measure throughput between two devices as well as between a device and a smartphone.

## 4 Getting Started with Bluetooth Demo Software

The Blue Gecko Bluetooth Wireless Starter Kit is meant to help you evaluate Silicon Labs' Bluetooth modules and get you started with your own software development. The kits come in different versions with different module radio boards. See <https://www.silabs.com/products/development-tools/wireless/bluetooth/bluegecko-bluetooth-low-energy-module-wireless-starter-kit> for details on the current configurations.

To get started with Bluetooth demo software, you should have downloaded Simplicity Studio 5 (SSv5) and the Bluetooth SDK v3.x as described in the [Simplicity Studio 5 User's Guide](#), available online and through the SSv5 help menu. The Bluetooth SDK comes with some prebuilt demos that can be flashed to your EFR32 device, and tested using a Smartphone. This section describes how to test three prebuilt demos on both Android and iOS devices:

- NCP Empty demo
- iBeacon demo
- Health Thermometer demo

### 4.1 Prepare the WSTK

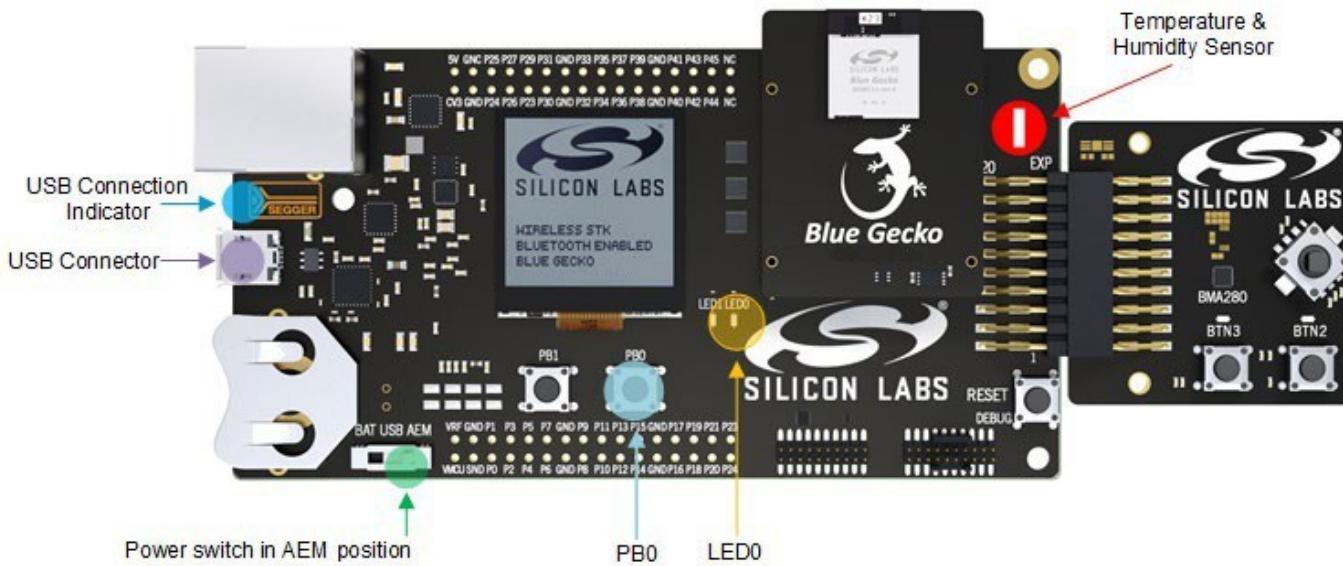
1. Connect a Bluetooth Module Radio Board to the WSTK Main Board as shown in the following figure.
2. Connect the WSTK to a PC using the **Main Board USB** connector.
3. Turn the **Power switch** to "AEM" position.

**Note:** At this stage you might be prompted to install the drivers for the WSTK Main Board but you can skip this for now.

4. Check that the blue **USB Connection Indicator** LED turns on or starts blinking.
5. Check that the Main Board LCD display turns on and displays a Silicon Labs logo.

Before starting to test the demo application note the following parts on the WSTK Main Board

- Temperature & Humidity Sensor
- PB0 button
- LED0



## 4.2 Flash the Demo

- With your device connected as described above, open SSv5.
- Select your device in the Debug Adapters view.
- On the Example Projects & Demos tab, click **RUN** on the demo of choice.

## 4.3 Test the Bluetooth Demos Using an Android or iOS Smartphone

### 4.3.1 Testing the NCP Empty Demo

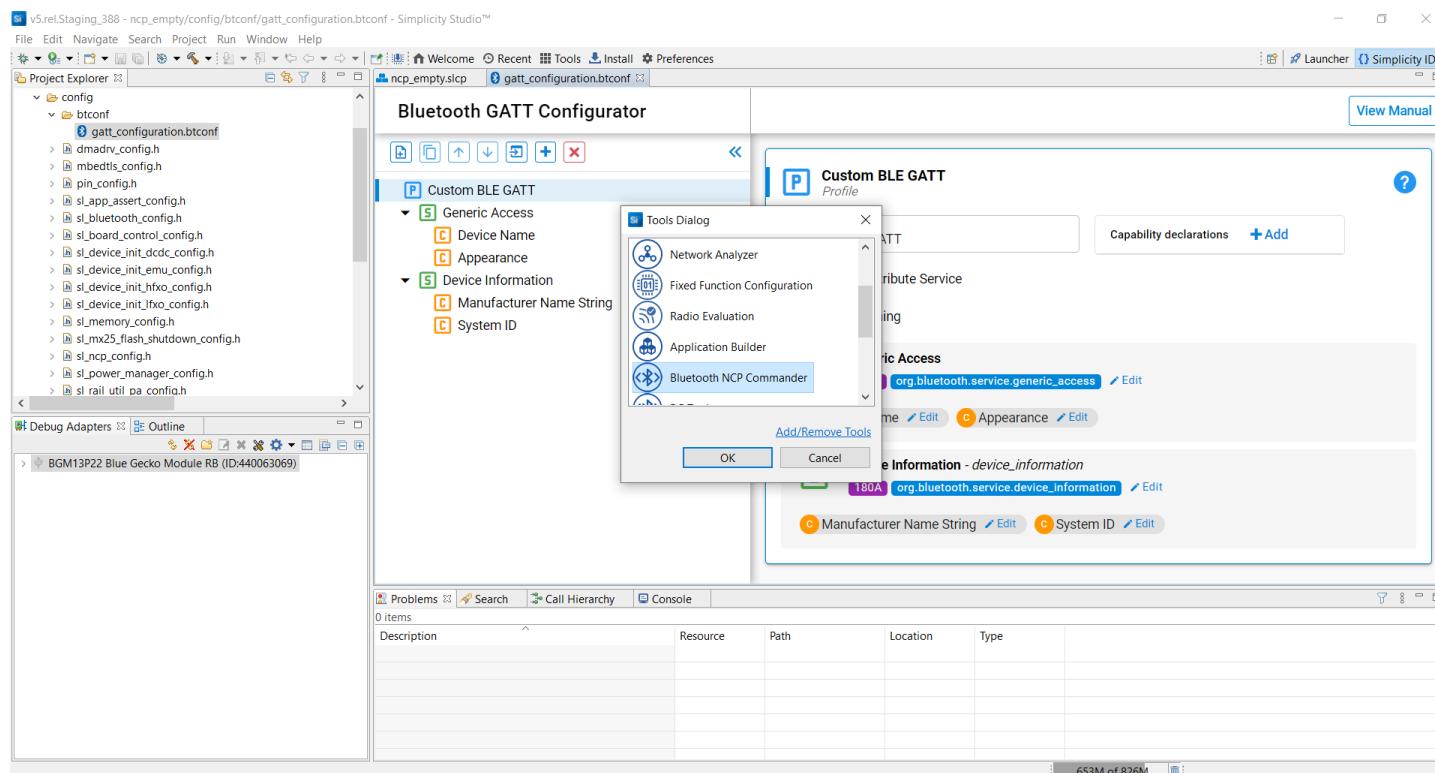
Load the **NCP Empty** demo on the target:

- Open Simplicity Studio with a WSTK and radio board connected and select the corresponding debug adapter.
- On the OVERVIEW tab, under “General Information”, select the Gecko SDK Suite if it is not selected. On the Example Projects & Demos tab, select the **NCP Empty** demo and click **RUN**. This flashes the demo to your device, but it does not start advertising automatically.

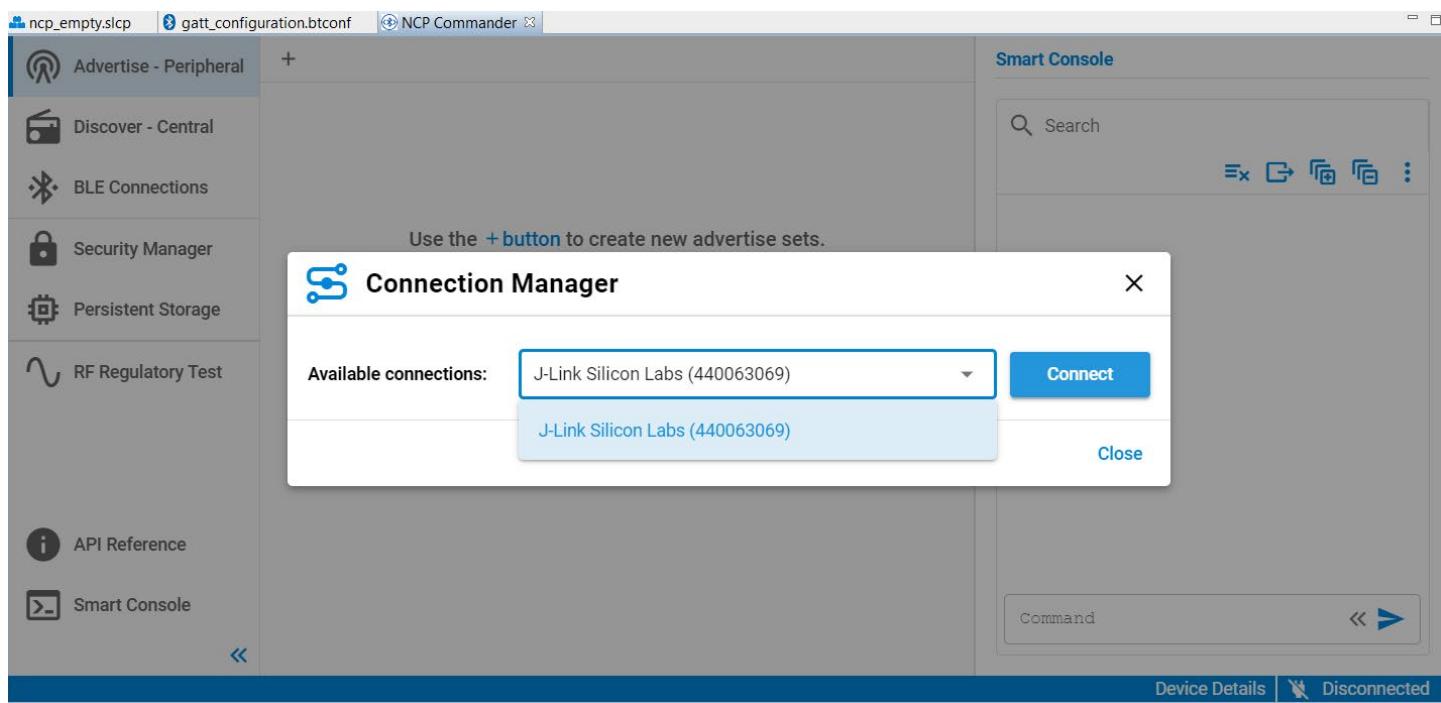
At this point, BGAPI 3.0 commands can be sent to the kit. Starting with Bluetooth SDK version 3.1.0, Silicon Labs introduced a new tool, the Bluetooth NCP Commander. This will replace BGTool. Bluetooth NCP Commander and BGTool can be used to send BGAPI 3.0 commands to the kit, using UART. Connections, advertising and other standard Bluetooth Low Energy operation can be controlled via this tool.

### Bluetooth NCP Commander

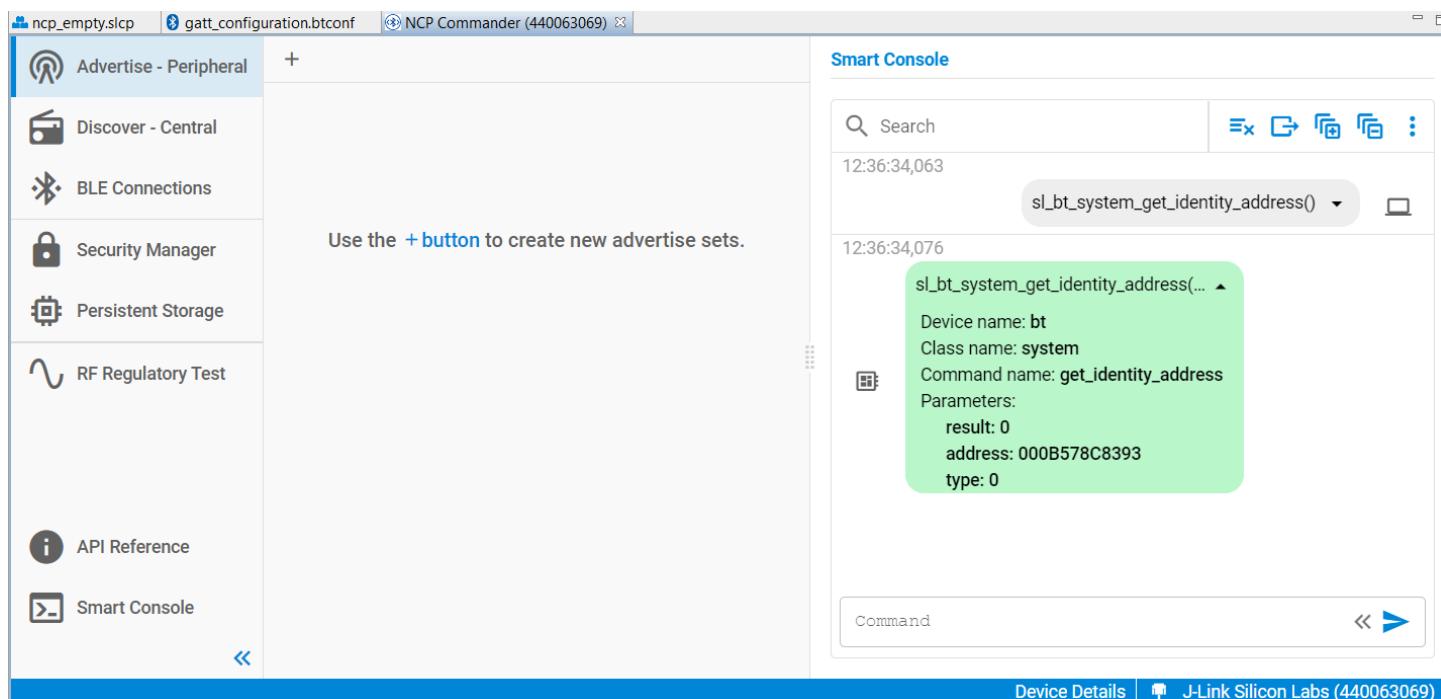
Bluetooth NCP Commander can be opened through the Project Configurator’s Configuration Tools tab, or from the Simplicity Studio Tools menu.



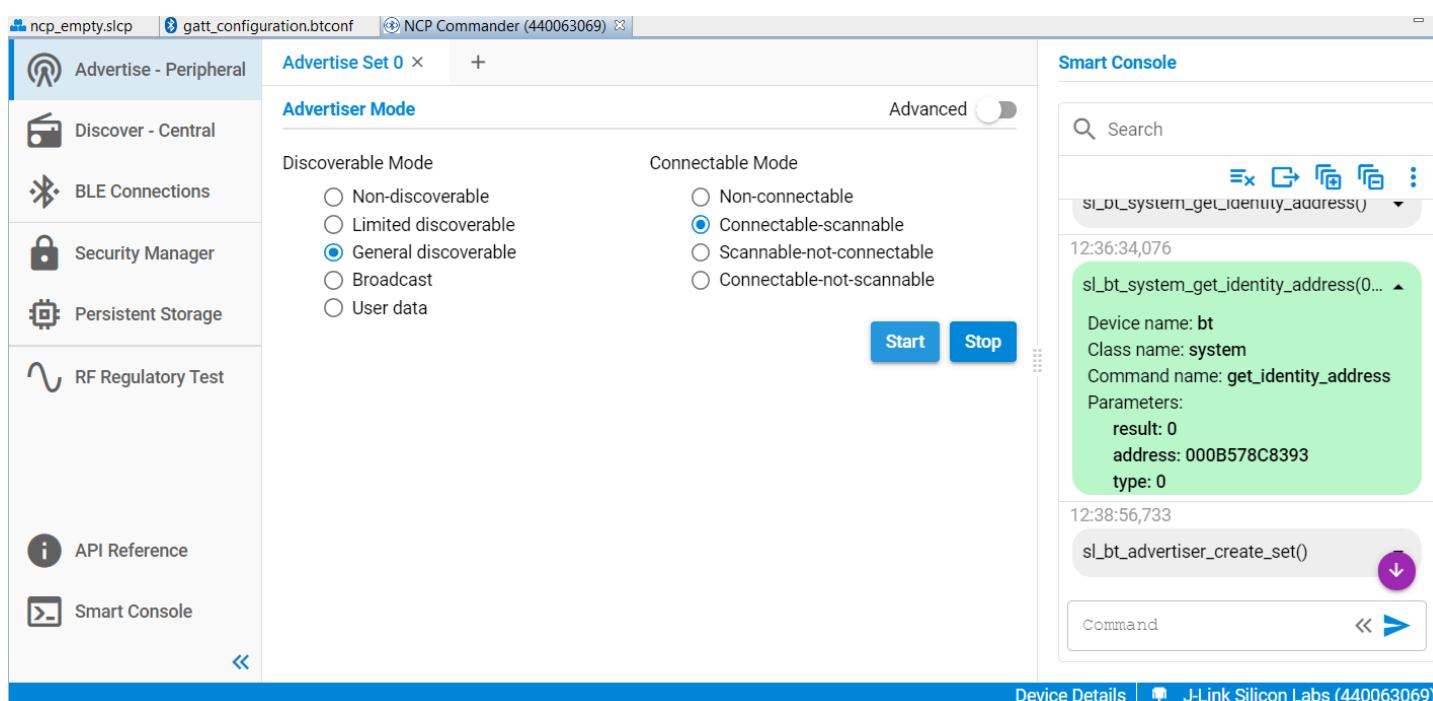
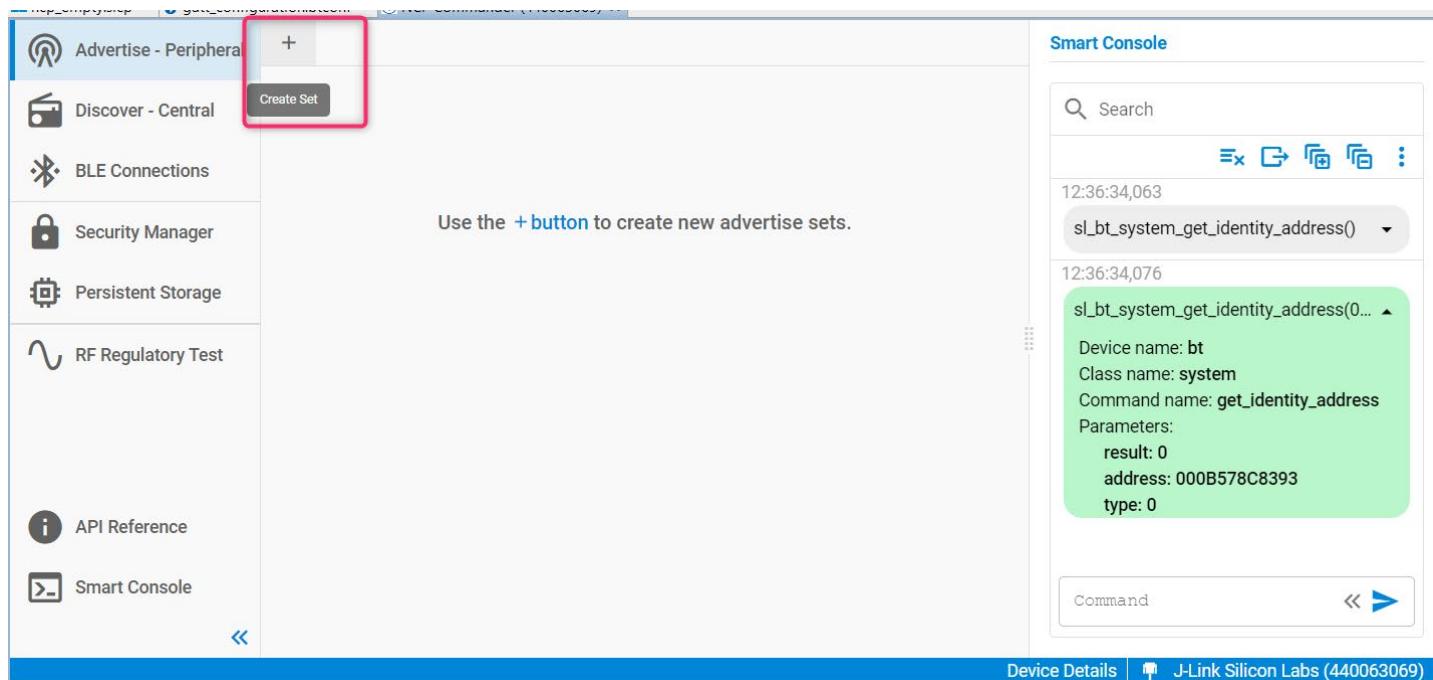
1. Launch Bluetooth NCP Commander and then establish the virtual UART connection to the kit using the JLink adapter:



2. Click **Connect**. If everything works correctly you should see the result of the "sl\_bt\_system\_get\_identity\_address" command displayed in green:

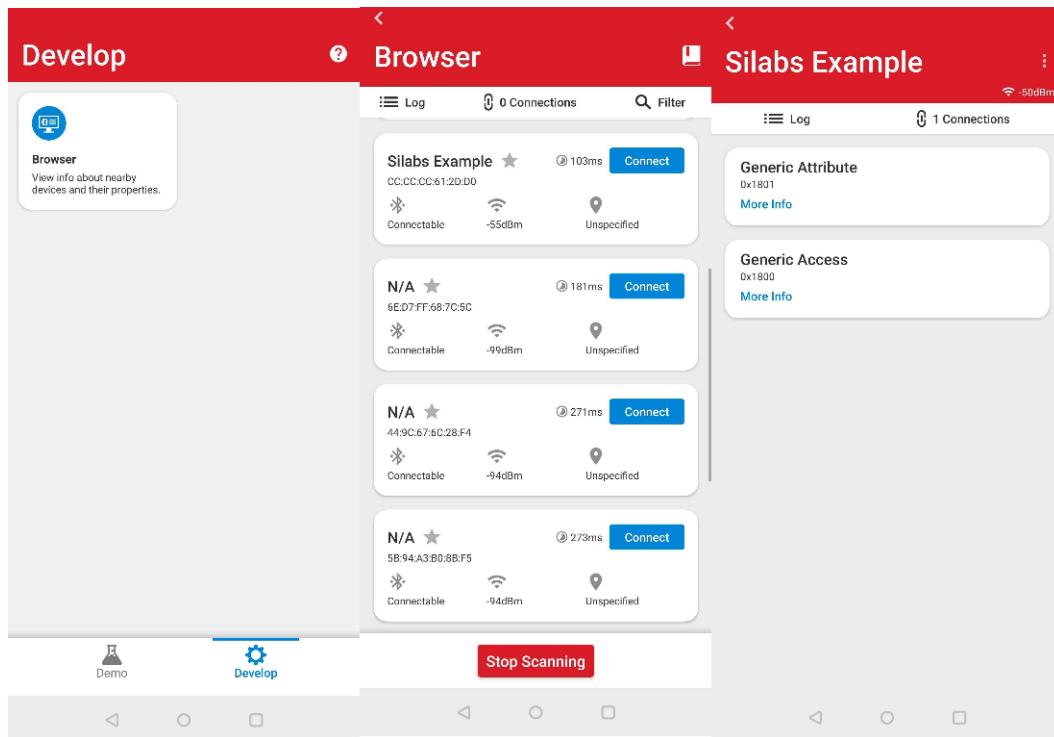


3. In the 'Advertise' menu, click on the '+' button (**Create Set**) to create an advertiser and then click **Start** to start advertising. This will instruct the Bluetooth stack to start a connectable advertisement.

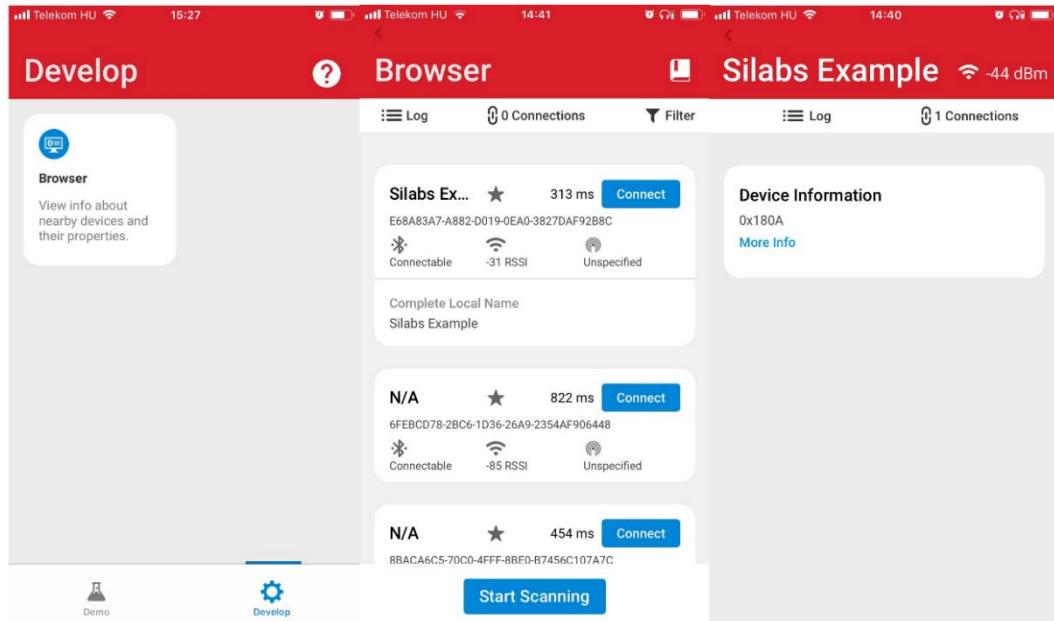


## Testing with the Smartphone App

1. On the master side (smartphone), install the **EFR Connect** app from the Google Play Store or the App Store, and open it. To find your advertising device, tap the Develop tab, and tap **Bluetooth Browser**. This shows all advertising devices nearby. Connect to your device by tapping **Connect** next to "Silabs Example". Its GATT database is automatically discovered and displayed. Tap any service to list its characteristics and tap any characteristic to read its value.



### 4.1 Testing on Android Smartphone

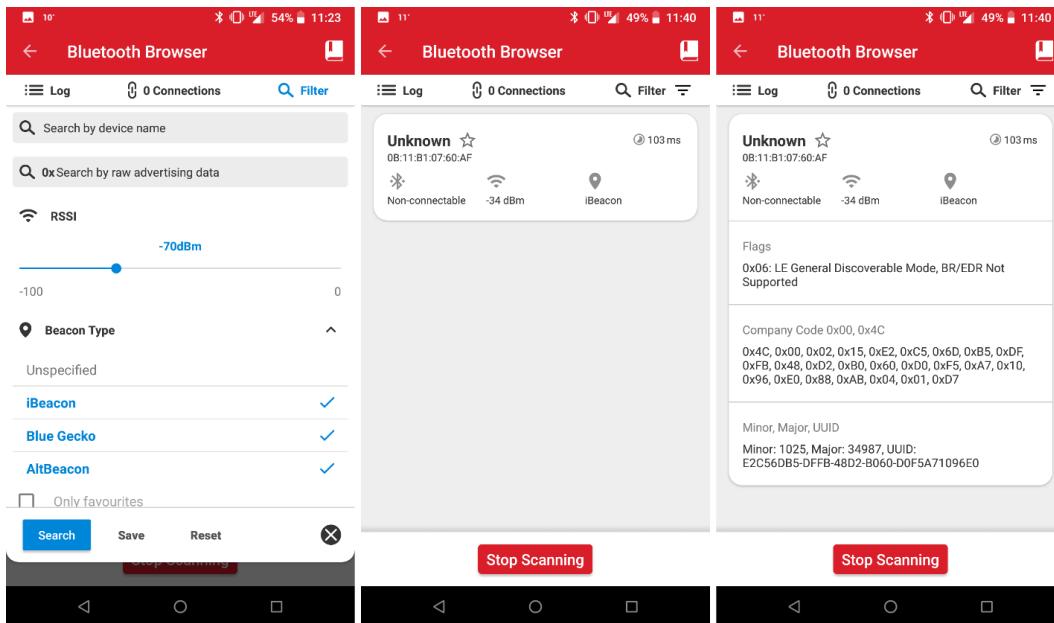


### 4.2 Testing on iOS Smartphone

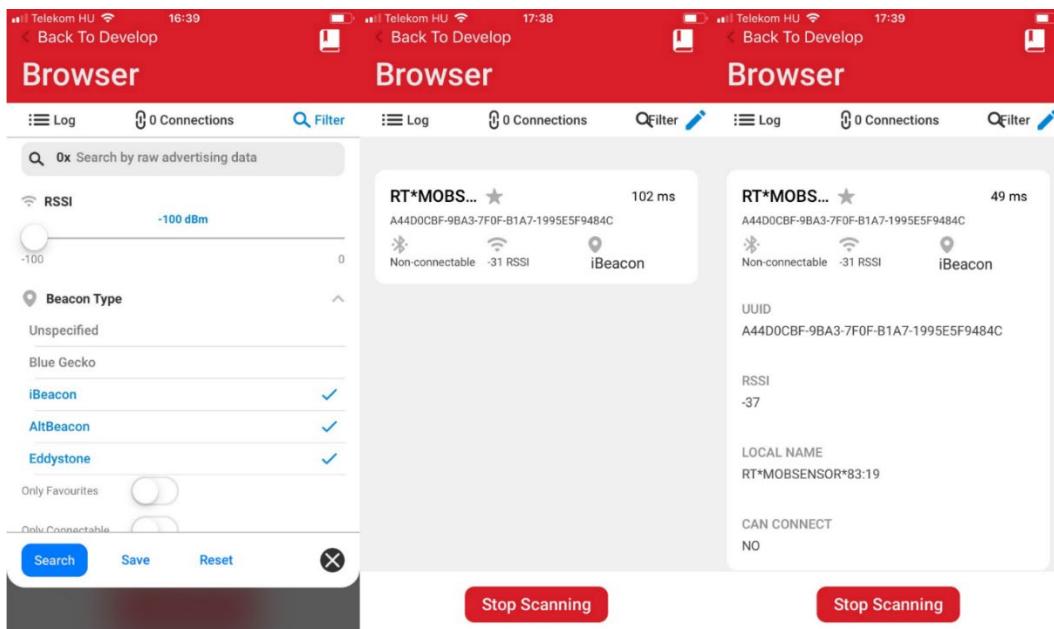
#### 4.3.2 Testing the iBeacon Demo

Bluetooth beacons are unconnectable advertisements that help you locate a device, determine your own position, or get minimal information about an asset the beaconing device is attached to.

After flashing the **iBeacon** demo to your device, you can find the beacon signal with the **Bluetooth Browser** in the **EFR Connect** app. Start **EFR Connect**, tap the Develop tab, and tap **Bluetooth Browser**. To filter beacons, tap **Filter**, and select the beacon types you want to be displayed. The app provides you with basic information about the beacon, like RSSI - which can help determine the distance of the beacon. Tap the beacon to get more information about the data it provides.



#### 4.3 Testing on Android Smartphone



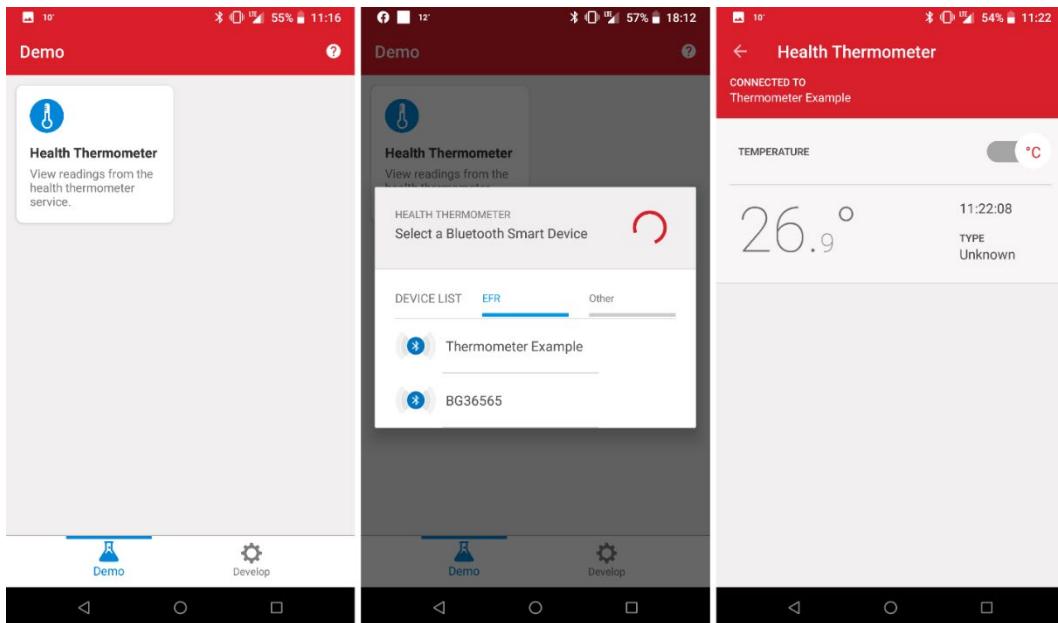
#### 4.4 Testing on iOS Smartphone

#### 4.3.3 Testing the Health Thermometer Demo

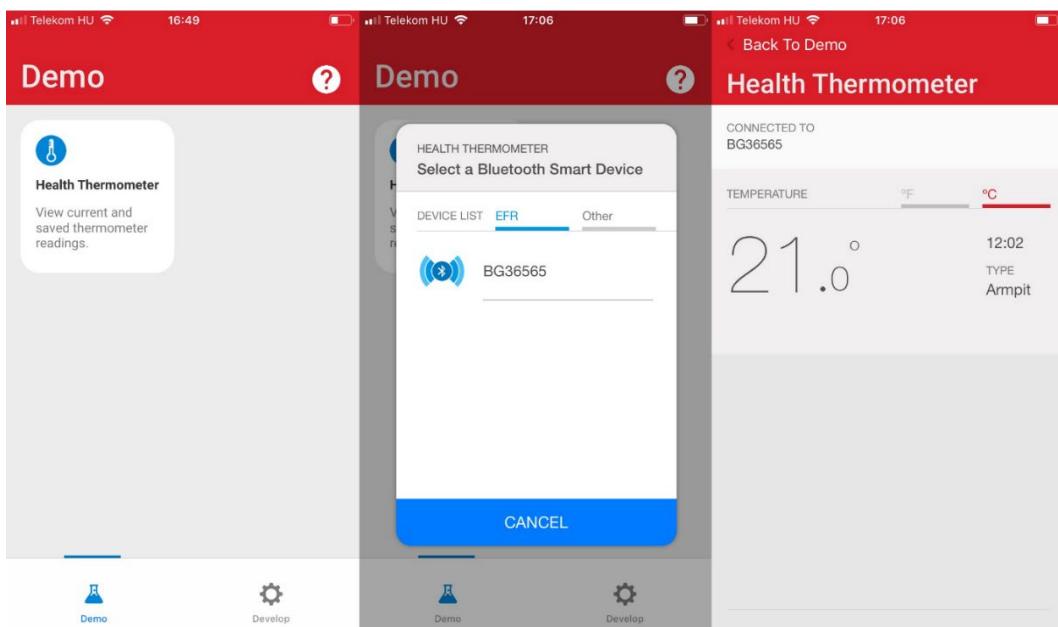
While the NCP Empty demo implements a minimal GATT database with basic static information like device name, the Health Thermometer demo extends this database with live temperature measurements.

After flashing the **Health Thermometer** demo to your device, start **EFR Connect**, tap the Demo tab, and tap **Health Thermometer**. Find your device advertising as Thermometer Example in the device list, and tap it to connect. The smartphone app automatically finds the Temperature measurement characteristic of the device, reads its value periodically, and displays the value on the screen of the phone.

Try touching the temperature sensor located on the WSTK (see section [4.1 Prepare the WSTK](#)). You should be able to see the temperature changing.



4.5 Testing on Android Smartphone



4.6 Testing on iOS Smartphone

## 5 Starting Application Development

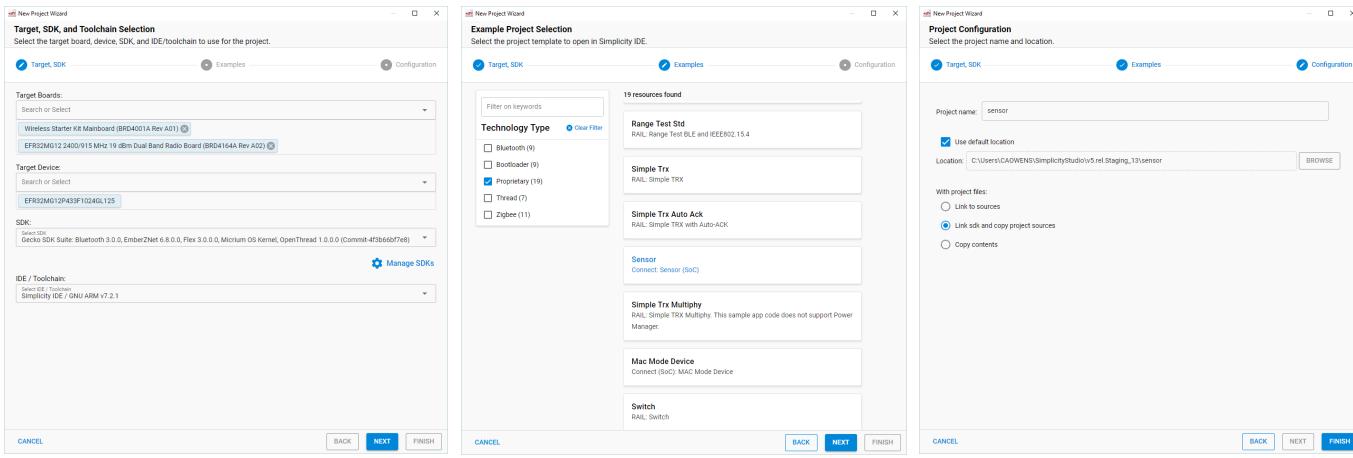
Developing a Bluetooth application consists of two main steps: defining the GATT database structure and defining the event handlers for events such as `connection_opened`, `connection_closed`, and so on.

The most common starting point for application development is the **SoC Empty** example. This project contains a simple GATT database (including the Generic Access service, Device Information service, and OTA service) and a while loop that handles some events raised by the stack. You can extend both the GATT database and the event handlers of this example according to your needs.

**Note:** Beginning with Bluetooth SDK version 2.7.0.0, all devices must be loaded with the Gecko Bootloader as well as the application. While you are getting started, the easiest way to do this is to load any of the precompiled demo images that come with the bootloader configured as part of the image. When you flash your application it overwrites the demo application, but the bootloader remains. Subsequently you may wish to build your own bootloader, as described in *UG266: Silicon Labs Gecko Bootloader User Guide*.

New Project creation is done through three dialogs:

- Target, SDK, and Toolchain
- Examples
- Configuration



An indicator at the top of the dialog shows you where you are.



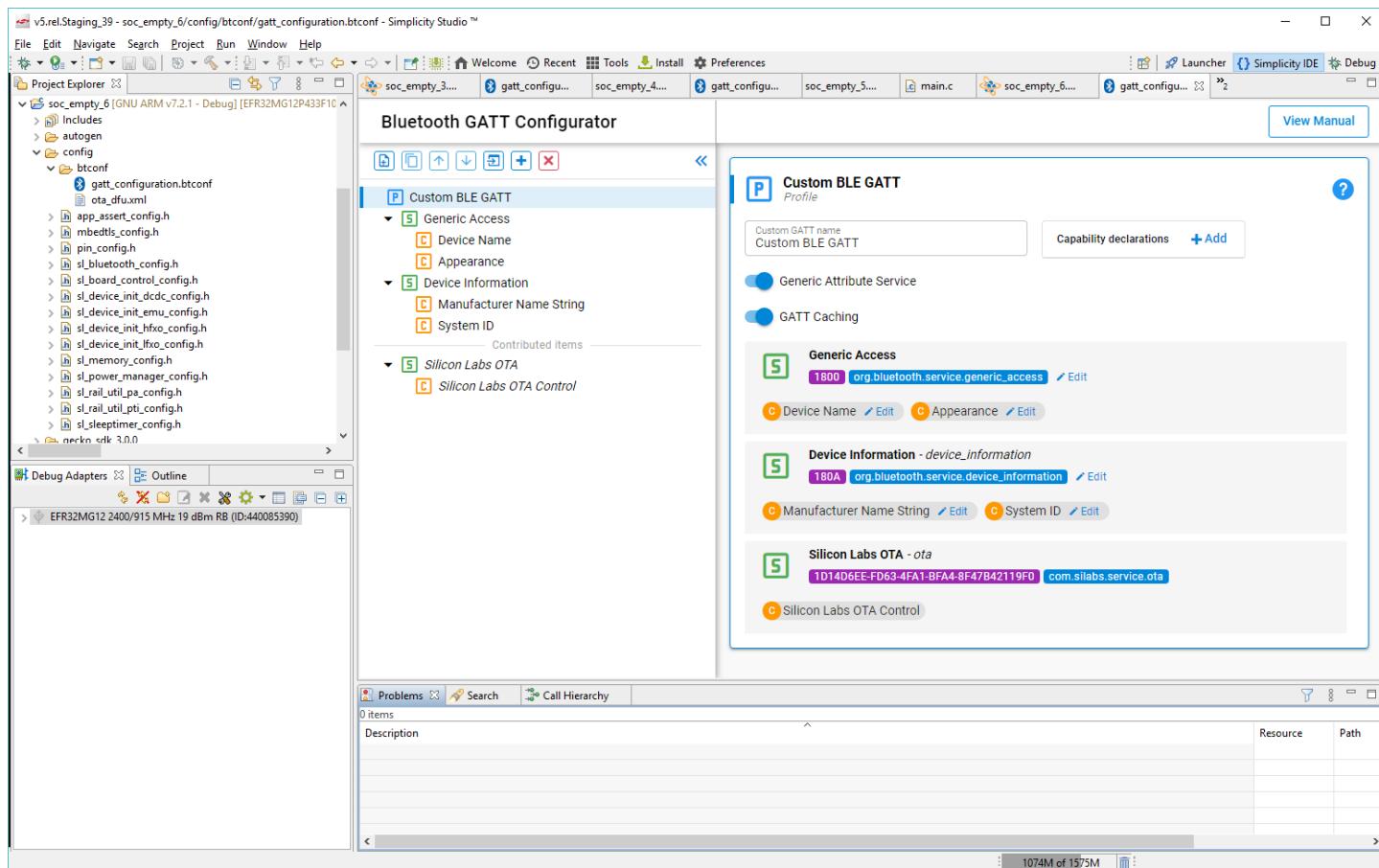
You can start a project from different locations in the Launcher Perspective, as described in the [Simplicity Studio 5 User's Guide](#). While you are getting started, we suggest starting from the File menu, as that takes you through all three of the above dialogs.

1. Select **New >> Silicon Labs Project Wizard**.
2. Review your SDK and toolchain. If you wish to use IAR instead of GCC, be sure to change it here. Once you have created a project it is difficult to change toolchains. Click **NEXT**.
3. On the Example Project Selection dialog, filter on Bluetooth and select **SoC Empty**. Click **NEXT**.
4. On the Project Configuration dialog, rename your project if you wish. Note that if you change any linked resource, it is changed for any other project that references it. While you are getting started the default choice to include project files but link to the SDK is best. Click **FINISH**. If the example has documentation, the project opens on a readme tab. Note that a Simplicity IDE perspective control is now included in the upper right of the screen.

## 5.1 GATT Database

A visual GATT Configurator is available on the gatt\_configuration.btconf tab when you create the project, to help you create your own GATT database with a few clicks.

You can create your own database at this point, or return to it later either by double-clicking the gatt\_configuration.btconf file under your project in Project Explorer, or through the Project Configurator's Advanced > GATT Configurator component. For more information, see section [6.1 The GATT Configurator](#).



A reference for each characteristic is generated and defined in `gatt_db.h`. You can use these references in your code to read / write the values of the characteristics in the local GATT database with `sl_bt_gatt_server_read_attribute_value()` / `sl_bt_gatt_server_write_attribute_value()` commands.

You can also build the GATT database from the application using the GATT database API. In this case you need to install the Dynamic GATT Database software component on the Project Configurator Software Components tab.

You will find the event handlers in the main loop in `app.c`. You can extend this list with further event handlers. The full list of events – and stack commands – can be found in the [Bluetooth Software API Reference Manual](#).

## 5.2 Component Configuration

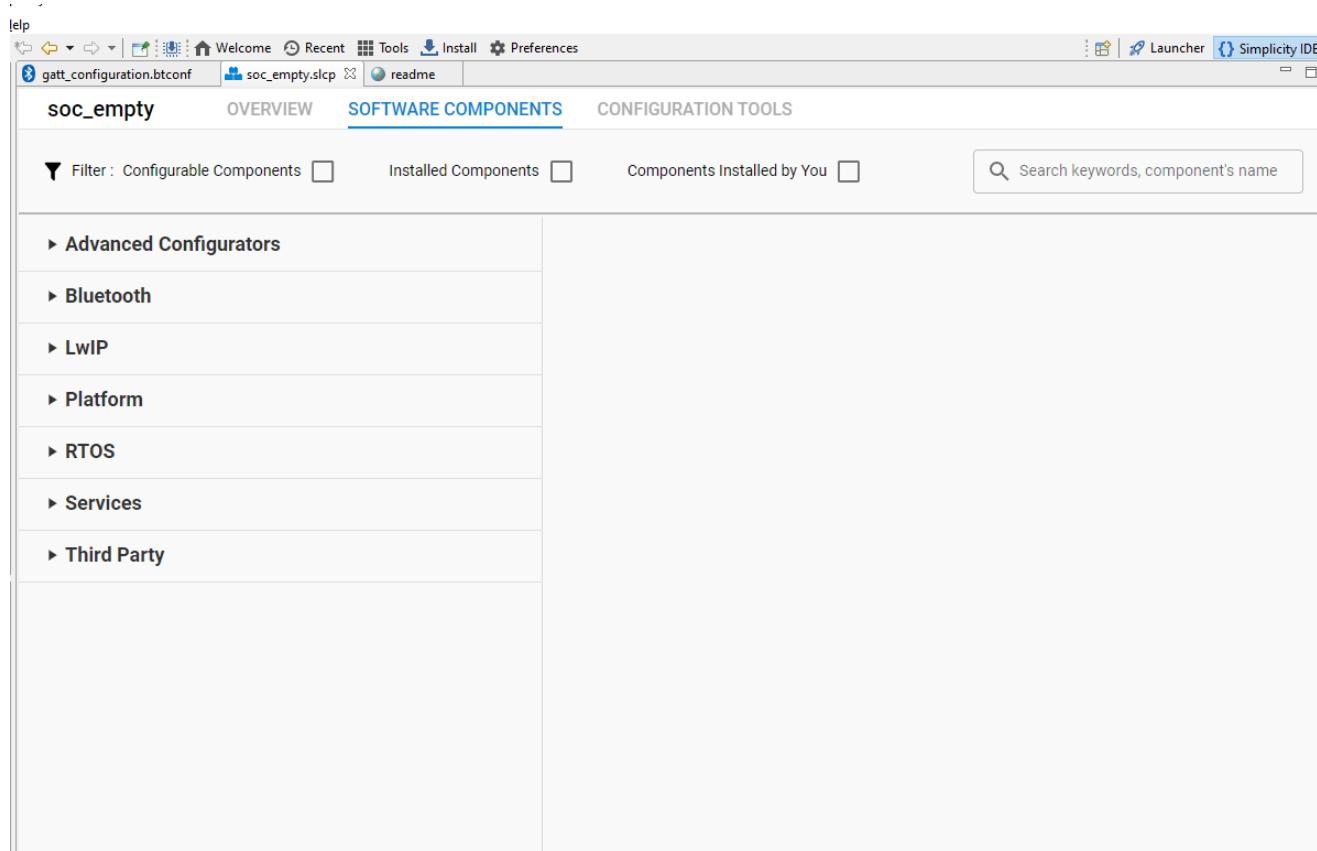
Bluetooth SDK v3.x projects are based on a Gecko Platform component-based architecture. Software features and functions can be installed and configured through Simplicity Studio's Component Editor. When you install a component, the installation process will:

1. Copy the corresponding SDK files from the SDK folder into the project folder.
2. Copy all the dependencies of the given component into the project folder.
3. Add new include directories to the project settings.
4. Copy the configurations files into the `/config` folder.
5. Modify the corresponding auto-generated files to integrate the component into the application.

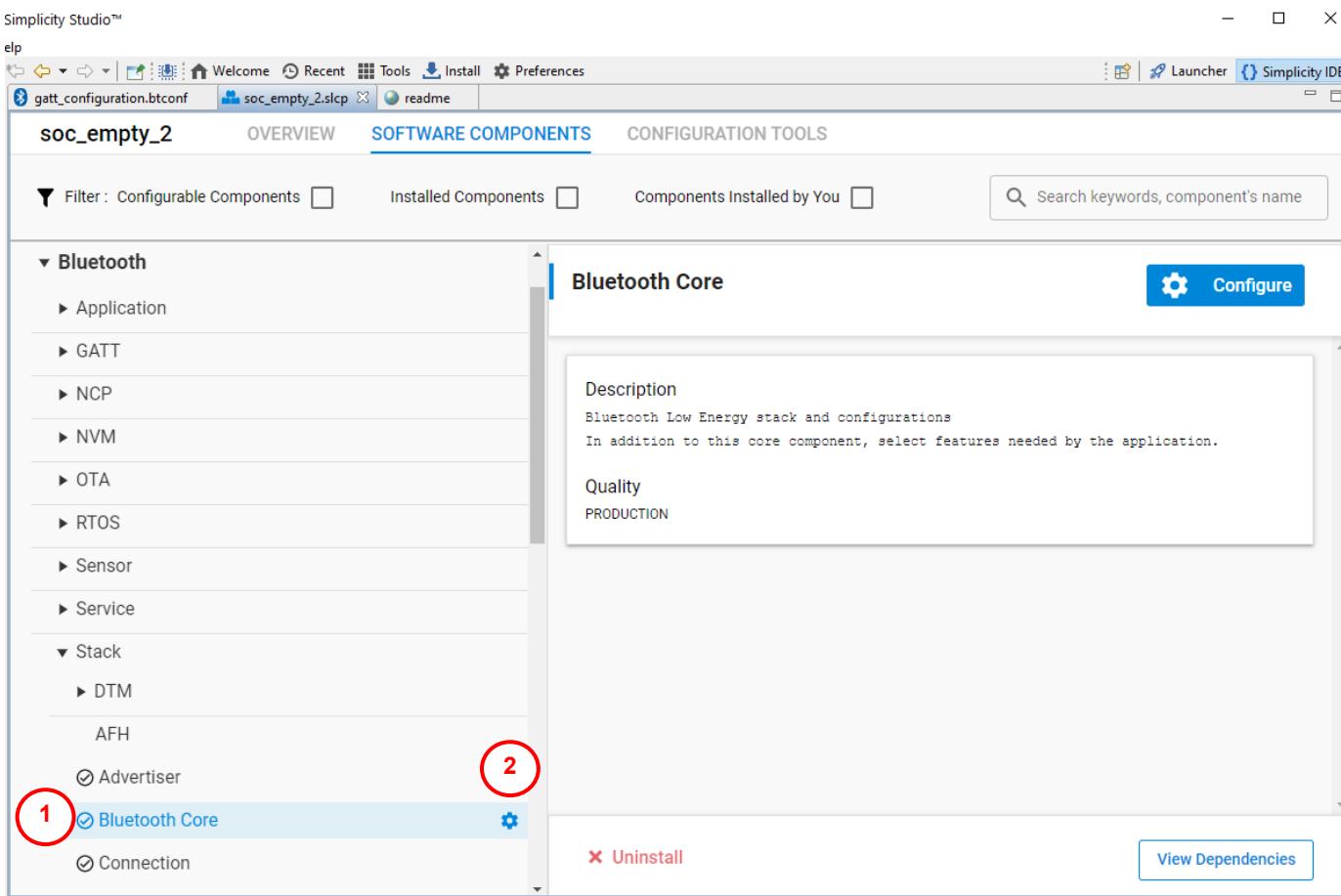
Additionally, “init” type software components will implement the initialization code for a given component, utilizing their corresponding configuration file as input.

Some software components (like OTA DFU) will fully integrate into the application to perform a specific task without the need of any additional code, while other components provide an API to be used in the application.

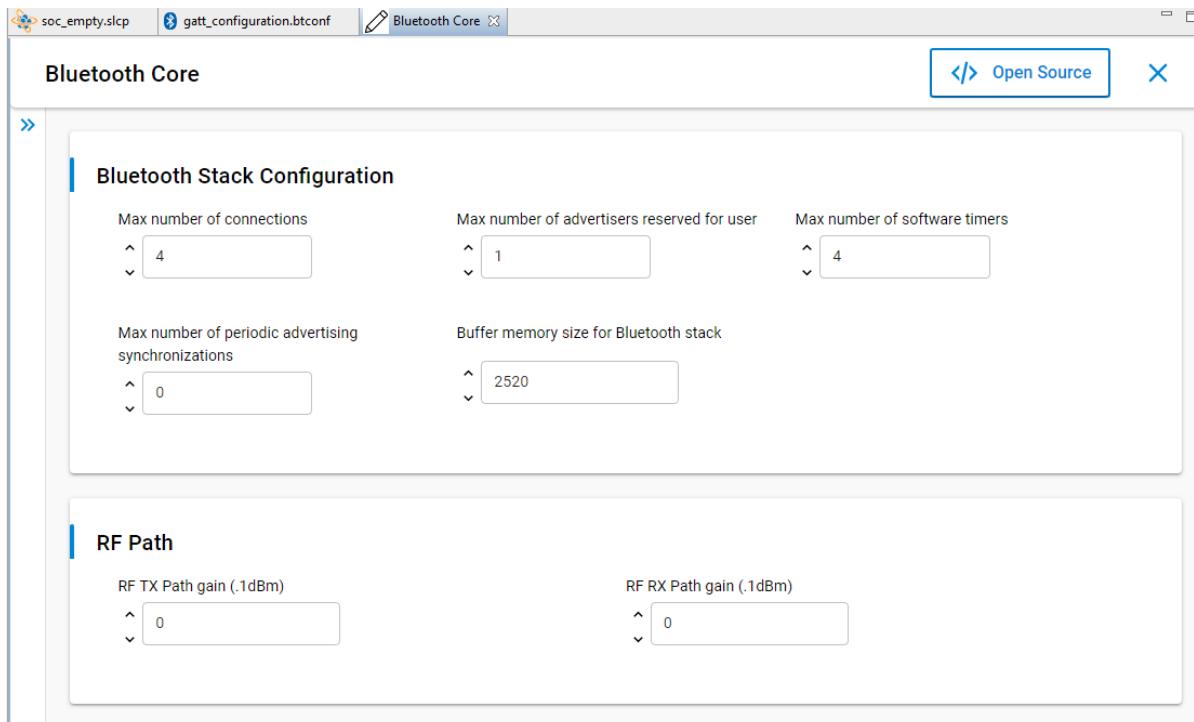
To see the component library, click the <project-name>.slcp tab of your project, and click **Software Components**. A number of filters as well as a keyword search are available to help you explore the various component categories. Note that, if you have more than one SDK installed, components for all are presented.



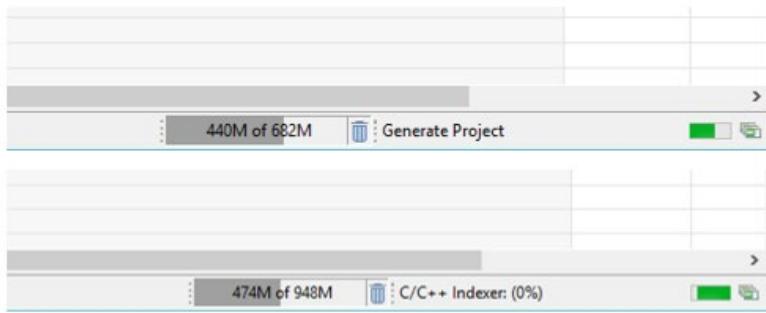
Components installed in the project are checked (1), and can be uninstalled. Configurable components are indicated by a gear symbol (2).



Click **Configure** to open the Component Editor and see a configurable component's parameters.



As you change component configurations, your changes are automatically saved and project files are automatically generated. You can see generation progress in the lower right corner of the Simplicity IDE. Wait until generation is complete before building the application image.



### 5.3 Building and Flashing

To build and debug your project click **Debug** (bug icon) on the Simplicity IDE It will build and download your project, and open up the Debug perspective. Click **Play** (next to Debug) to start running your project on the device.



### 5.4 Enabling Field Updates

Deploying new firmware for devices in the field can be done by UART DFU (Device Firmware Update) or, for SoC applications, OTA DFU. For more information on each of these methods refer to [AN1086: Using the Gecko Bootloader with the Silicon Labs Bluetooth Applications](#).

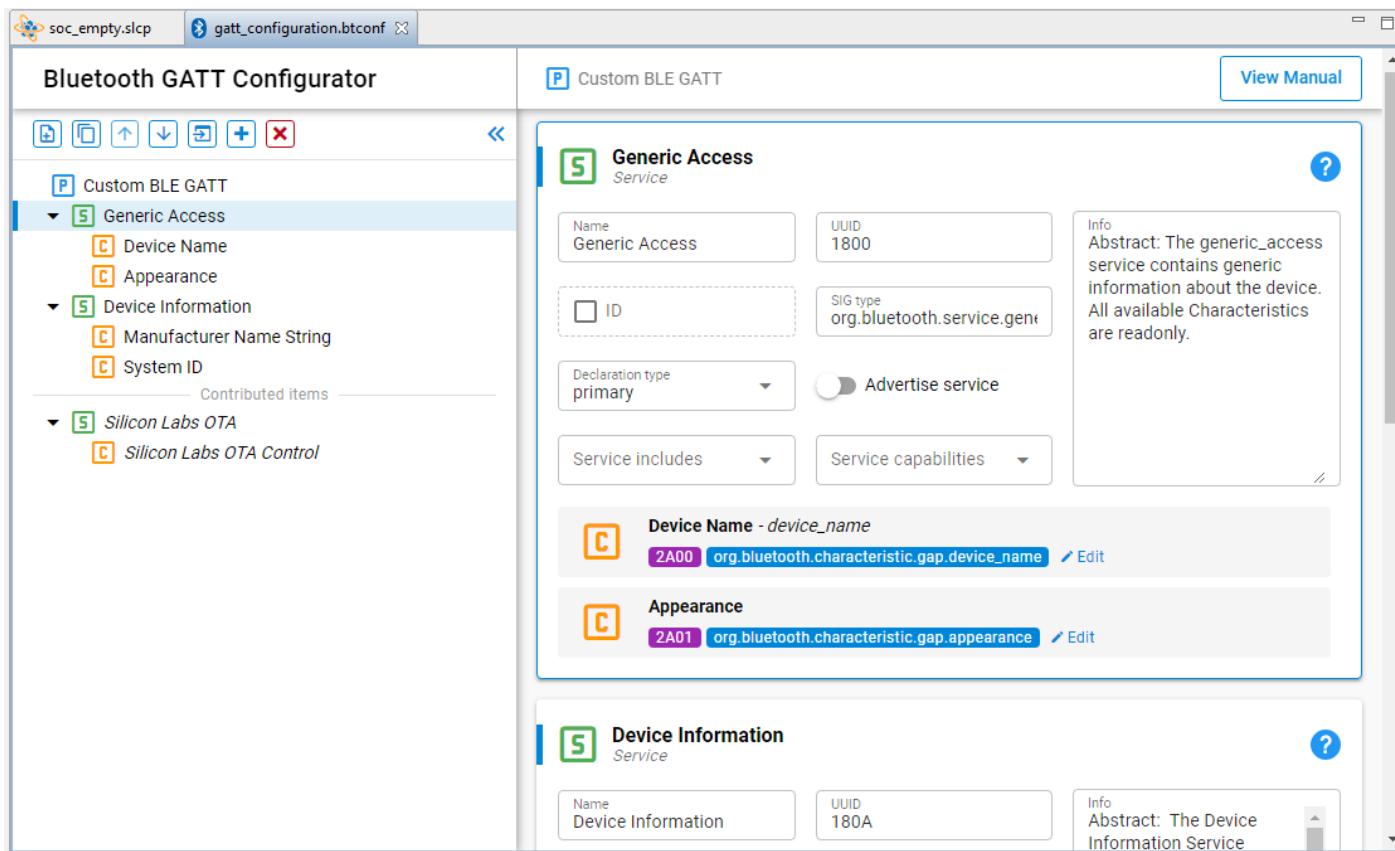
## 6 Development Tools

### 6.1 The GATT Configurator

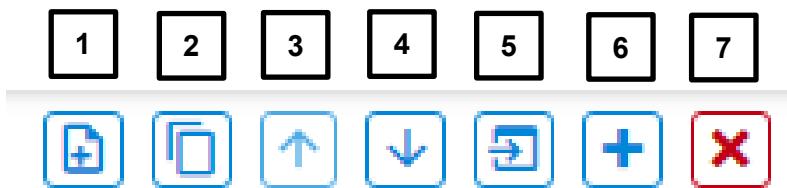
Every Bluetooth connection has a GATT client and a GATT server. The server holds a GATT database: a collection of *Characteristics* that can be read and written by the client. The Characteristics are grouped into *Services*, and the group of Services determines a *Bluetooth Profile*.

If you are implementing a GATT server (typically on the peripheral device), you have to define a GATT database structure. Clients (typically the central device) can also have a GATT database, even if no device will query it, so you can keep the default database structure in your code. This structure can either be designed in runtime using the dynamic GATT API or in compile-time using the GATT Configurator. For an SoC application, implementing a static GATT database with the GATT Configurator is recommended, because it ensures faster startup and lower memory consumption than you can achieve with dynamic GATT configuration.

The GATT Configurator is a simple-to-use tool to help you build your own GATT database. A list of project Profiles/Services/Characteristics/Descriptors is shown on the left and details about the selected item is shown on the right. An options menu is provided above the Profiles list.



The GATT Configurator menu is:



- 1) Add an item.
- 2) Duplicate the selected item.
- 3) Move the selected item up.
- 4) Move the selected item down.
- 5) Import a GATT database.
- 6) Add Predefined.
- 7) Delete the selected item.

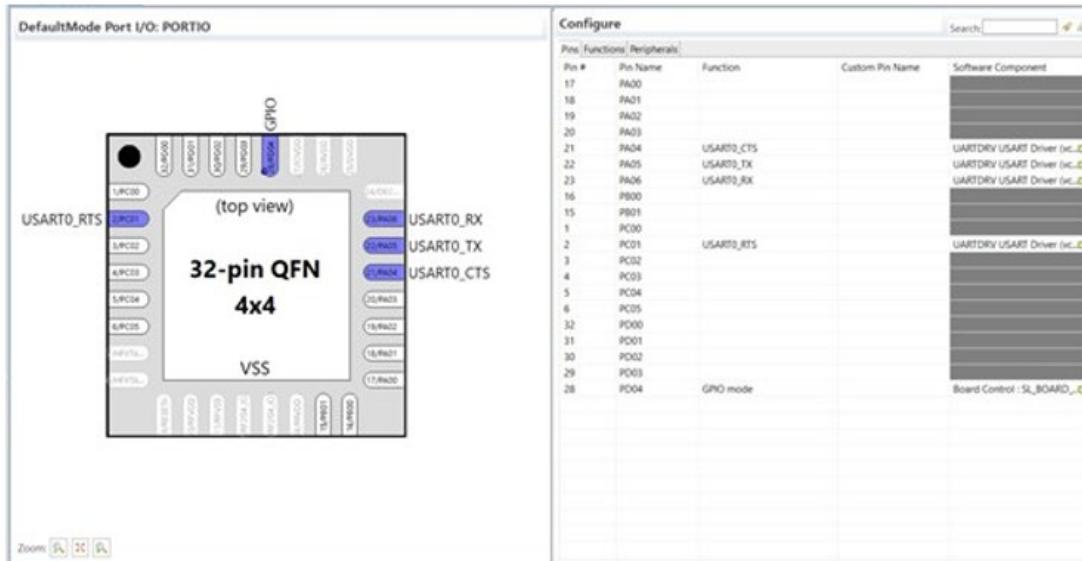
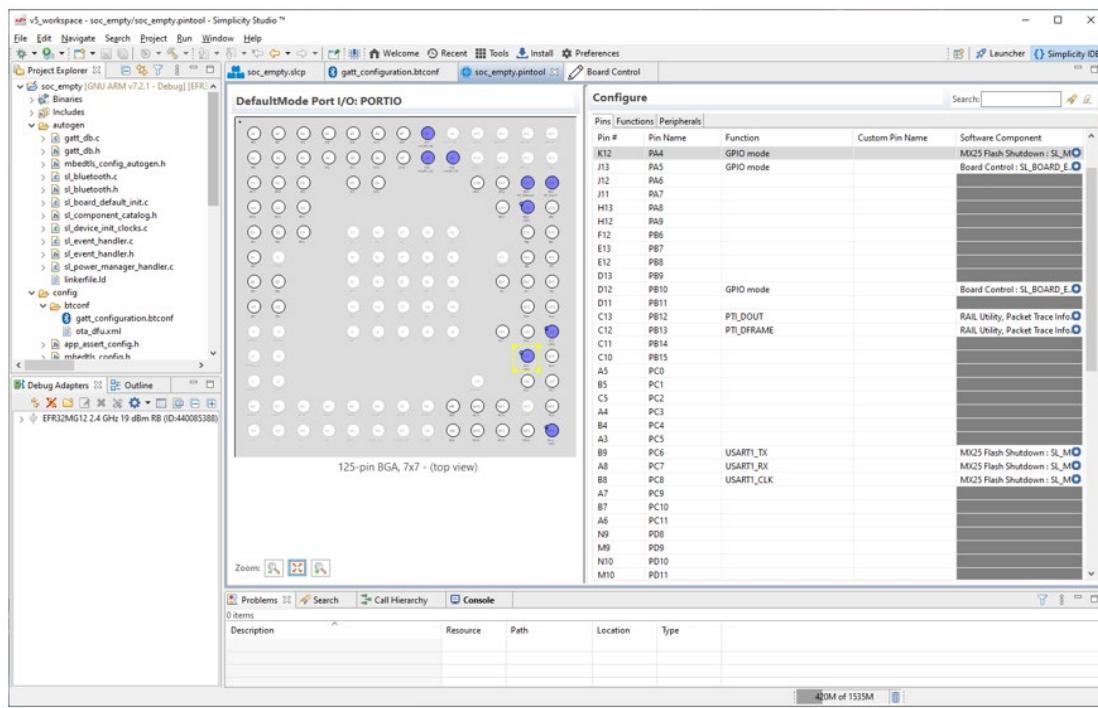
To add a custom service, click the **Profile (Custom BLE GATT)**, and then click **Add** (1). To add a custom characteristic, select a service and then click **Add** (1). To add a predefined service/characteristic click **Add Predefined** (6). To learn more about the configurator see [UG438: GATT Configurator User's Guide for Bluetooth SDK v3.x](#).

You can find a detailed description of any Profile/Service/Characteristic/Descriptor on <https://www.bluetooth.com/specifications/gatt>.

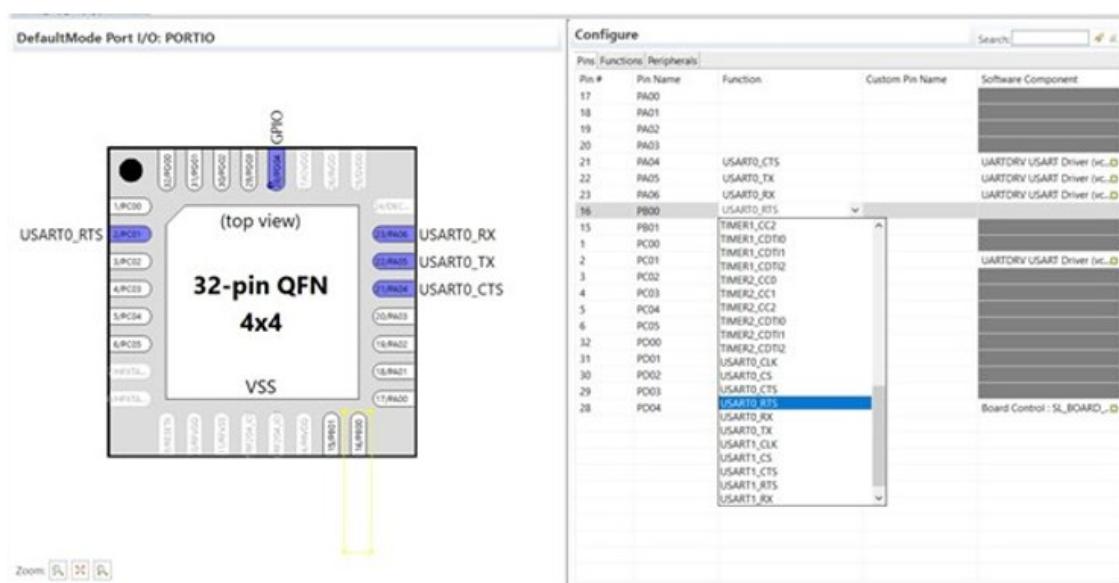
Characteristics are generally complex structures of fields. If you want to know what fields a characteristic has, visit <https://www.bluetooth.com/specifications/gatt/characteristics>.

## 6.2 The Pin Tool

Simplicity Studio 5 offers a Pin Tool that allows you to easily configure new peripherals or change the properties of existing ones. In the Project Configurator SOFTWARE COMPONENTS tab, expand the Advanced Configurators group and open the Pin Tool. The graphical view differs based on the chip.



For example, you can reassign the pins used for USART communication to the appropriate layout for a custom board design by selecting the desired pin in the list and then selecting its functionality from the drop-down list.



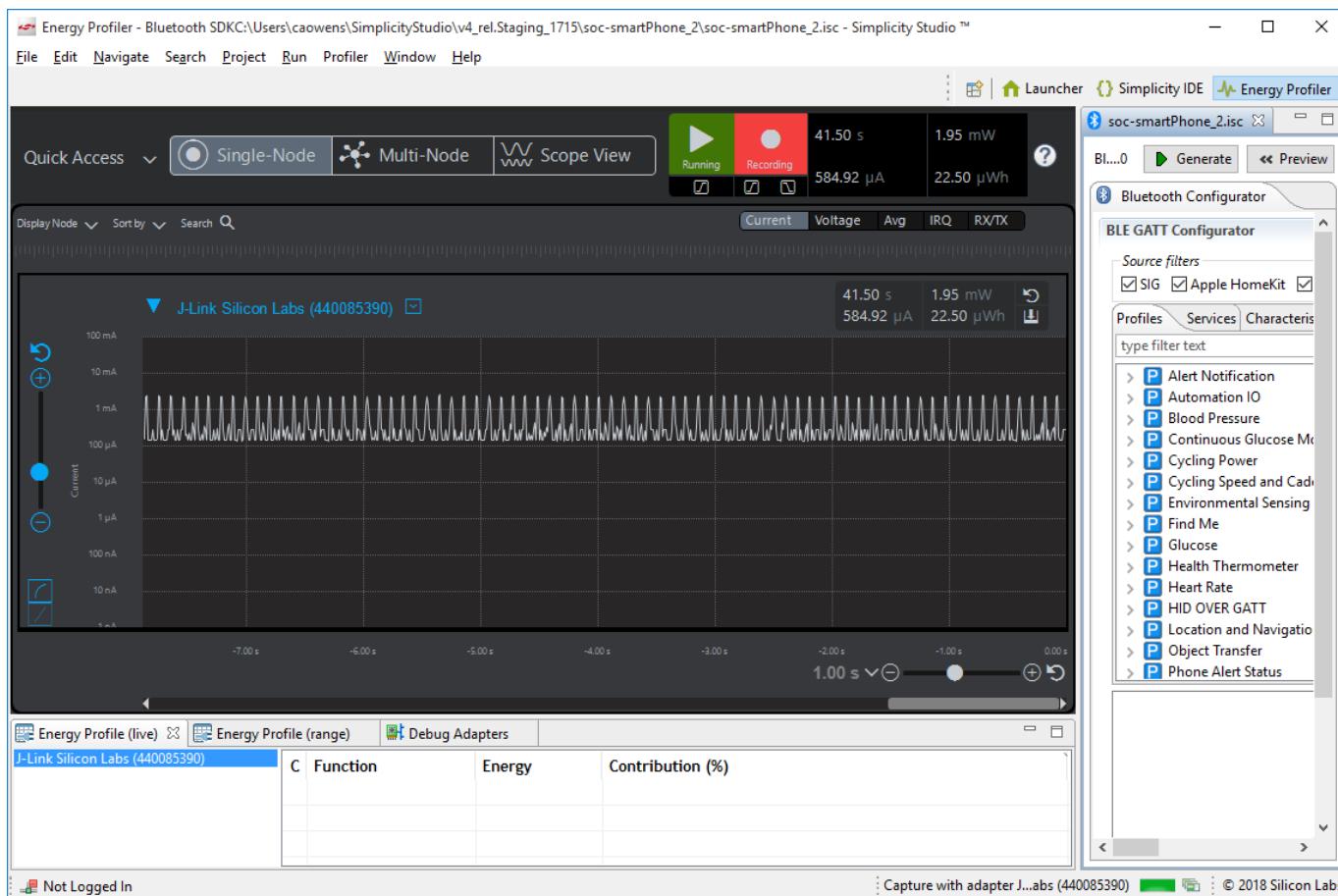
After clicking the selected item, the layout will be updated. After saving the file, the configuration source codes will be automatically generated. For more information see the [Simplicity Studio 5 User's Guide](#).

### 6.3 Multi-Node Energy Profiler

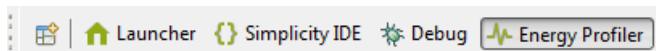
Multi-Node Energy profiler is an add-on tool, with which you can easily measure the energy consumption of your device in runtime. You can easily find peak and average consumption, and check for sleep mode current.

**Note:** The SDK sample apps for EFR32BG22 enable EM2 debug (see `init_mcu.c`), which adds current consumption overhead compared to the datasheet values.

To profile the current project, click Tools in the menu bar and select Energy Profiler or right-click on the `<project>.slcp` file in the Project Explorer view and select **Profile as / Simplicity Energy Profiler target**. This automatically builds your project, uploads it to the device, and starts Energy Profiler. A new Energy Profiler perspective appears, shown in the following figure.



See [UG343: Multi-Node Energy Profiler User's Guide](#) for details on how to use this tool. You can switch easily between Simplicity IDE and Energy Profiler perspectives using the Perspective buttons in the upper right corner of your current perspective.

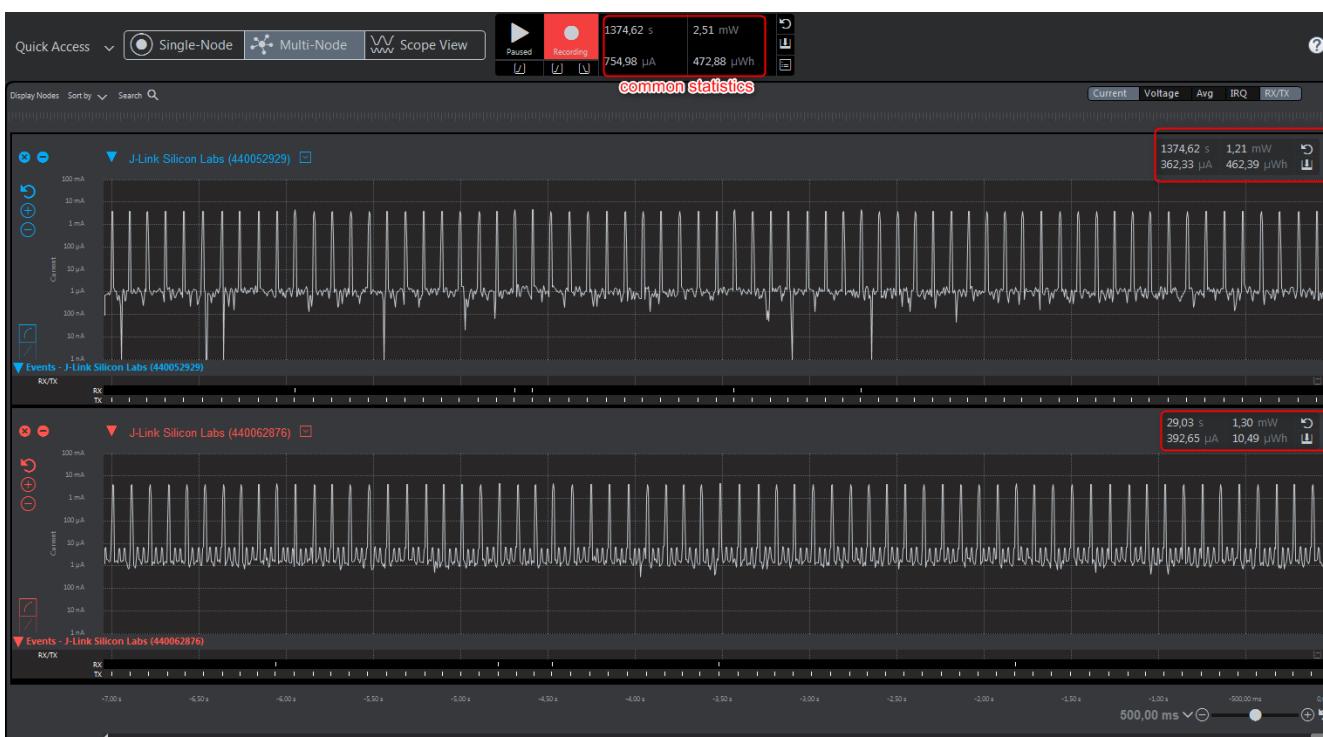


You can see peaks in the energy consumption diagram. Pause profiling by clicking Play, click one of the peaks, and zoom in with time axis (y-axis) zoom until you see three distinguishable peaks. These represent the three advertisement packets sent on the three advertisement channels. You can also see the three corresponding Tx events in the Rx/Tx bar below, provided that you enabled Rx/Tx view in the upper right corner. Note that the maximum consumption may now be greater than it appeared on the diagram before you zoomed in. This is because in zoomed-out mode, the displayed values are averaged. If you need exact values, always zoom in.

To measure average consumption, simply click and drag your mouse over a time interval. A new window appears in the upper right corner showing consumption information for the given interval. Bluetooth communication typically has a periodicity: the advertisement or the connection interval. It is recommended to measure average over an advertisement or connection interval to obtain a proper average consumption. Overall average is measured as well, but this is influenced by transient events.



Multi-node Energy Profiler is also able to simultaneously measure the consumption of multiple devices. To start measuring a new device click the Quick Access menu (upper left corner) and select **Start Energy Capture**. To stop measuring, click the Quick Access menu, and select **End/Save session**.

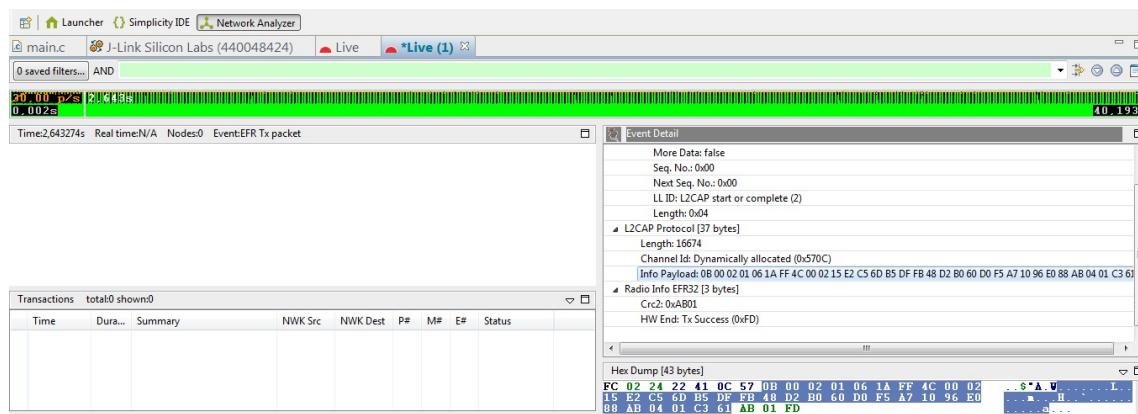


To learn more about how to use this tool, see the [Simplicity Studio 5 User's Guide](#).

## 6.4 Network Analyzer

Silicon Labs Network Analyzer is a free-of-charge packet capture and debugging tool that can be used to debug Bluetooth connectivity between Wireless Geckos and other Bluetooth devices. It significantly accelerates the network and application development process with graphical views of network traffic, activity, and duration.

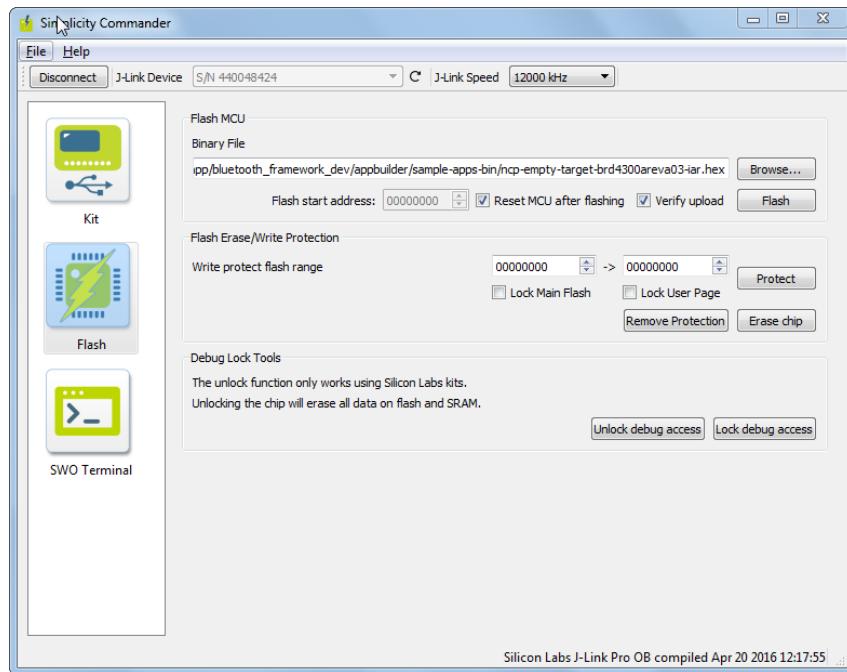
The Packet Trace application captures the packets directly from the Packet Trace Interface (PTI) available on the Wireless Gecko SoCs and modules. It therefore provides a more accurate capture of the packets compared to air-based capture.



**Figure 6.1. Bluetooth Traffic Capture with Packet Trace**

## 6.5 Simplicity Commander

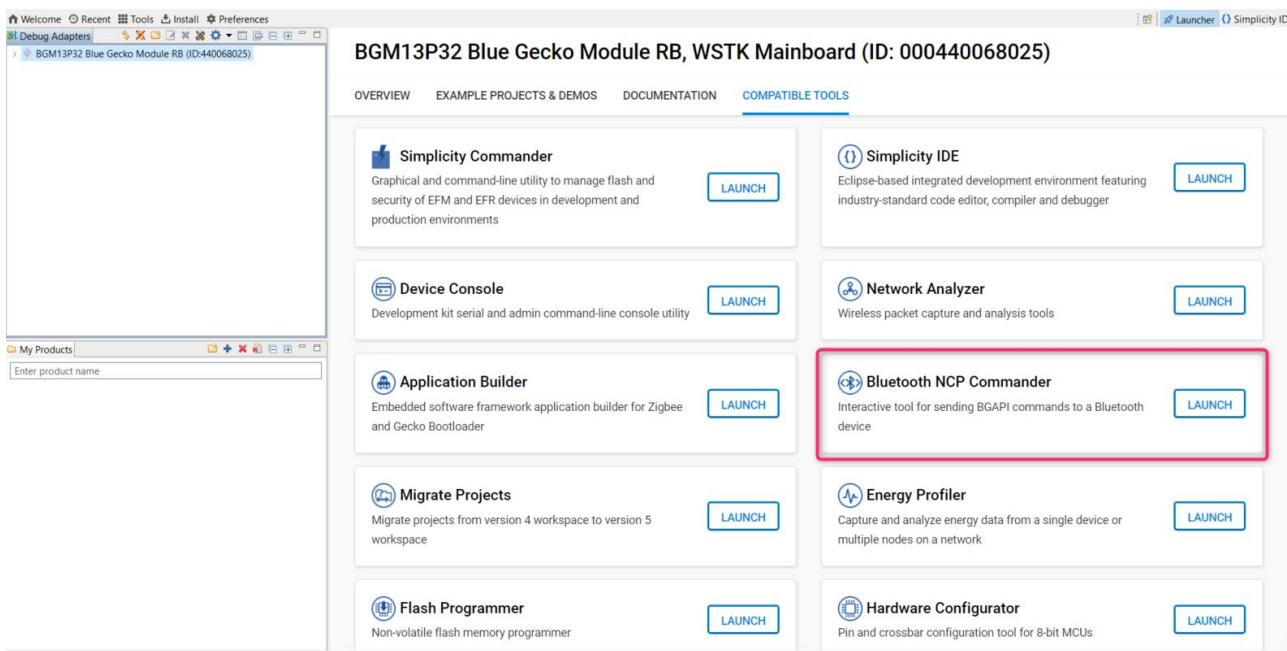
Simplicity Commander is a simple flashing tool, which can be used to flash firmware images, erase flash, lock and unlock debug access, and write-protect flash pages via the J-Link interface. Both GUI and CLI (Command Line Interface) are available. See *UG162: Simplicity Commander Reference Guide* for more information.



**Figure 6.2. Simplicity Commander**

## 6.6 Bluetooth NCP Commander

The Bluetooth NCP Commander application can be used to test and evaluate Bluetooth SoCs and modules, and it can be used to control the Bluetooth hardware using the BGAPI Serial Protocol (NCP) over a Serial/UART interface. See *AN1259: Using the Silicon Labs v3.x Bluetooth® Stack in Network Co-Processor Mode* for more information.



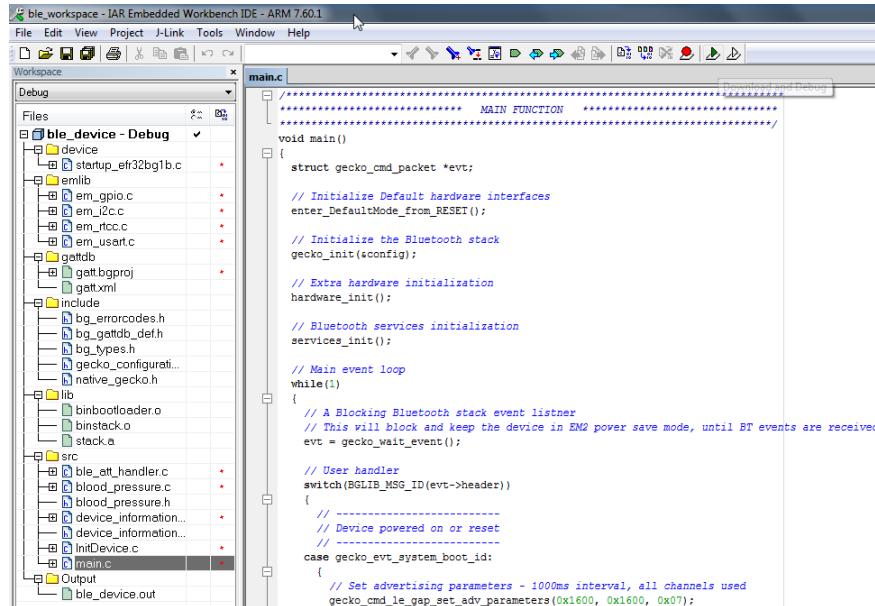
**Figure 6.3. NCP Commander Application**

Bluetooth NCP Commander is also available as a standalone application, located in:

C:\SiliconLabs\SimplicityStudio\v5\developer\adapter\_packs\ncp\_commander

## 6.7 IAR Embedded Workbench

IAR's Embedded Workbench can also be used as an IDE for developing and debugging Bluetooth applications. You must use the version of IAR that is compatible with the SDK version. See the SDK's release notes for compatible version information.



**Figure 6.4. IAR Embedded Workbench**

# Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



## IoT Portfolio

[www.silabs.com/IoT](http://www.silabs.com/IoT)



## SW/HW

[www.silabs.com/simplicity](http://www.silabs.com/simplicity)



## Quality

[www.silabs.com/quality](http://www.silabs.com/quality)



## Support & Community

[www.silabs.com/community](http://www.silabs.com/community)

### Disclaimer

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

### Trademark Information

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