AN1255: Transitioning from the v2.x to the v3.x *Bluetooth*® SDK

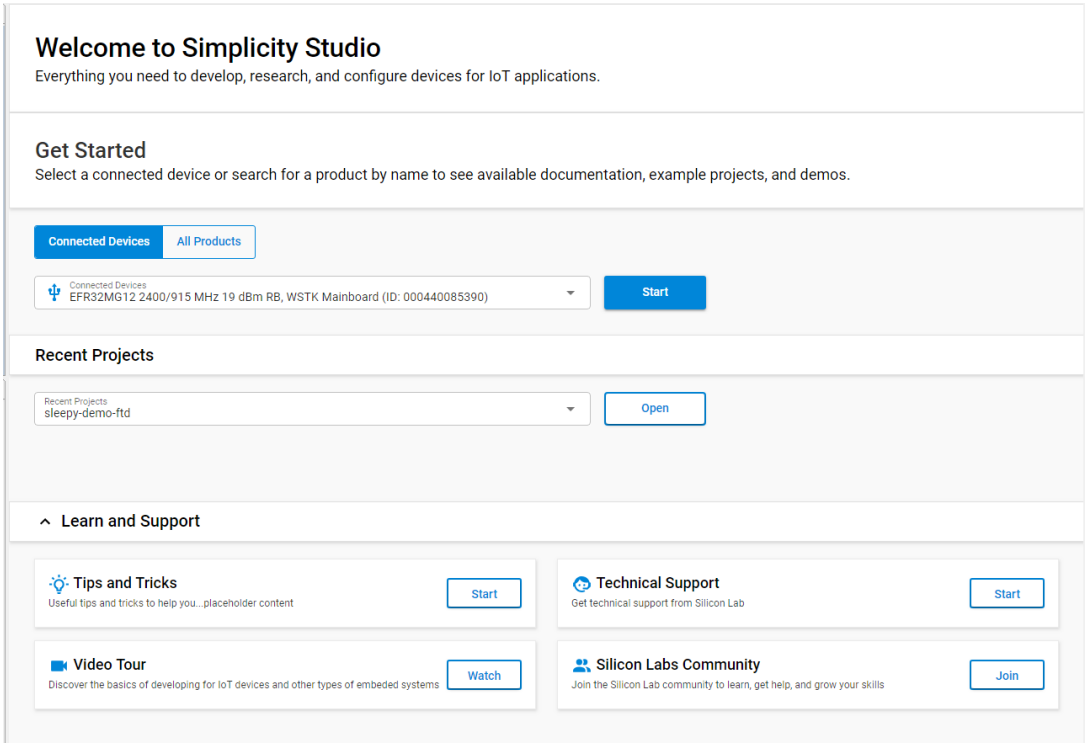
Bluetooth Software Development Kit (SDK) v3.0 contains a number of changes compared to Bluetooth SDK v2.x. Many of these changes are due to an underlying framework redesign that results in an improved developer experience within the new Simplicity Studio 5. Projects are now built on a component architecture. Simplicity Studio 5 includes project configuration tools that provide an enhanced level of software component discoverability, configurability, and dependency management. These include a Component Editor, and a redesigned GATT configurator.

In Bluetooth SDK v3.0, sample applications have a new software architecture, the Bluetooth API is updated, and the GATT configurator is completely redesigned. Additionally the stack can now be configured in separate header files, and platform components can be added to the project with the Component Editor instead of copying and including files manually. While these changes are a result of overall improvements in the SDK and in Simplicity Studio 5 it also means that migrating projects from Bluetooth SDK v2.x to v3.0 is not trivial. This document explains the steps needed to move your Bluetooth v2.x project into the v3.0 environment.

KEY POINTS

Reviews differences in:

- Software architecture
- The API
- GATT configurator
- Stack configuration
- Adding platform components



1 Introduction

Silicon Labs has introduced both a complete update to its Simplicity Studio tool suite, as well as a new, component-based Gecko Platform architecture.

Version 5 of the Simplicity Studio tool suite represents a ground-up redesign of the underlying architecture. The Simplicity Studio 5 (SSv5) framework includes a new user interface engine that enables attractive and responsive web-like user interfaces. The integrated development environment (IDE) was also upgraded with the latest versions of Eclipse and the C/C++ Development Tooling (CDT). This added robustness, performance improvements, and enables developers to customize their experience using the latest plug-ins from the Eclipse Marketplace.

Gecko Software Development Kit suite version 3.0.0.0 (GSDK v3.0) is released with SSv5. It introduces a new underlying Gecko Platform architecture based on components. Both the Bluetooth and the Flex v3.x SDKs are based on this new component-based architecture. With SSv5 and GSDK v3.0, Bluetooth developers will benefit from the following component-based project configuration features:

- Search and filter to find and discover software components that work with the target device
- Automatically pull in all component dependencies and initialization code
- Configurable software components including peripheral inits, drivers, middleware, and stacks
- All configuration settings in C header files for usage outside of Simplicity Studio
- Configuration validation to alert developers to errors or issues
- Easily manage all project source via git or other SCM tools
- Managed migration to future component and SDK versions
- Simplified transitions from Silicon Labs development kits to custom hardware

Other features of the SSv5/GSDK v3.0 development environment include:

- Project source management options (link to SDK sources or copy all contents to user folder)
- Graphical pin configuration
- Redesigned Bluetooth Configurator with a fresh UI that's more intuitive for Bluetooth and GATT customization
- Redesigned Radio Configurator with a fresh UI that's more intuitive for single- and multi-PHY customization
- Iterative development (configure components, edit sources, compile, debug) using SSv5 configuration tools and third-party IDEs
- GNU makefiles as a build option

Other changes are specific to the SDK. Bluetooth SDK v3.0 contains a number of changes compared to Bluetooth SDK v2.x. The main changes are as follows:

1. The project structure of the sample applications has changed. Many autogenerated files and unified configuration files help adding and configuring software components.
2. The entire project generation is now based on software components. This makes it possible to add functionality to the project with only a click, instead of copying files manually and looking for dependencies.
3. The sample applications have a completely new software architecture to align with the concept of software components. Adding RTOS to your project is also simplified.
4. The Bluetooth API has changed. The API commands and events use a new nomenclature to comply with Silicon Labs standards. Additionally, some new classes and commands are introduced and some of them are removed to make the API more transparent and consistent.
5. The GATT configurator is completely redesigned. The new user interface is more modern, while the generator tool makes it possible to add partial extensions to the GATT database. This means that the GATT database can be easily extended programmatically.
6. The stack can now be configured in a separate header file (instead of main.c). A Component Configurator tool in SSv5 is available to configure all the parameters, which makes it easy to use predefined values, and to validate custom values.
7. The AppLoader application has new features which requires the update of the bootloader.

This document guides you through these changes, and describes the migration steps necessary to move a project from Bluetooth SDK v2.x to v3.0.

For details about the release, see the release notes provided with the SDK.

2 Project Structure

In Bluetooth SDK v2.x a Bluetooth project contains the following folders:

/app	Application specific files
/hardware	Development board configuration files and drivers for external peripherals
/platform	Device configuration files and drivers for the MCU peripherals
/protocol	Bluetooth stack files
/util	Utilities

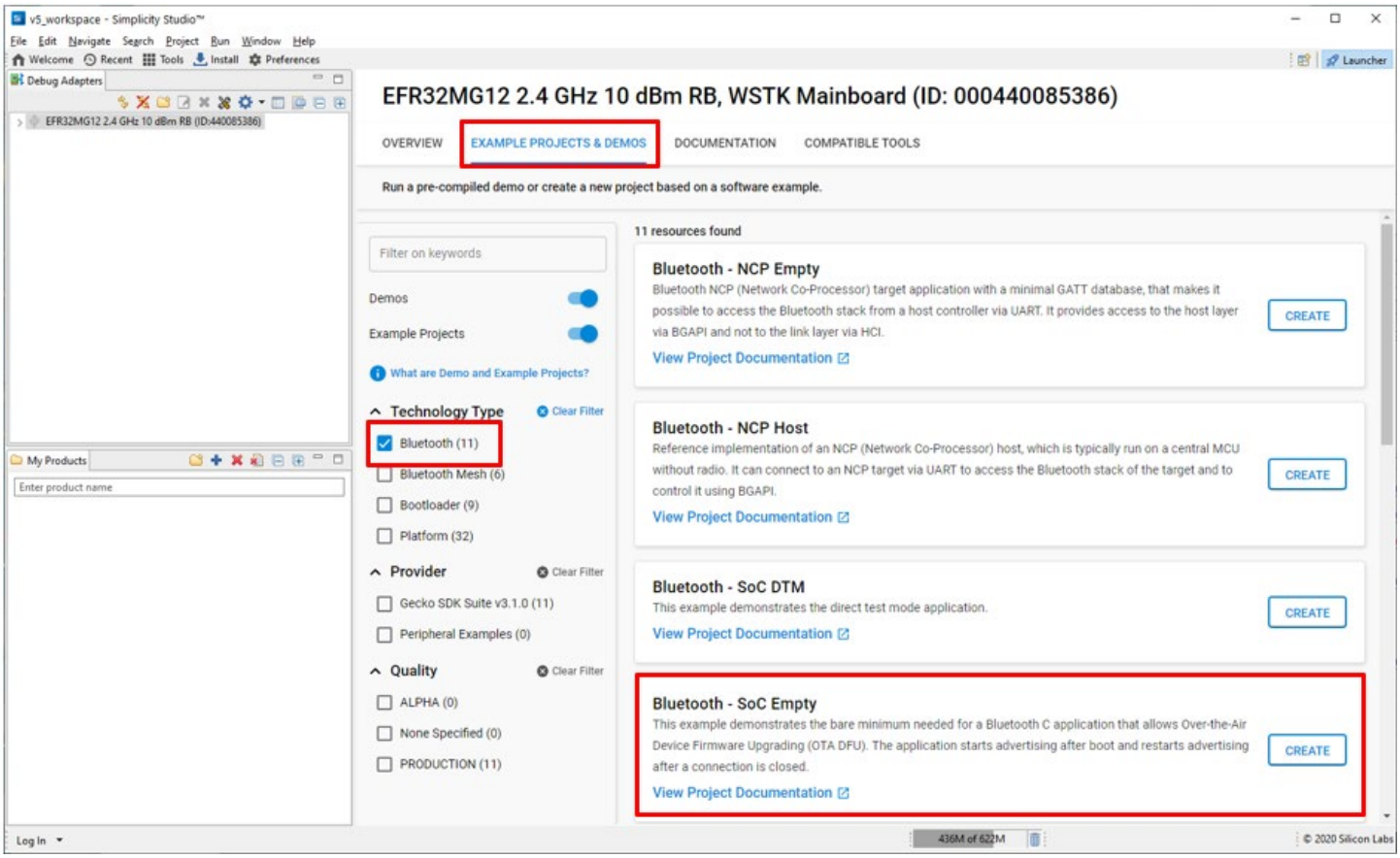
When a new project is generated, a subset of the SDK files (source files, headers, configuration headers) is copied from the SDK folder into the project folders based on hardware type and on the needs of the sample application. Additionally, some files (for example init_mcu.c, hal-config.h, and so on) are generated from templates into the root folder of the project. Again, the output is based on the hardware type and on the needs of the sample application.

In Bluetooth SDK v3.0 auto-generated files and configurations headers are clearly separated from the static SDK files and gathered into separate folders:

/autogen	Automatically generated files based on the installed software components
/config	Editable configuration files for the software components
/gecko_sdk_3.0.0	
/app	Application specific files
/hardware	Development board configuration files and drivers for external peripherals
/platform	Device configuration files and drivers for the MCU peripherals
/protocol	Bluetooth stack files
/util	Utilities

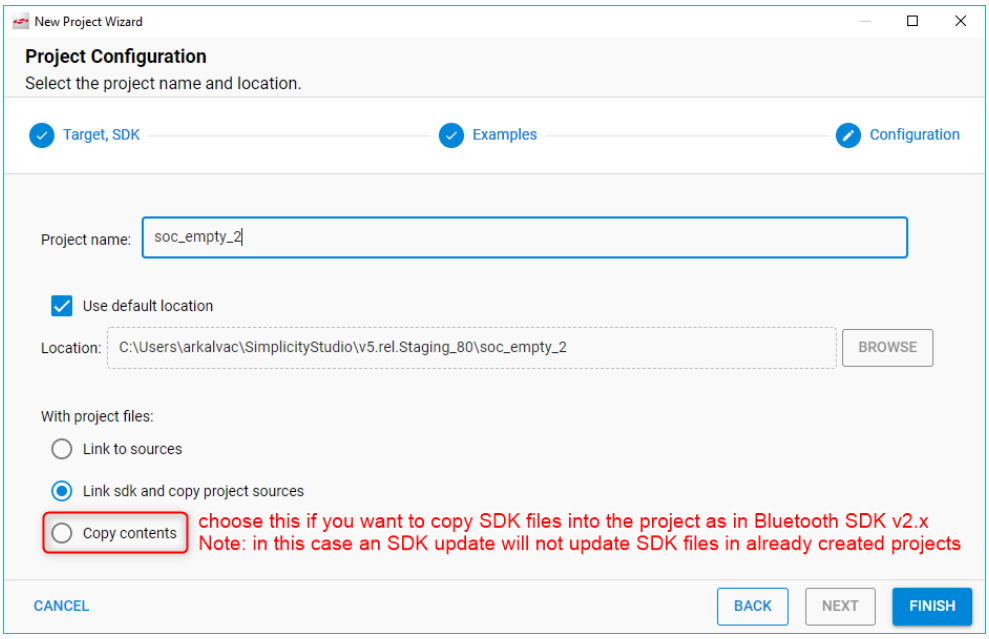
Developers can now easily see which files can be modified by the generator script and it is easy to access all configuration files. This is especially important, because in Bluetooth SDK v3.0 many more files are generated by the addition of software components (see Section 3 Software Components).

Due to the new project structure – and due to the new auto-generated files required by the Bluetooth stack – migrating a v2.x project into Bluetooth SDK v3.0 must begin by creating a new SoC-Empty project in the new SDK. Application logic should be pulled into this new project. After you select a compatible part on SSv5's Welcome page, an SoC-Empty project can be created from the Part-specific Launcher perspective. The Technology filter makes it easy to find applicable projects.

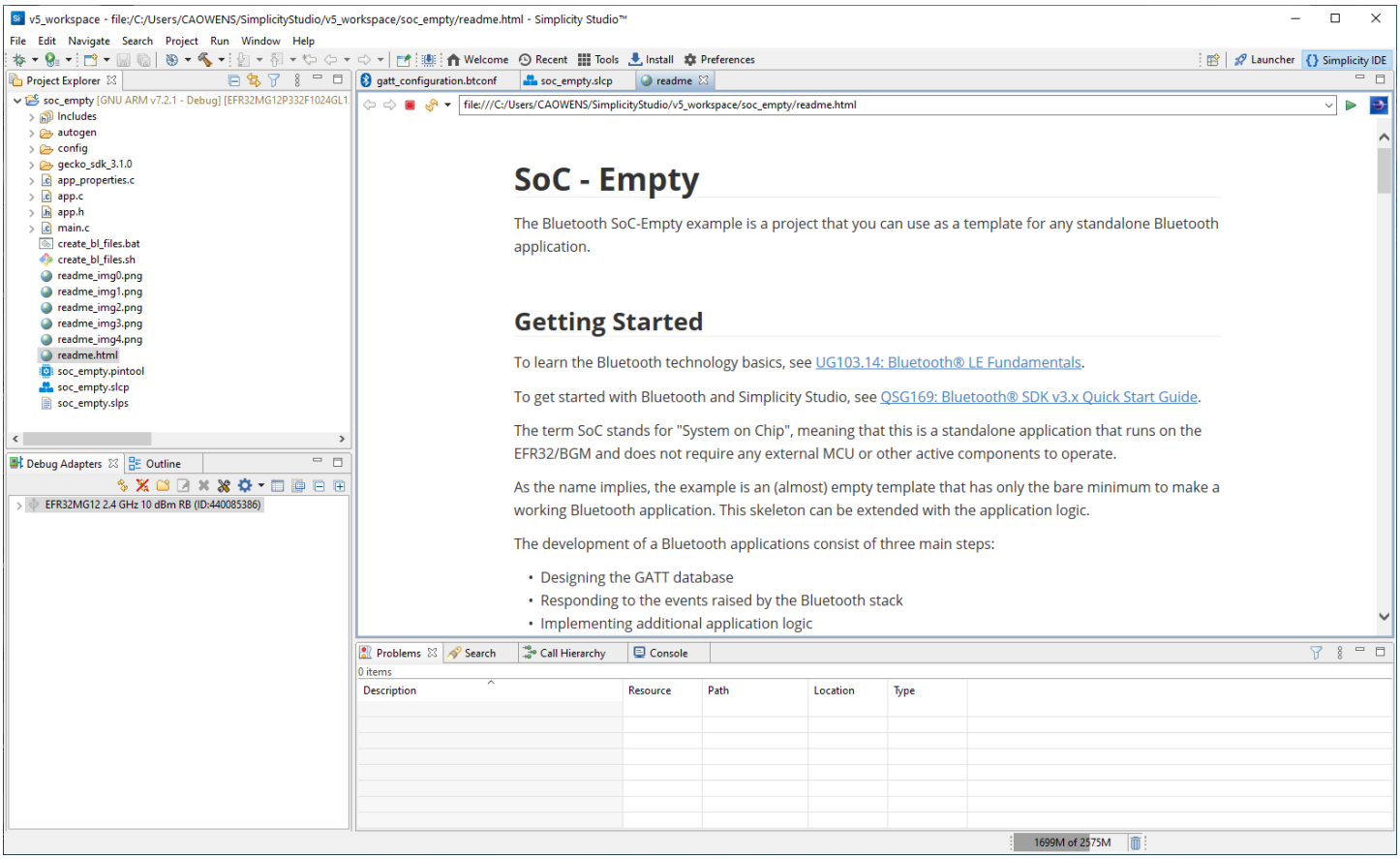


In the project configuration dialog you can rename the project, change the project location, and define how to handle project files.

Note: In Bluetooth SDK v2.x all SDK files are copied into the project by default. In Bluetooth SDK v3.0 SDK files, considered to be static, are linked by default. If you want to version-control your full project, it is recommended to change this setting to “Copy contents” in the Project Configuration wizard, when you create the new project to have all content needed by the project in one folder.



Bluetooth Projects automatically open on a ‘readme’ tab that describes the example. Other tabs are the Project Configurator (<project name>.slcp and the GATT configurator (gatt_configuration.btconf) (see section 6 GATT Configurator for more information).



3 Software Components

To add a new software component, for example a UART driver, in Bluetooth SDK v2.x the user must:

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.

And additionally:

4. Write the initialization code manually in the application.
5. Configure the component manually in the config files.
6. Use the API of the component in the application.

This is quite a cumbersome process, especially when figuring out the dependencies between components.

In Bluetooth SDK v3.0 software components can be added easily by installing them from the Component Library. The installation process will automatically execute the first three steps listed above, and it also modifies the corresponding auto-generated files to integrate the component into the application (*"glue logic"*).

Additionally the Component Configurator provides the possibility of:

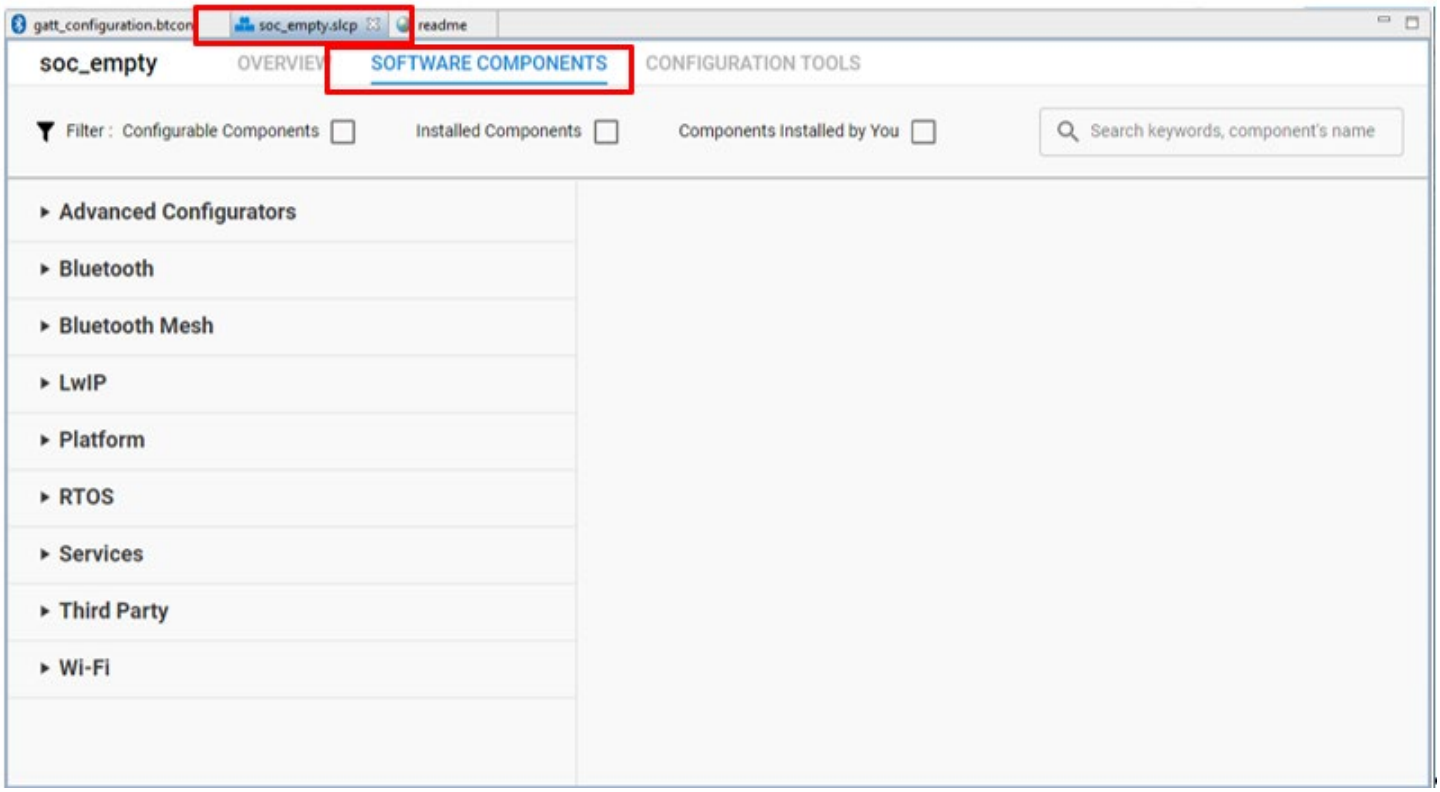
1. Adding an "*init*" type component that initializes the software component
2. Configuring the component with a GUI

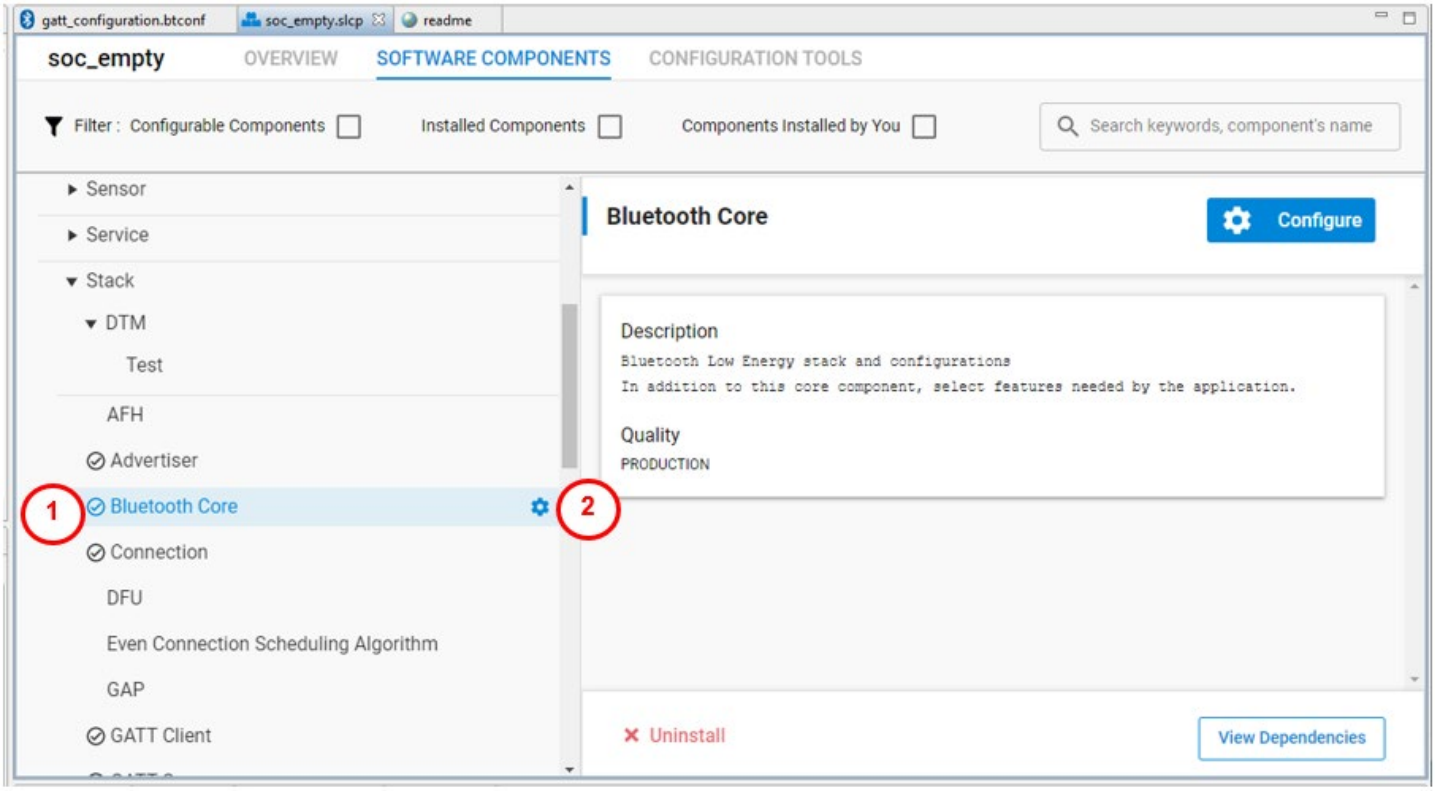
Some software components (like OTA DFU) will fully integrate into the application to perform a specific task without the need of any additional code, while other components provide an API to be used in the application. The only task left for the developer is to use the API of those component in the application.

It is important to note that in Bluetooth SDK v3.0 the Bluetooth stack itself is also just a collection of software components that can be added to and removed from the project.

When migrating a project into Bluetooth SDK v3.0, start by finding out which functionality can be provided by installing a software component from the Component Library. Although this means you must become familiar with the software components first, it will save time later. Generally speaking, if you would have copied an additional SDK file into your project, you will probably find a software component that solves the integration of that file.

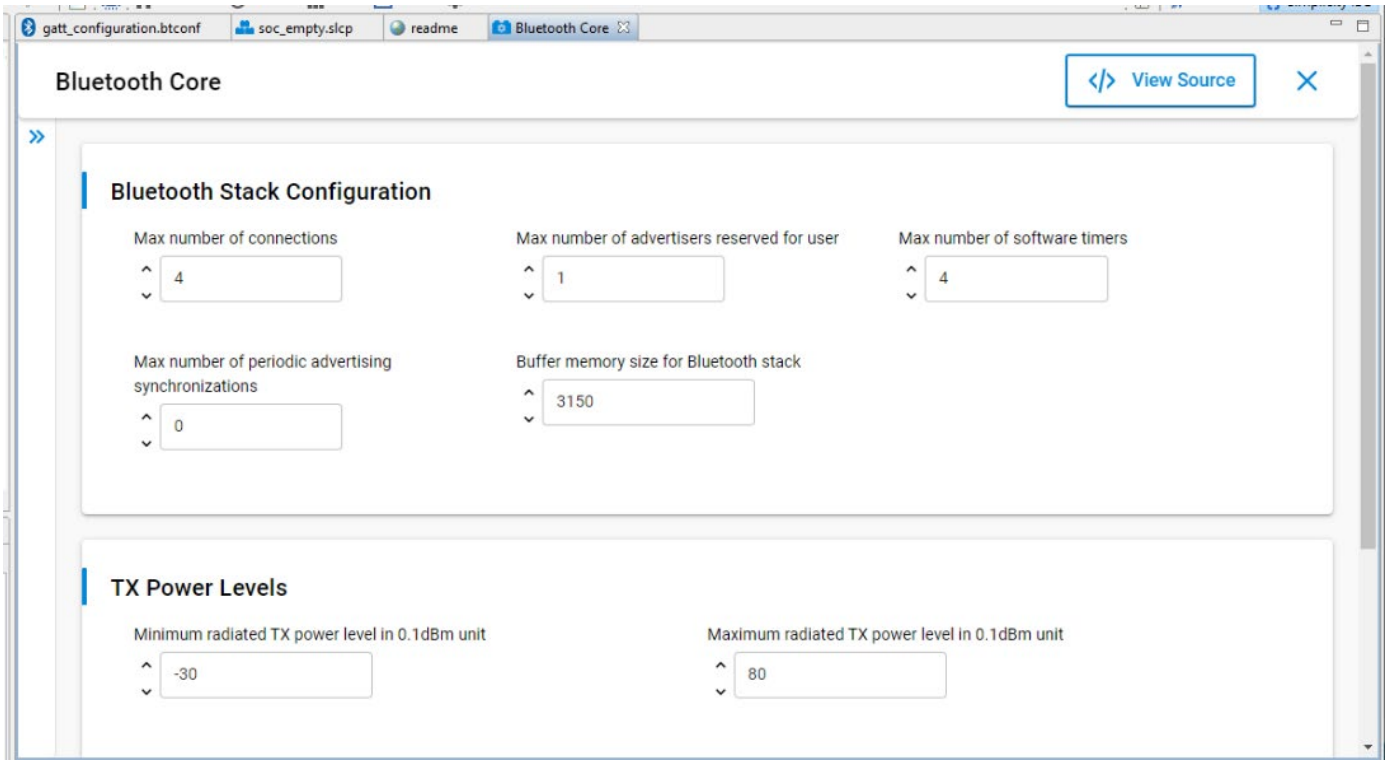
To see the component library, click the <project-name>.slcp tab of your project, and click **Software Components**. A number of filters as well as a keyword search are available to help you explore the various component categories. Note that components for all installed SDKs are presented.





Components installed in the project are checked (1), and can be uninstalled. Configurable components are indicated by a gear symbol (2). To display only installed components or only configurable components, use the checkboxes at the top of the editor.

Click **Configure** to see a configurable component's parameters in the Component Configurator.



See QSG169: Bluetooth® SDK v3.x Quick Start Guide and the online Simplicity Studio 5 User's Guide for more information about the SSv5 project interface and the Bluetooth SDK.

4 Software Architecture

4.1 Bluetooth Event Handling

In Bluetooth SDK v2.x the sample applications were written with simplicity in mind. Although many platform files are included in the project – to make the features provided by the hardware available for the application – the application itself has a simple and transparent architecture. Taking the SoC-Empty project as an example:

- The stack configuration is defined in main.c
- MCU, board and app are initialized in main.c
- Stack is initialized in app.c
- There is a simple loop in app.c to fetch and handle Bluetooth events

main.c

```
int main(void)
{
  /* Initialize device */
  initMcu();
  /* Initialize board */
  initBoard();
  /* Initialize application */
  initApp();

  /* Start application */
  appMain(&config);

}
```

app.c

```
void appMain(gecko_configuration_t *pconfig)
{
  /* Initialize stack */
  gecko_init(pconfig);

  while (1) {
    /* Event pointer for handling events */
    struct gecko_cmd_packet* evt;

    /* Check for stack event. */
    evt = gecko_wait_event();

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

      case gecko_evt_system_boot_id:
        printLog("system booted\r\n");
        break;

      case gecko_evt_le_connection_opened_id:
        printLog("connection opened\r\n");
        break;

      default:
        break;
    }
  }

}
```

This software architecture is very easy to understand. However, it has drawbacks:

- Sleep is managed by the Bluetooth stack. `gecko_wait_event()` puts the device into sleep mode, and processing continues only when a Bluetooth event is triggered. If your application implements processes that should run regularly you must rely on the Bluetooth stack software timers.
- The application is totally Bluetooth-event-driven. If your application needs custom events, you have to use the external signals of the Bluetooth stack to extend the Bluetooth events with custom events.
- If your application needs a state machine, it may be challenging to merge the states of the state machine with the Bluetooth event handlers / connection states.
- Since different software components (for example one that is reading your sensors, one that blinks the LEDs, and so on) may have different needs, you always have to adjust your software architecture to these needs.
- While all of these problems can be addressed by using RTOS, RTOS may consume too many resources for your application.

In contrast to this, the sample applications of Bluetooth SDK v3.0 were written with flexibility in mind. Although this makes the code a bit less transparent, at the same time it makes it easy to extend the software with a new software component.

- Each software component has an `init` and a `process` function. The process functions are run in an infinite loop, and the device is sent into sleep mode when neither of them needs the CPU. Sleep is automatically handled by the power manager. Here, the Bluetooth stack is just one of the software components, it does not have a distinguished role.
- To regularly run a process the common sleeptimer can be used instead of the soft timers of the Bluetooth stack.
- When an interrupt wakes up the device, you do not need to trigger an external signal to get out of the `wait_event` phase. All the process functions will be called and can catch their respective events triggered by interrupts.

Furthermore:

- Due to the unified software architecture software components can easily be integrated into the code by adding the proper function calls to predefined places in the code. This process is automatically handled by the Component Editor, which makes the addition of a software component as easy as ticking a checkbox.
- Software components define their dependencies, therefore whenever a new component is added, all of its dependencies are also automatically added to your project.
- In the new software architecture all configuration is put into header files. On one hand this separates functionality from config, and on the other hand it makes it possible to provide a Component Editor for all configurations. Software components, including the Bluetooth stack, can be easily configured within SSv5.

For these reasons, the new SoC-Empty example project is a bit more complicated. Main.c calls a generic `init` and `process` function, which then calls the `init` and `process` functions of each installed component. The Bluetooth component implements the Bluetooth event fetching in its own `process` function and ultimately calls the event handler function (`sl_bt_on_event`), which is to be implemented in the application (app.c). Note: some components (like OTA DFU) may also handle some Bluetooth events before the application.

main.c

```
int main(void)
{
  sl_system_init();
  app_init();

  while (1) {
    sl_system_process_action();
    app_process_action();
    sl_power_manager_sleep();
  }
}
```

sl_system_process_action.c

```
void
sl_system_process_action(void)
{
  sl_platform_process_action();
  sl_service_process_action();
  sl_stack_process_action();
sl_internal_app_process_action();
}
```

sl_event_handler.c

```
void
sl_stack_process_action(void)
{
  sl_bt_step();
}
```

sl_bluetooth.c

```
void sl_bt_step()
{
  sl_bt_msg_t evt;

  sl_status_t status =
       sl_bt_pop_event(&evt);

  if(status != SL_STATUS_OK){
    return;
  }
  sl_bt_process_event(&evt);
}

void
sl_bt_process_event(sl_bt_msg_t
*evt)
{
  sl_bt_ota_dfu_on_event(evt);
  sl_bt_on_event(evt);
}
```

app.c

```
void
sl_bt_on_event(sl_bt_msg_t*
evt)
{
  // Handle stack events
}
```

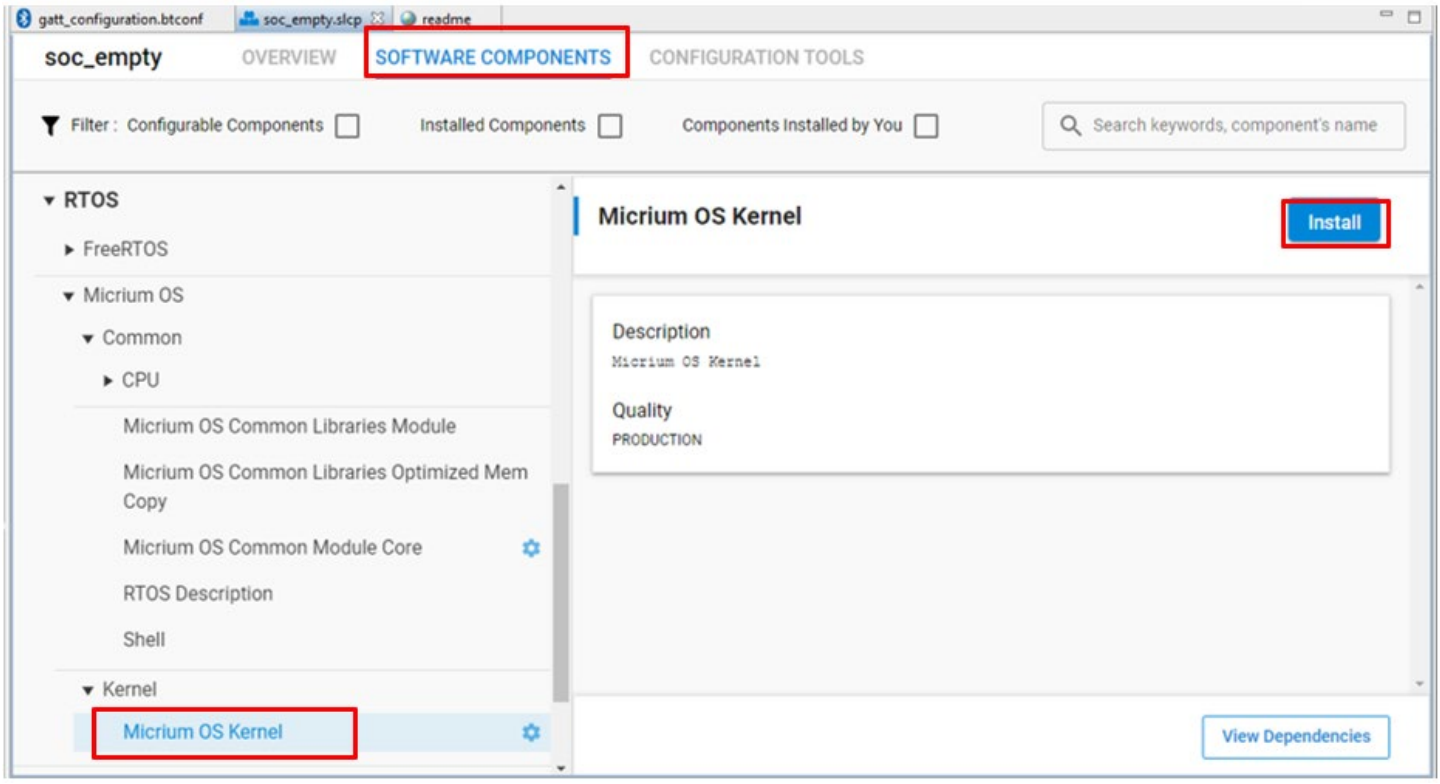
However, in the end implementing Bluetooth handlers is the same as in Bluetooth SDK v2.x. You can add all your event handlers in a switch-case statement of `sl_bt_on_event()` defined in app.c:

```
void sl_bt_on_event(sl_bt_msg_t* evt)
{
  // Handle stack events
  switch (SL_BT_MSG_ID(evt->header)) {

    case sl_bt_evt_system_boot_id:
      app_log("System booted\r\n");

      break;


    case sl_bt_evt_connection_opened_id:
      app_log("Connection opened\r\n");

      break;
    default:
      break;
  }
}
```

When migrating a Bluetooth application from v2.x to v3.0, copy the Bluetooth event handlers from appMain() into sl_bt_on_event().

4.2 RTOS

The new unified software architecture is written so that adding RTOS to your application is as simple as possible. For example, the Micrium OS Kernel can be added to your project with a click:



The Bluetooth event handler function looks the same as in the RTOS-less version, therefore migrating your RTOS-based application is as easy as migrating your RTOS-less application. Move the Bluetooth event handlers into `sl_bt_on_event()` defined in app.c. The application task, which runs parallel to the Bluetooth event handler task, must be started in `app_init()`, which is also defined in app.c:

```
SL_WEAK void app_init(void)
{
  RTOS_ERR err;

  OSTaskCreate(&myAppTaskTCB,
               "MY App Task",
               myAppTask,
               0u,
               MY_APP_TASK_PRIO,
               &myAppTaskStk[0u],
               (MY_APP_TASK_STACK_SIZE / 10u),
               MY_APP_TASK_STACK_SIZE,
               0u,
               0u,
               0u,
               (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
               &err);

}
```

5 Bluetooth API

The Bluetooth API changes in Bluetooth SDK v3.0. The most apparent change is the renaming of all the BGAPI commands and events to align with the unified Silicon Labs coding standard. However, there are also new and removed commands and also new BGAPI classes to make the API more logical and transparent. This section describes all the changes related to the API.

5.1 Functionality Breaks

Advertising Set Management

While in Bluetooth SDK v2.x advertisement set handles were assigned by the developer arbitrarily, in Bluetooth SDK v3.0, advertising set allocation is managed by the stack. Before any advertising operations, use the API command `sl_bt_advertiser_create_set()` to create an advertising set. This command returns the handle of the created set. Thereafter, any advertiser commands (`sl_bt_advertiser_....`) performing advertising operations can be called by passing the assigned advertising set handle. If a command is called with a non-allocated handle, it will return an error. To free an advertising set, use command `sl_bt_advertiser_delete_set(handle)`.

OTA Configuration and OTA Advertising Data

OTA configuration (flags and OTA device name) is removed from Bluetooth configuration structure. To set OTA flags and the OTA device name, use `sl_bt_ota_set_configuration()` and `sl_bt_ota_set_device_name()` respectively. Use the new command `sl_bt_ota_set_advertising_data()` to set OTA advertising data.

Some devices have multiple antenna ports. If you do not use the default port, the appropriate port should be set by `sl_bt_ota_set_rf_path()` to ensure that the AppLoader uses the same port as the application.

5.2 Error Code Changes

In Bluetooth SDK v3.0 Bluetooth error codes (such as `bg_err_invalid_conn_handle`) become part of the unified SL_STATUS codes, which provide unique error codes for the whole software platform. This means that the values of the Bluetooth error codes change. New definitions (such as `SL_STATUS_INVALID_HANDLE`) are introduced to cover the new codes. You can, however, also use the old definitions in your code, as they are automatically mapped to the new values in the v3.0 SDK. In case your code contains hard-coded error codes, they must be changed. The Bluetooth error codes can now be found in sl_status.h.

5.3 Changes in the BGAPI Classes

The BGAPI classes are slightly restructured in Bluetooth SDK v3.0. Most importantly the `le_gap` class, which accommodated many functions, is split into three new classes: `gap`, `scanner` and `advertiser`. The legacy `hardware` class, which accommodated only two functions, is removed. The soft timer functions are moved into the `system` class. `le_gap`, `le_connection`, and `flash` classes are renamed as shown in Table 1. A new `ota` class is created to accommodate functions configuring OTA upgrade (with Apploader), since OTA configuration is now removed from the Bluetooth configuration structure.

Table 5-1. Changes in the BGAPI Classes

API 2.x	API 3.0	Notes
le_gap	gap	Prefix "le_" is redundant as the stack supports LE only.
-	scanner	Split from le_gap. Provides API for scanning functions. Enables better size optimization.
-	advertiser	Split from le_gap. Provides APIs for advertising functions. Enables better size optimization.
le_connection	connection	Prefix "le_" is redundant.
hardware	-	Removed. Soft timer commands are moved to system class.
flash	nvm	Renamed for aligning with functionality it provides.
-	ota	New class for OTA configurations

5.4 Changes in BGAPI Commands

BGAPI command functions change both their name, to align with Silicon Labs standards, and their structure, to make the error checking and the handling of return values simpler.

In Bluetooth SDK v3.0 BGAPI command function names start with `sl_bt_` instead of `gecko_cmd_` used in Bluetooth SDK v2.x. This means that all function name should be changed according to this rule, when migrating a project from v2.x to v3.0.

No compatibility layer is provided due to the additional changes listed below. However, a “Bluetooth API migration helper” component can be found in the Component Library. If this component is installed, a header file is added to the project that provides verbose compiler errors when an old API call is found in the code. A suggestion for the new API is also present in the error message.

In Bluetooth SDK v2.x command functions returned a complex structure, providing both error code and return values. In Bluetooth SDK v3.0 only a status code is returned, and the return values are passed back using pointer arguments. If the output of a command contains variable size data, the application needs to give the destination for the data as well as the maximum size of the destination. See an example below.

Command functions in v2.x

```
/* Function */
struct gecko_msg_gatt_server_read_attribute_value_rsp_t*
gecko_cmd_gatt_server_read_attribute_value(uint16 attribute, uint16 offset);

/* Response structure */
struct gecko_msg_gatt_server_read_attribute_value_rsp_t
{
  uint16 result,
  uint8array value
}
```

Command functions in v3.0

```
sl_status sl_bt_gatt_server_read_attribute_value(uint16 attribute, uint16 offset, size_t
max_value_size, size_t *value_len, void *value);
```

While for most commands the renaming means only changing `gecko_cmd_` to `sl_bt_`, many functions are renamed due to changed functionality, changed API class or simply to make the API more logical. Furthermore, some API functions are split into multiple ones, and some functions are merged. These name changes are listed in Table 5-2. Changes in the BGAPI Commands. For other functions not in this table, you only need to change the beginning of the function name from `gecko_cmd_` to `sl_bt_`.

The following table below was created based on Bluetooth SDK v3.0. If you are using Bluetooth SDK v3.1, additional changes apply. The additional changes are listed in the Bluetooth SDK 3.1.0.0 Release Notes.

Table 5-2. Changes in the BGAPI Commands

API 2.x	API 3.0	Notes
gecko_cmd_le_gap_enable_whitelisting	sl_bt_gap_enable_whitelisting	le_gap class renamed to gap
gecko_cmd_le_gap_set_data_channel_ classification	sl_bt_gap_set_data_channel_ classification	
gecko_cmd_le_gap_set_privacy_mode	sl_bt_gap_set_privacy_mode	
-	sl_bt_advertiser_create_set	New API for creating an advertising set. **Call this command to create an advertising set before any advertising operations on an advertising set. The returned handle of the created advertising can then be used in other commands for advertising operations. A created advertising set can be released by command sl_bt_advertiser_delete_set.**
-	sl_bt_advertiser_delete_set	New API for deleting an advertising set.

API 2.x	API 3.0	Notes
gecko_cmd_le_gap_bt5_set_adv_parameters	sl_bt_advertiser_set_timing sl_bt_advertiser_set_channel_map sl_bt_advertiser_set_report_scan_request	
gecko_cmd_le_gap_set_mode	sl_bt_advertiser_start sl_bt_advertiser_stop	
gecko_cmd_le_gap_bt5_set_mode	sl_bt_advertiser_start sl_bt_advertiser_stop sl_bt_advertiser_set_timing sl_bt_advertiser_set_configuration	
gecko_cmd_le_gap_set_adv_data gecko_cmd_le_gap_bt5_set_adv_data	sl_bt_advertiser_set_data sl_bt_ota_set_advertising_data	New command sl_bt_ota_set_advertising_data for setting OTA advertising data. sl_bt_advertiser_set_data no longer supports setting OTA advertising data.
gecko_cmd_le_gap_set_adv_parameters	sl_bt_advertiser_set_timing sl_bt_advertiser_set_channel_map	
gecko_cmd_le_gap_set_adv_timeout	sl_bt_advertiser_set_timing	
gecko_cmd_le_gap_set_advertise_timing	sl_bt_advertiser_set_timing	
gecko_cmd_le_gap_set_advertise_report_scan_request	sl_bt_advertiser_set_report_scan_request	
gecko_cmd_le_gap_set_advertise_phy	sl_bt_advertiser_set_phy	
gecko_cmd_le_gap_set_advertise_channel_map	sl_bt_advertiser_set_channel_map	
gecko_cmd_le_gap_set_advertise_configuration	sl_bt_advertiser_set_configuration	
gecko_cmd_le_gap_clear_advertise_configuration	sl_bt_advertiser_clear_configuration	
gecko_cmd_le_gap_set_advertise_tx_power	sl_bt_advertiser_set_tx_power	
gecko_cmd_le_gap_set_long_advertising_data	sl_bt_advertiser_set_long_data	sl_bt_set_long_data_set_long_data data does not support setting OTA advertising data.
gecko_cmd_le_gap_start_advertising	sl_bt_advertiser_start	
gecko_cmd_le_gap_stop_advertising	sl_bt_advertiser_stop	
gecko_cmd_le_gap_start_periodic_advertising	sl_bt_advertiser_start_periodic_advertising	
gecko_cmd_le_gap_stop_periodic_advertising	sl_bt_advertiser_stop_periodic_advertising	
gecko_cmd_le_gap_set_advertise_random_address	sl_bt_advertiser_set_random_address	
gecko_cmd_le_gap_clear_advertise_random_address	sl_bt_advertiser_clear_random_address	
gecko_cmd_le_gap_open gecko_cmd_le_gap_connect	sl_bt_connection_open	Moved to connection class.
gecko_cmd_le_gap_set_scan_parameters	sl_bt_scanner_set_timing sl_bt_scanner_set_type	
gecko_cmd_le_gap_set_discovery_timing	sl_bt_scanner_set_timing	
gecko_cmd_le_gap_set_discovery_type	sl_bt_scanner_set_mode	
gecko_cmd_le_gap_start_discovery gecko_cmd_le_gap_discover	sl_bt_scanner_start	
gecko_cmd_le_gap_end_procedure	sl_bt_scanner_stop	

API 2.x	API 3.0	Notes
gecko_cmd_le_gap_set_discovery_ extended_scan_response	-	Removed. A single event API (sl_bt_evt_scanner_scan_report) for advertising reports.
gecko_cmd_le_gap_set_conn_phy	sl_bt_connection_set_default_preferred_ phy	
gecko_cmd_le_gap_set_conn_ parameters gecko_cmd_le_gap_set_conn_timing_ parameters	sl_bt_connection_set_default_parameters	
gecko_cmd_le_connection_close	sl_bt_connection_close	
gecko_cmd_le_connection_disable_slave _latency	sl_bt_connection_disable_slave_latency	
gecko_cmd_le_connection_get_rssi	sl_bt_connection_get_rssi	
gecko_cmd_le_connection_read_channel _map	sl_bt_connection_read_channel_map	
gecko_cmd_le_connection_set_ parameters gecko_cmd_le_connection_set_timing_ parameters	sl_bt_connection_set_parameters	
gecko_cmd_le_connection_set_phy gecko_cmd_le_connection_set_preferred _phy	sl_bt_connection_set_preferred_phy	
gecko_cmd_system_get_bt_address	sl_bt_system_get_identity_address	
gecko_cmd_system_set_bt_address	sl_bt_system_set_identity_address	
gecko_cmd_system_set_tx_power	sl_bt_system_set_max_tx_power	
gecko_cmd_system_set_device_name	sl_bt_ota_set_device_name	
gecko_cmd_hardware_get_time	-	Use sleeptimer API
gecko_cmd_hardware_set_lazy_soft_ timer	sl_bt_system_set_lazy_soft_timer	Moved to system class
gecko_cmd_hardware_set_soft_timer	sl_bt_system_set_soft_timer	Moved to system class
gecko_cmd_flash_ps_erase	sl_bt_nvm_erase	Class renamed from flash to nvm
gecko_cmd_flash_ps_erase_all	sl_bt_nvm_erase_all	Class renamed from flash to nvm
gecko_cmd_flash_ps_load	sl_bt_nvm_load	Class renamed from flash to nvm
gecko_cmd_flash_ps_save	sl_bt_nvm_save	Class renamed from flash to nvm
gecko_cmd_<class_name>_ <cmd_name>	sl_bt_<class_name>_<cmd_name>	For rest of the commands that aren't mentioned in this table, prefix change from **gecko_cmd** to **sl_bt**

5.5 Changes in BGAPI events

Just like BGAPI commands, BGAPI events are also renamed. In Bluetooth SDK v3.0 event IDs start with `sl_bt_evt_` instead of `gecko_evt_`. Similarly, the event struct types start with `sl_bt_evt_` instead of `gecko_msg_evt_`, although these types are rarely referenced in the application.

Additionally, some events are renamed because of changing BGAPI class. Table 5-3 contains all the events that have more than just the name change.

Table 5-3. Changes in the BGAPI events

API 2.x	API 3.0	Notes
gecko_evt_le_gap_scan_response_id struct gecko_msg_le_gap_scan_response_evt_t	sl_bt_evt_scanner_scan_report_id sl_bt_evt_scanner_scan_report_t	Moved to scanner class
gecko_evt_le_gap_extended_scan_response_id struct gecko_msg_le_gap_extended_scan_response_evt_t	sl_bt_evt_scanner_scan_report_id sl_bt_evt_scanner_scan_report_t	Moved to scanner class
gecko_evt_le_gap_adv_timeout_id struct gecko_msg_le_gap_adv_timeout_evt_t	sl_bt_evt_advertiser_timeout_id sl_bt_evt_advertiser_timeout_t	Moved to advertiser class
gecko_evt_le_gap_scan_request_id struct gecko_msg_le_gap_scan_request_evt_t	sl_bt_evt_advertiser_scan_request_id sl_bt_evt_advertiser_scan_request_t	Moved to advertiser class
gecko_evt_hardware_soft_timer_id struct gecko_msg_evt_hardware_soft_timer_t	sl_bt_evt_system_soft_timer_id sl_bt_evt_system_soft_timer_t	Moved to system class
gecko_evt_le_connection_opened_id struct gecko_msg_evt_le_connection_opened_t	sl_bt_evt_connection_opened_id sl_bt_evt_connection_opened_t	Renamed to connection class
gecko_evt_le_connection_closed_id struct gecko_msg_evt_le_connection_closed_t	sl_bt_evt_connection_closed_id sl_bt_evt_connection_closed_t	Renamed to connection class
gecko_evt_le_connection_rssi_id struct gecko_msg_evt_le_connection_rssi_t	sl_bt_evt_connection_rssi_id sl_bt_evt_connection_rssi_t	Renamed to connection class
gecko_evt_le_connection_parameters_id struct gecko_msg_evt_le_connection_parameters_t	sl_bt_evt_connection_parameters_id sl_bt_evt_connection_parameters_t	Renamed to connection class
gecko_evt_le_connection_phy_status_id struct gecko_msg_evt_le_connection_phy_status_t	sl_bt_evt_connection_phy_status_id sl_bt_evt_connection_phy_status_t	Renamed to connection class
gecko_evt_<class_name>_<event_name>_id struct gecko_msg_<class_name>_<event_name>_evt_t	sl_bt_evt_<class_name>_<event_name>_id sl_bt_evt_<class_name>_<event_name>_t	For the rest of the events that aren't mentioned in this table

5.6 Enums and Defines

Due to the renamed classes some enumerations and definitions used by the Bluetooth API also change. The following table summarizes these changes.

Table 5-4. Changes in the BGAPI enumerations and definitions

API 2.x	API 3.0	Notes
le_gap_address_type_public le_gap_address_type_random le_gap_address_type_public_identity le_gap_address_type_random_identity	gap_public_address gap_static_address gap_random_resolvable_address gap_random_nonresolvable_address	Class renamed. le_gap_address_type_public_identity and le_gap_address_type_random_identity removed
le_gap_phy_1m le_gap_phy_2m le_gap_phy_coded	gap_1m_phy gap_2m_phy gap_coded_phy	Class renamed. enum items renamed.
le_gap_non_discoverable le_gap_limited_discoverable le_gap_general_discoverable le_gap_broadcast le_gap_user_data	advertiser_non_discoverable advertiser_limited_discoverable advertiser_general_discoverable advertiser_broadcast advertiser_user_data	Moved to advertiser class
le_gap_non_connectable le_gap_directed_connectable le_gap_undirected_connectable le_gap_connectable_scannable le_gap_scannable_non_connectable le_gap_connectable_non_scannable	advertiser_non_connectable advertiser_directed_connectable advertiser_connectable_scannable advertiser_scannable_non_connectable advertiser_connectable_non_scannable	Moved to advertiser class. le_gap_undirected_connectable removed. Corresponding enum value is advertiser_connectable_scannable
le_gap_discover_limited le_gap_discover_generic le_gap_discover_observation	scanner_discover_limited scanner_discover_generic scanner_discover_observation	Moved to scanner class.
FLASH_PS_KEY_CTUNE	NVM_KEY_CTUNE	

5.7 Changes in the C API

The Bluetooth API has some commands that are not part of the BGAPI classes. They can be used in SoC applications only, for example to fetch events. These commands are part of the so-called C API. Due to the unified nomenclature these commands are also renamed in Bluetooth SDK v3.0, and some of them will not exist anymore due to the new software architecture. Return values are also unified similar to the BGAPI commands.

Table 5-5. Changes in the C API

API 2.x	API 3.0	Notes
struct gecko_cmd_packet	sl_bt_msg_t	
BGLIB_MSG_ID	SL_BT_MSG_ID	
struct gecko_cmd_packet* **gecko_wait_event**()	sl_status_t **sl_bt_wait_event**(sl_bt_msg_t* evt)	In API 3.0, an event object is copied into the memory provided by application.
struct gecko_cmd_packet* **gecko_peek_event**()	sl_status_t **sl_bt_pop_event**(sl_bt_msg_t* evt)	
int **gecko_event_pending**()	bool **sl_bt_event_pending**()	
errorcode_t **gecko_stack_init**(const gecko_configuration_t *config)	sl_status_t **sl_bt_init_stack**(const sl_bt_configuration_t *config)	
errorcode_t **gecko_init**(const gecko_configuration_t *config)	-	Component Configurator calls the init functions. If not using SSv5, call sl_bt_init_stack() and the BGAPI class init functions
uint32_t **gecko_can_sleep_ms**()	-	Power manager takes care of sleeping
uint32_t **gecko_can_sleep_ticks**(void)	-	
uint32_t **gecko_sleep_for_ms**(uint32 max)	-	
void **gecko_priority_handle**(void)	void **sl_bt_priority_handle**(void)	
void **gecko_external_signal**(uint32 signals)	void **sl_bt_external_signal**(uint32_t signals)	
void **gecko_send_system_awake**()	void **sl_bt_send_system_awake**()	
void **gecko_send_system_error**(uint16 reason, uint8 data_len, const uint8* data)	void **sl_bt_send_system_error**(uint16_t reason, uint8_t data_len, const uint8_t* data)	
gecko_send_evt_user_message_to_host	**sl_bt_send_evt_user_message_to_host**	
gecko_send_rsp_user_message_to_target	**sl_bt_send_rsp_user_message_to_target**	

5.8 Migration Example

The following code snippets show an example how a v2.x application is to be updated to work in the v3.0 environment, considering all the changes mentioned above. The sample code simply starts advertising on boot and restarts advertising on connection close.

Code written in Bluetooth SDK v2.x

```
#include "native_gecko.h"

void appMain(gecko_configuration_t* pconfig)
{
 initLog();
 gecko_init(pconfig);

 while (1)
 {
  struct gecko_cmd_packet* evt;
  evt = gecko_peek_event();

  // Handle stack events
  switch (BGLIB_MSG_ID(evt->header)) {
    case gecko_evt_system_boot_id:
      gecko_cmd_le_gap_set_advertise_timing(0,
                                            160, 160,
                                            0, 0);


      gecko_cmd_le_gap_start_advertising(0,
                 le_gap_general_discoverable,
                 le_gap_connectable_scannable);


      printLog("Started advertising\n");
      break;


    case gecko_evt_le_connection_opened_id:
      printLog("Connection opened\n");
      break;


    case gecko_evt_le_connection_closed_id:
      printLog("Connection closed\n");
      if (boot_to_dfu) {
        // Enter to OTA DFU mode.
        gecko_cmd_system_reset(2);
      } else {
        gecko_cmd_le_gap_start_advertising(0,
                le_gap_general_discoverable,
                le_gap_connectable_scannable);
        printLog("Started advertising\n");
      }
      break;

    default:
      break;
  }
 }
}
```

Code in Bluetooth SDK v3.0

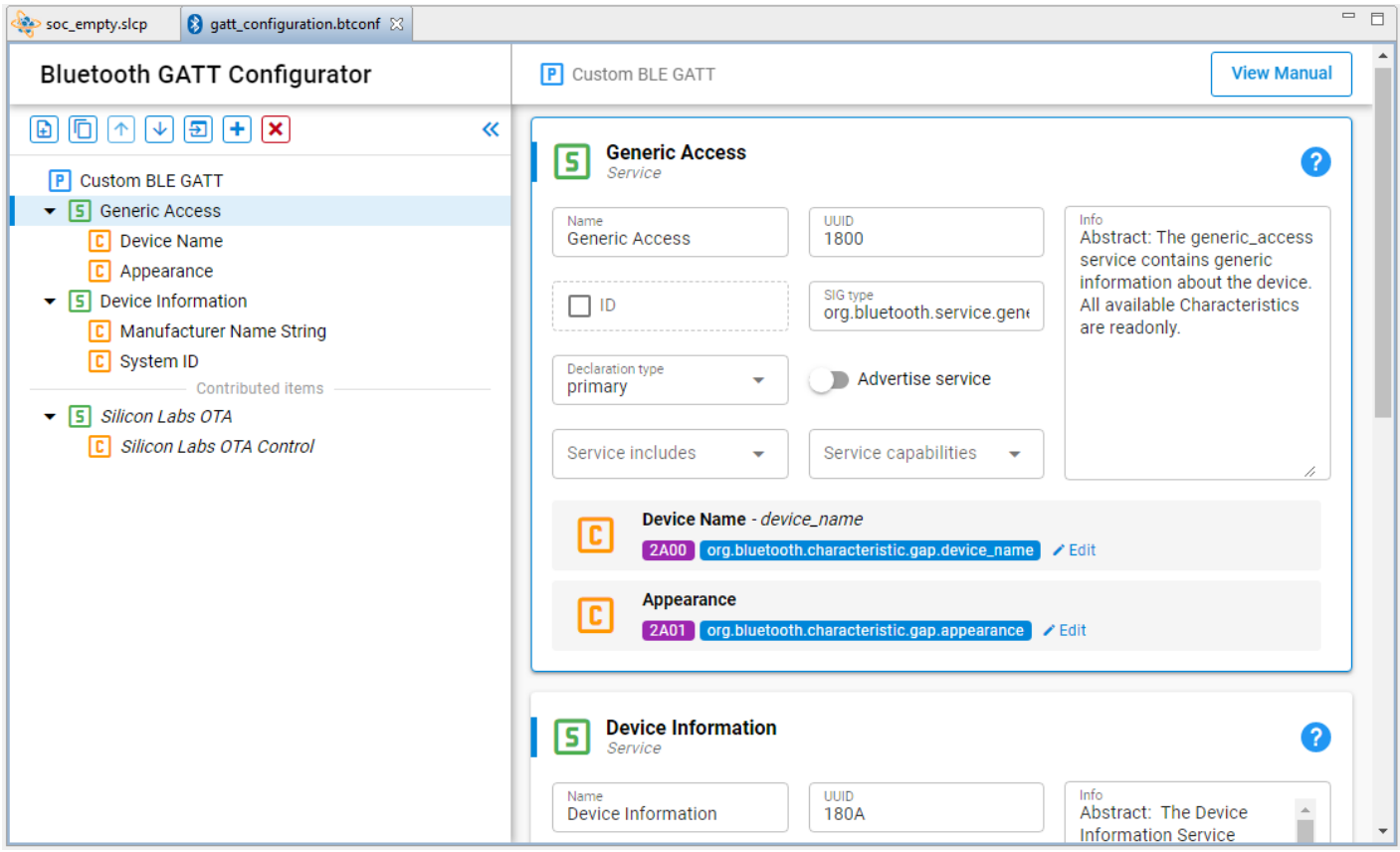
```
#include "sl_bt_api.h"
static uint8_t adv_h = 255; //adv. handle

void sl_bt_on_event(sl_bt_msg_t* evt)
{
  // Handle stack events
  switch (SL_BT_MSG_ID(evt->header)) {
    case sl_bt_evt_system_boot_id:
      err= sl_bt_advertiser_create_set(&adv_h);
      app_assert(err == SL_STATUS_OK,
      "[E: 0x%04x] Failed to create advertising
      set\n", (int)err);

      err = sl_bt_advertiser_set_timing(adv_h,
                                        160, 160,
                                        0, 0);
      app_assert(err == SL_STATUS_OK,
      "[E: 0x%04x] Failed to set advertising
      timing\n", (int)err);

      err = sl_bt_advertiser_start(adv_h,
             advertiser_general_discoverable,
             advertiser_connectable_scannable);
      app_assert(err == SL_STATUS_OK,
      "[E: 0x%04x] Failed to start
      advertising\n", (int)err);

      app_log("Started advertising\n");
      break;

    case sl_bt_evt_connection_opened_id:
      app_log("Connection opened\n");
      break;

    case sl_bt_evt_connection_closed_id:
      app_log("Connection closed\n");

      err = sl_bt_advertiser_start(adv_h,
             advertiser_general_discoverable,
             advertiser_connectable_scannable);
      app_assert(err == SL_STATUS_OK,
      "[E: 0x%04x] Failed to start
      advertising\n", (int)err);

      app_log("Started advertising\n");
      break;

    default:
      break;
  }
}
```

6 GATT Configurator

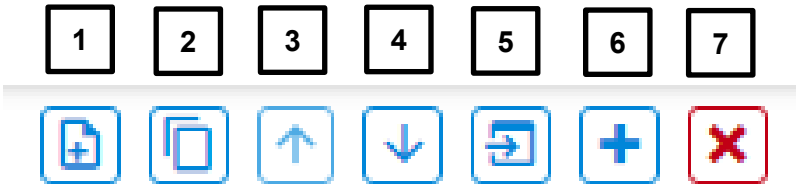
In Bluetooth SDK v3.0 the GATT Configurator is completely redesigned. The new user interface is more modern, while the generator tool makes it possible to add partial extensions to the GATT database.

New projects automatically open a GATT configurator tab. If the tab isn't open when modifying an existing project, go to the Project Configurator Configuration Tools tab. Click **Open** next to GATT Configurator. Alternatively you can double-click the gatt_configuration.btconf file in the Project Explorer view.

While the user interface design is completely new, the underlying functionality stays the same as in Bluetooth SDK v2.x: you can add predefined and custom services/characteristics to your GATT database and configure them.



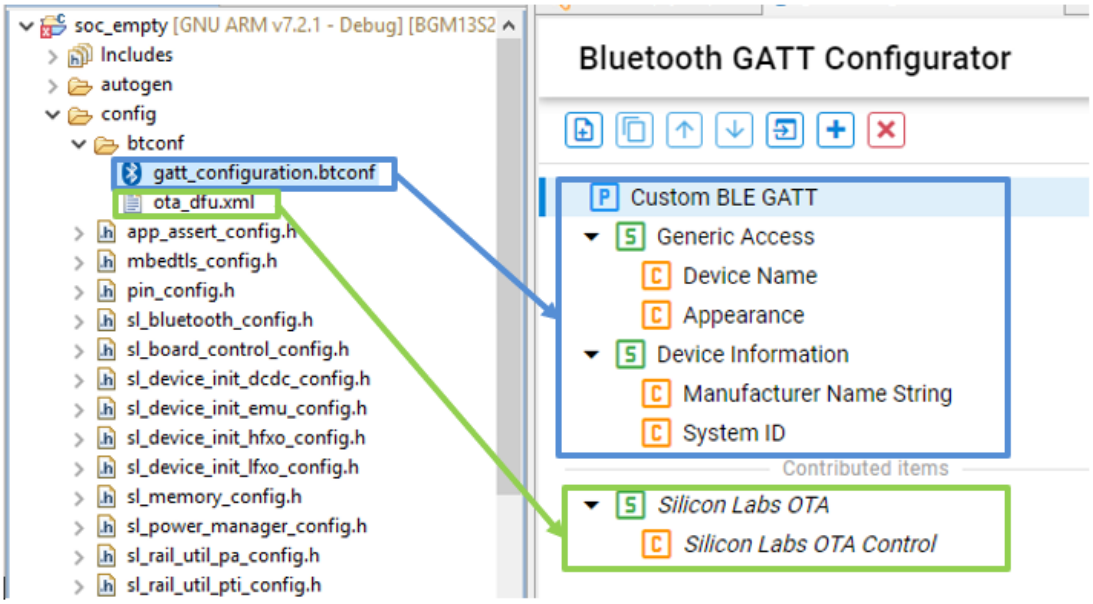
The GATT Configurator menu is:



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*.

A new GATT configurator feature is that it can accept partial database extensions from additional .xml files. This is important, because some software components may need to contribute to the GATT database with its custom service. For example, the OTA DFU component adds the OTA service to the GATT database by adding an ota_dfu.xml file, which defines the service, next to the .btconf file:



Another important innovation is that the GATT database-related files (gatt_db.c, gatt_db.h) are automatically regenerated as you edit and save the file (changes are not autosaved). There is no need to manually start the generator script as in Bluetooth SDK v2.x.

Migrating the GATT database is easy. You can import the database from your old project by clicking Import (6), and selecting the gatt.xml file of your old project. If your old database contains an OTA service, remove it from the database. It is defined by the new OTA DFU component.

7 Stack Configuration and Initialization

7.1 Configuration

The Bluetooth stack can be configured by passing a configuration structure to the stack init function. In Bluetooth SDK v2.x the configuration structure is defined in gecko_configuration.h, and the default configuration is usually defined in main.c, like this:

```
static gecko_configuration_t config = {
  .config_flags = 0,                                      /* Check flag options from UG136 */
#if defined(FEATURE_LFXO) || defined(PLFRCO_PRESENT)
  .sleep.flags = SLEEP_FLAGS_DEEP_SLEEP_ENABLE,           /* Sleep is enabled */
#else
  .sleep.flags = 0,
#endif
  .bluetooth.max_connections = MAX_CONNECTIONS,           /* Maximum number of simultaneous
                                                           * connections */
  .bluetooth.max_advertisers = MAX_ADVERTISERS,           /* Maximum number of advertisement sets   */
  .bluetooth.heap = bluetooth_stack_heap,                 /* Bluetooth stack memory for connection
                                                           * management */
  .bluetooth.heap_size = sizeof(bluetooth_stack_heap),/* Bluetooth stack memory for connection
                                                           * management */
#if defined(FEATURE_LFXO)
  .bluetooth.sleep_clock_accuracy = 100,                  /* Accuracy of the Low Frequency Crystal
                                                           * Oscillator in ppm. *
                                                           * Do not modify if you are using a module */
#elif defined(PLFRCO_PRESENT)
  .bluetooth.sleep_clock_accuracy = 500,                  /* In case of internal RCO the sleep clock
accuracy is 500 ppm */
#endif
  .gattdb = &bg_gattdb_data,                              /* Pointer to GATT database */
  .ota.flags = 0,                                         /* Check flag options from UG136 */
  .ota.device_name_len = 3,                              /* Length of the device name in OTA DFU mode */
  .ota.device_name_ptr = "OTA",                           /* Device name in OTA DFU mode */
  .pa.config_enable = 1,                                  /* Set this to be a valid PA config */
#if defined(FEATURE_PA_INPUT_FROM_VBAT)
  .pa.input = GECKO_RADIO_PA_INPUT_VBAT,                  /* Configure PA input to VBAT */
#else
  .pa.input = GECKO_RADIO_PA_INPUT_DCDC,                  /* Configure PA input to DCDC */
#endif // defined(FEATURE_PA_INPUT_FROM_VBAT)
  .rf.flags = GECKO_RF_CONFIG_ANTENNA,                    /* Enable antenna configuration. */
  .rf.antenna = GECKO_RF_ANTENNA,                         /* Select antenna path! */

};
```

In Bluetooth SDK v3.0 the configuration structure is defined in sl_bt_stack_config.h with slightly renamed type definitions:

Table 7-1. Changes in Bluetooth Configuration Types

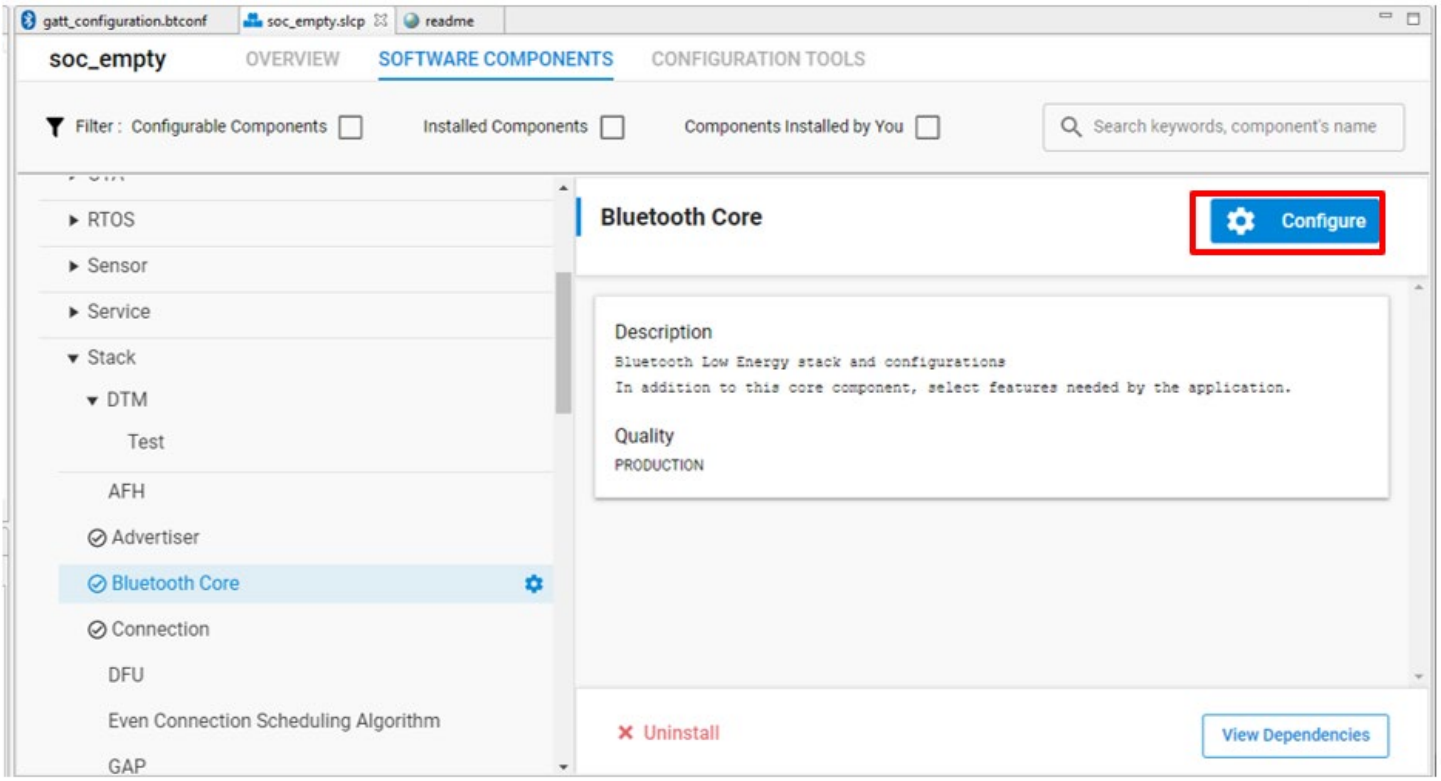
API 2.x	API 3.0	Notes
gecko_configuration_t	sl_bt_configuration_t	
gecko_bluetooth_config_t	sl_bt_stack_config_t	
gecko_*	sl_bt_*	

The configuration structure itself does not change much between v2.x and v3.0. Only the OTA config is removed from the structure. New runtime commands are provided for setting OTA flags and device name.

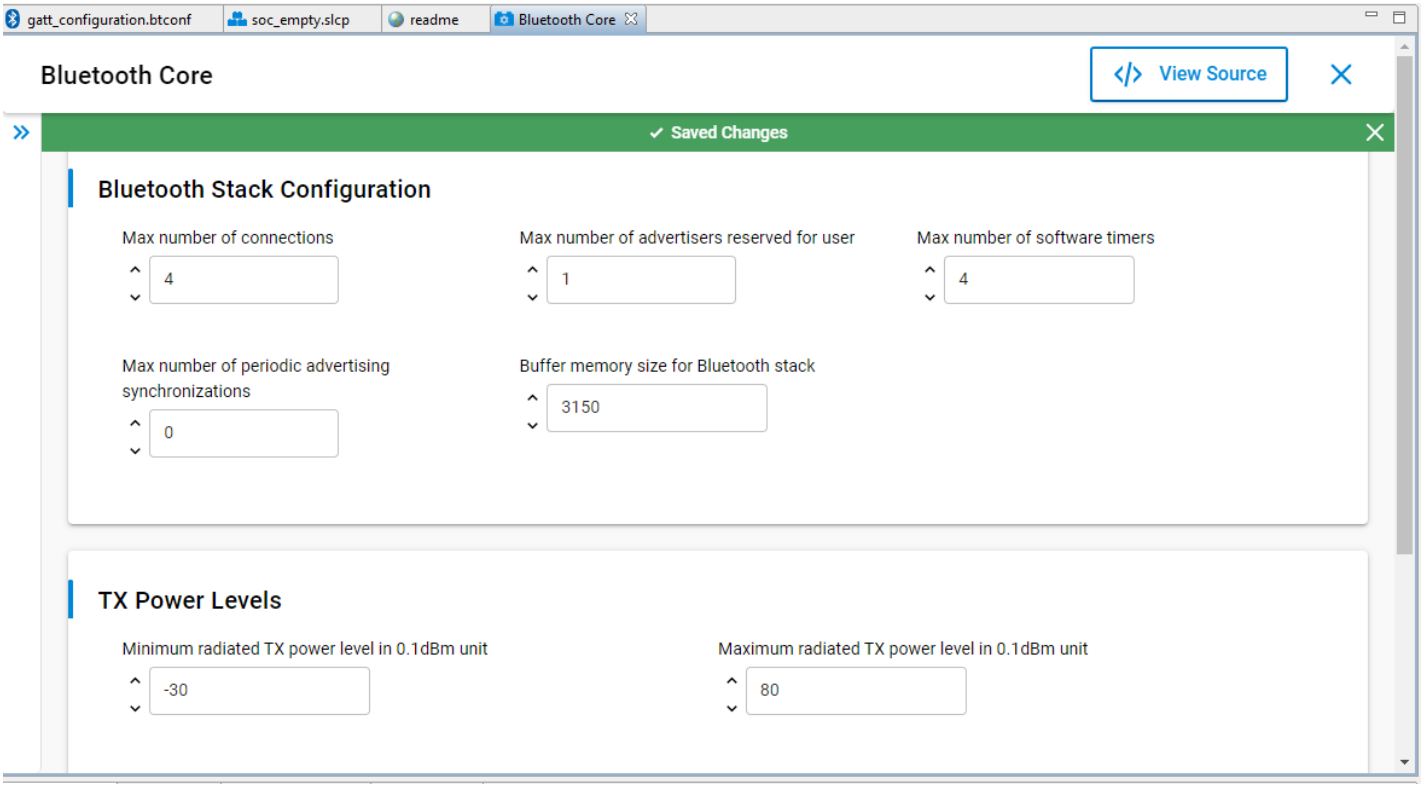
In Bluetooth SDK v3.0 the default configuration is defined in the sl_bluetooth_config.h header file instead of main.c. Moving the default configuration out of main.c separates config from functionality, and it also makes it possible to configure the stack using the Component Editor.

To edit the configuration parameters using a normal text editor, open sl_bluetooth_config.h.

To edit the configuration parameters using the Component Editor in SSv5, go to the **Software Components** tab, find the Bluetooth Core component, and click **Configure**:



Set the configuration values in the input fields. The values will be automatically verified and saved into the header file without any further action needed.



7.2 Initialization

To initialize the Bluetooth stack, many functions have to be called: one for initializing the stack in general with the configuration structure and one for each BGAPI class (such as gap, gatt, connection, and so on) to initialize the classes. Furthermore, additional stack features (such as AFH) have to be initialized with a feature init function. All of these functions are slightly renamed in Bluetooth SDK v3.0.

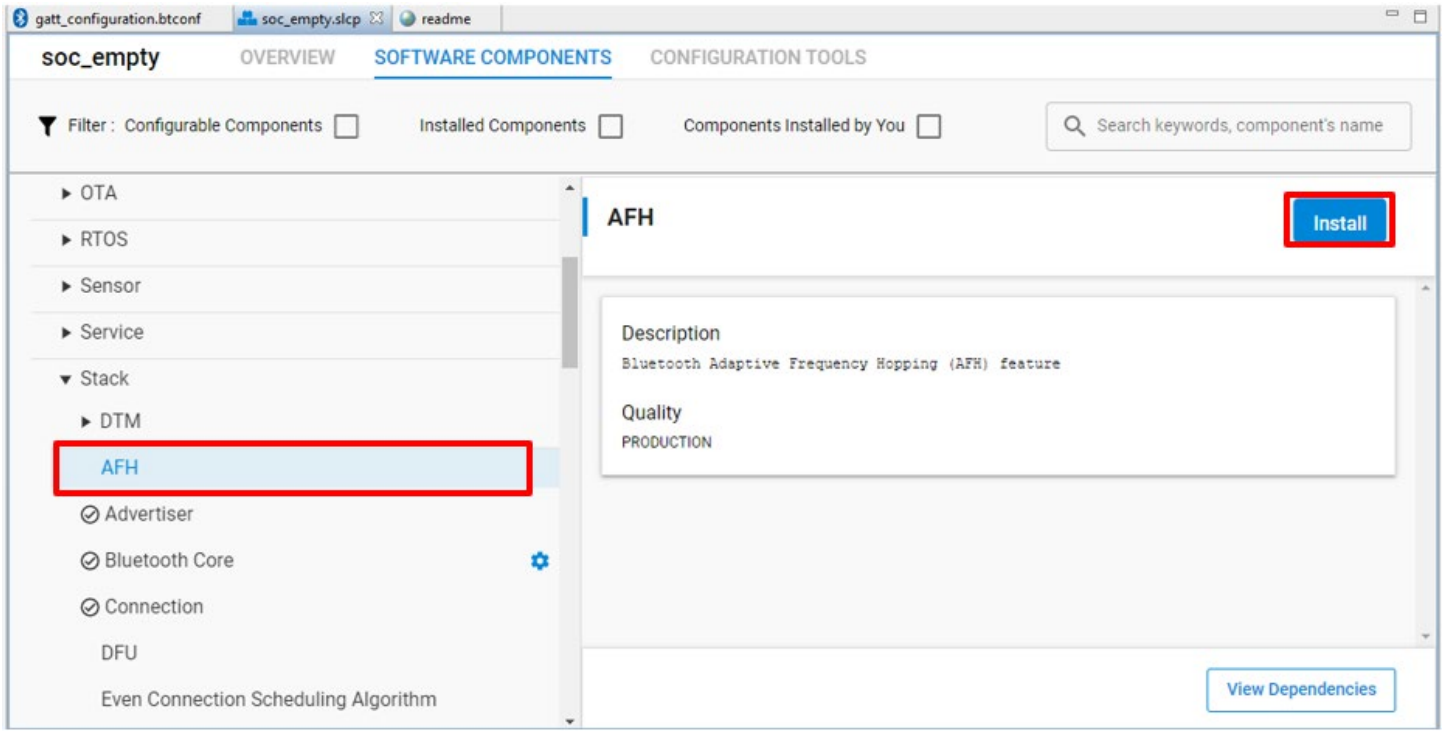
Table 7-2. Changes in Bluetooth init functions

API 2.x	API 3.0	Notes
gecko_stack_init(config)	sl_bt_init_stack(config)	
gecko_bgapi_class_*_init()	sl_bt_class_*_init()	For initializing BGAPI classes for NCP mode, add definition (-D) SL_BT_API_FULL in application project.
gecko_init_<feature>	sl_bt_init_<feature>	

In Bluetooth SDK v2.x, `gecko_init(*config)` can be used to initialize the stack. This function calls `gecko_stack_init(config)` and all the BGAPI class init functions that are usually needed in a general Bluetooth application. Unnecessary classes can be removed (to free up memory) and additional class init functions can be added based on the needs of the application. If a class is not initialized the BGAPI commands of that class cannot be called. Additional features always have to be initialized after the stack init if they are needed.

In Bluetooth SDK v3.0 each BGAPI class has a corresponding software component. This means that classes can be initialized by adding their respective software component. When adding the “Bluetooth” component to your project, most BGAPI classes are also added by default. To initialize additional classes, go to the Software Components tab and install the corresponding component. To remove unnecessary classes, uninstall the corresponding components. Filter on installed components to make the search easier. This may help make your application size smaller.

Additional features, such as Adaptive Frequency Hopping, Periodic Advertising, and so on can be initialized the same way as BGAPI classes, by installing the corresponding software component:



When migrating your project from v2.x to v3.0 it is recommended to remove all the initialization code from app.c and to add the proper software components to your project instead.

8 AppLoader

Beginning with Bluetooth SDK v3.0 the AppLoader uses a new feature of the bootloader: parsing GBL headers in RAM. This means that GBL headers do not have to be stored in flash during application update, which makes the process a bit faster, and more importantly it does not make the old application image corrupt if the GBL headers are incorrect.

However, this also means that the AppLoader in Bluetooth SDK v3.0 is incompatible with older bootloader versions. Therefore, if you migrate your project to Bluetooth SDK v3.0, and you use the AppLoader in the newly created project, rebuild the bootloader in v3.0 by selecting the bootloader example and building it, and update your device with the resulting compatible bootloader.

For information about building and using bootloaders, see UG266: Silicon Labs Gecko Bootloader User's Guide and online content at docs.silabs.com.

9 Migrating NCP projects

Migrating an NCP application is usually easy, since the stack and the application are well-separated. While the stack is running on the NCP target, the application is running on the NCP host. Therefore, a stack update usually does not affect the application except that the API changes must be respected.

An SDK update in the NCP use case means that:

1. The NCP target device must be programmed with the **NCP – Empty** sample app of the new SDK.
2. UART pins must be configured in the sample app.
3. The GATT database must be imported in the sample app.

Furthermore,

4. The NCP host device must include the new BGAPI header files, so that it can communicate with the target.
5. Deprecated API calls must be updated, if there are any.

Upgrading the NCP target code from Bluetooth SDK v2.x to v3.0 is easy. A new **NCP – Empty** project must be generated with Bluetooth SDK v3.0. The UART pins can be easily configured with the Pin Tool, and the GATT database can be easily imported with the GATT Configurator. Should you use deep sleep mode in the NCP target, you must install the Wake Lock component and configure it. For more information see AN1259: Using the Silicon Labs v3.x Bluetooth® Stack in Network Co-Processor Mode.

The NCP host update involves more changes. After updating the header files, not only the full Bluetooth API has to be updated but also some BGLIB commands and macros.

An NCP host code using Bluetooth SDK v2.x must contain the following header files:

- **bg_errorcodes.h**
- **bg_types.h**
- **host_gecko.h**
- **gecko_bglib.h**

and the following source file:

- **gecko_bglib.c**

An NCP host code using Bluetooth SDK v3.0 must contain the following header files:

- **sl_status.h** (in *SDK_DIR/platform/common/inc*)
- **sl_bt_types.h** (in *SDK_DIR/protocol/bluetooth/inc*)
- **sl_bt_api.h** (in *SDK_DIR/protocol/bluetooth/inc*)
- **sl_bt_ncp_host.h** (in *SDK_DIR/protocol/bluetooth/inc*)

and the following source files:

- **sl_bt_ncp_host_api.c** (in *SDK_DIR/protocol/bluetooth/src*)
- **sl_bt_ncp_host.c** (in *SDK_DIR/protocol/bluetooth/src*)

The new header files use the new nomenclature (commands/events starting with sl_bt_...) even if the underlying BGAPI packet content, which is sent to the target device via UART, may be unchanged in some cases. Therefore NCP host code must be completely updated according the description in section 5 Bluetooth API, using the new BGAPI.

Beside the changes in BGAPI (Bluetooth commands and events), the host API is also changed similarly to the changes in C API detailed in section 5.7 Changes in the C API. The following table summarizes the changes in the host API:

Table 9-1. Changes in the Host API

API 2.x	API 3.0	Notes
BGLIB_DEFINE	SL_BT_API_DEFINE	
BGLIB_INITIALIZE	SL_BT_API_INITIALIZE	
BGLIB_INITIALIZE_NONBLOCK	SL_BT_API_INITIALIZE_NONBLOCK	
struct gecko_cmd_packet	sl_bt_msg_t	
BGLIB_MSG_ID	SL_BT_MSG_ID	

API 2.x	API 3.0	Notes
struct gecko_cmd_packet* **gecko_wait_event**()	sl_status_t **sl_bt_wait_event**(sl_bt_msg_t* evt)	In API 3.0, an event object is copied into the memory provided by application.
struct gecko_cmd_packet* **gecko_peek_event**()	sl_status_t **sl_bt_pop_event**(sl_bt_msg_t* evt)	
int **gecko_event_pending**()	bool **sl_bt_event_pending**()	

The NCP host code must be updated according to these changes. For example fetching an event changes from:

```
struct gecko_cmd_packet *p;
p = gecko_wait_event();
switch (BGLIB_MSG_ID(p->header)) {…}
```

to:

```
sl_bt_msg_t evt;
sl_bt_msg_t *p = &evt;
sl_bt_wait_event(&evt);
switch (SL_BT_MSG_ID(p->header)) {…}
```

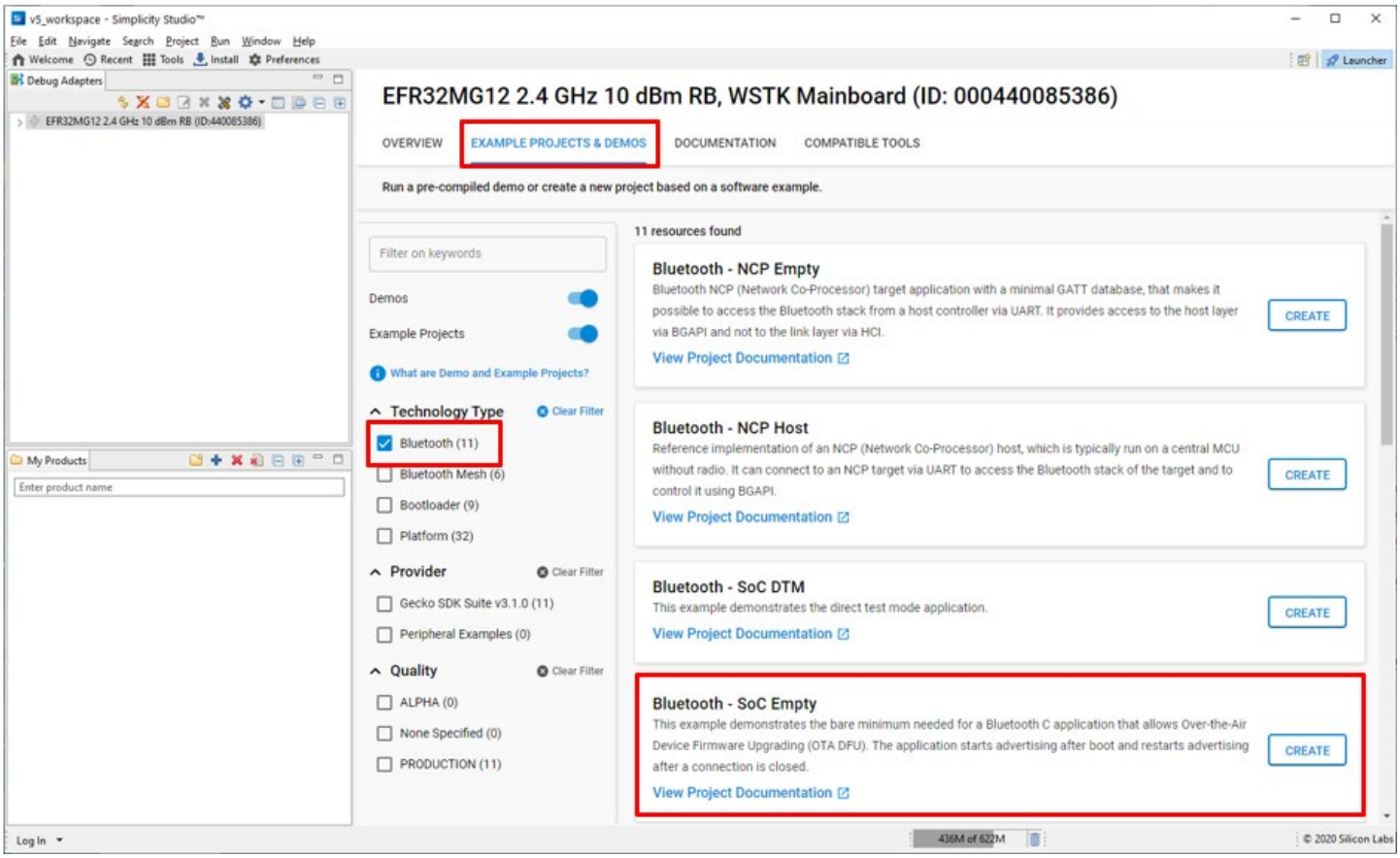
Regarding the software architecture, the **empty host example** created for PCs (in SDK_DIR/app/bluetooth/example_host/empty) is updated to align with the new SoC software architecture (see section 4 Software Architecture). While it is not necessary to update the architecture on the NCP host, it is recommended to use this new architecture on newly created NCP host projects, so that it aligns with SoC code.

The **NCP – host** example created for EFR devices (find it among the example projects in SSv5) is also updated to the new software architecture. If the host runs on a Silicon Labs device, it is highly recommended to start a new **NCP – host** project with the new software architecture, and migrate your code into it, just as for an SoC project.

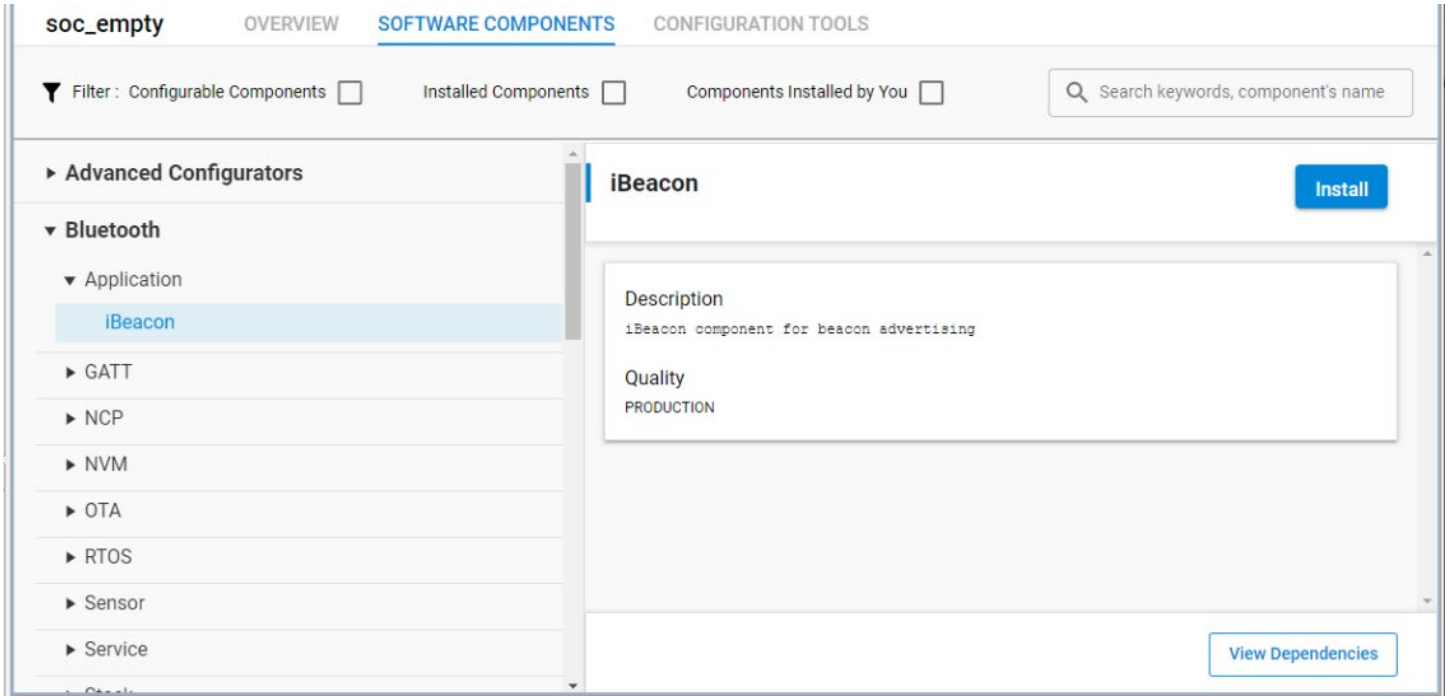
10 Migration Summary

This section summarizes the steps required to migrate a project from Bluetooth SDK v2.x with Simplicity Studio 4 to Bluetooth SDK v3.0 with Simplicity Studio 5.

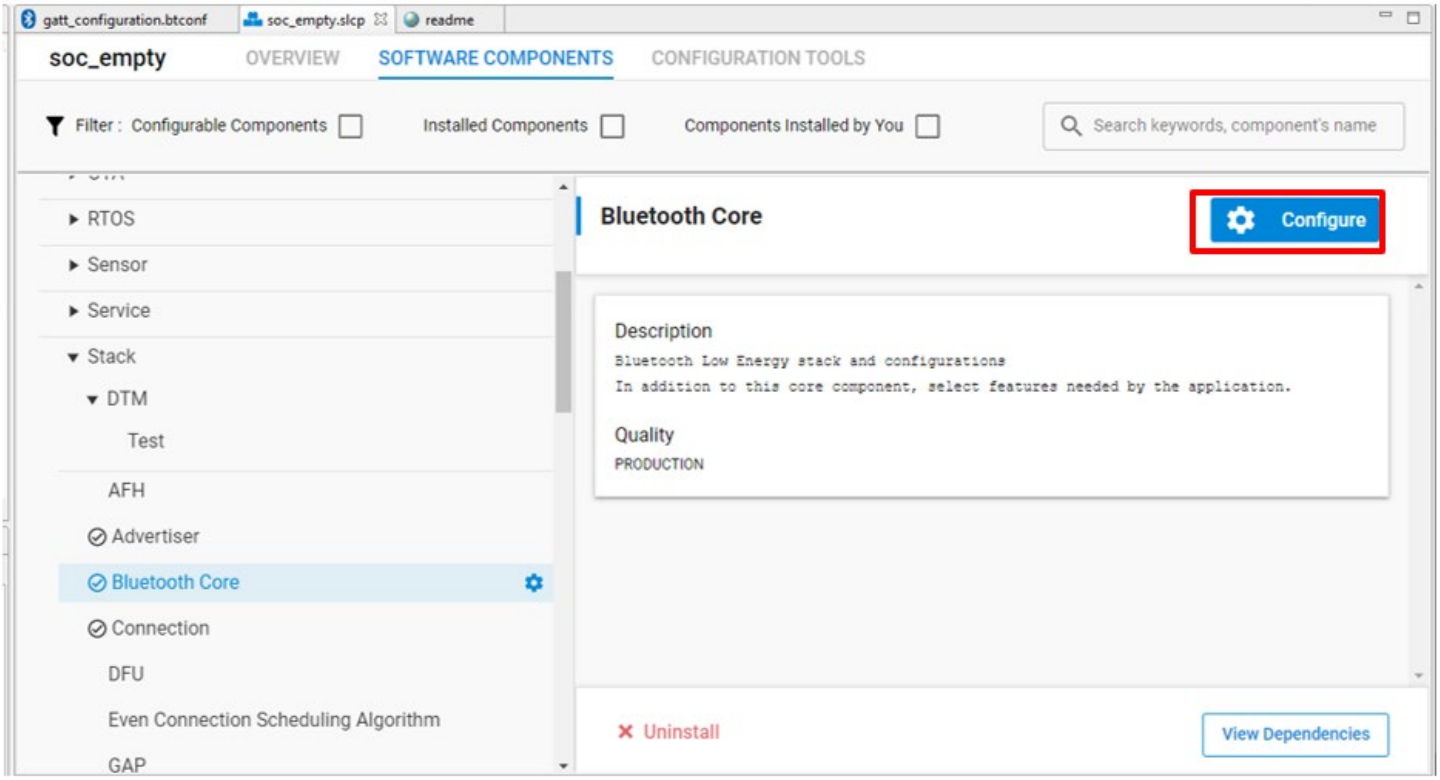
1. Create a new SoC-Empty project in the new SDK from the SSv5 Launcher view (section 2 Project Structure).



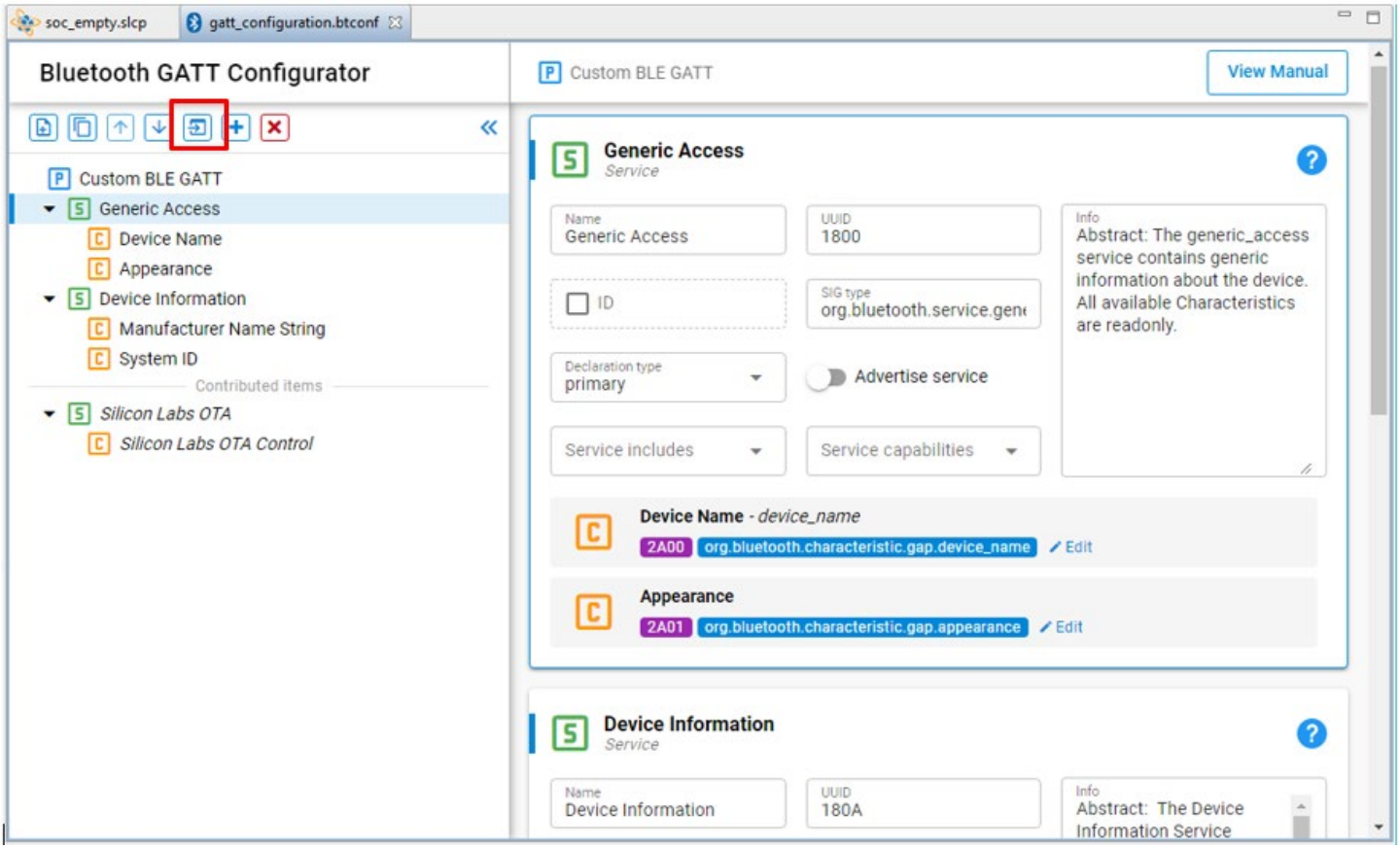
2. Familiarize yourself with the functionality available through the Component Library. Then install components that correspond to functions in your original project (section 3 Software Components).



3. If your application was RTOS-based, add the Micrium Kernel component (section 4.2 RTOS).
4. Remove all the initialization code from app.c and instead add the corresponding software components to your project (section 7.2 Initialization).
5. Edit any configuration parameters, either in sl_bluetooth_config.h using a text editor, or in the Component Editor in SSv5. Changes there are automatically saved (section 7.1 Configuration).



6. Copy the Bluetooth event handlers from `appMain()` into `sl_bt_on_event()` defined in app.c. If your application is RTOS-based, the parallel application task must be started in `app_init()`, which is also defined in app.c (section 4.1 Bluetooth Event Handling).
7. Review the Bluetooth API changes detailed in section 5 Bluetooth API. Edit your application code accordingly. Most function names are just renamed from `gecko_cmd_` to `sl_bt_` While there are exceptions, the compiler will give a verbose error message giving the name of the replacement function when compiling with the old API.
8. If your new project uses AppLoader, rebuild the bootloader in GSDK v3.0, and update your device with the compatible bootloader (section 8 AppLoader).
9. Import the GATT database from your old project by opening the GATT Configurator, clicking Import, and selecting the gatt.xml file of your old project. If your old database contains an OTA service, remove it from the database. It is defined by the new OTA DFU component.



10. If the application uses a Host/NCP configuration, both the target and the host must be updated.

    Target:

    1. The target device must be programmed with the **NCP – Empty** sample app of the new SDK.
    2. UART pins must be configured in the sample app.
    3. The GATT database must be imported in the sample app.

    Host:

    1. The host device must include the new BGAPI header files, so that it can communicate with the target.
    2. Deprecated API calls must be updated, if there are any.

    See section 9 Migrating NCP projects for more information.

Simplicity Studio

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

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

http://www.silabs.com