AN1254: Transitioning from the v2.x to the v3.x Proprietary Flex SDK

Flex Software Development Kit (SDK) v3.0 contains a number of changes compared to Flex 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 (SSv5). Projects are now built on a component architecture. SSv5 includes project configuration tools that provide an enhanced level of software component discoverability, configurability, and dependency management. These include a Component Configurator, and a redesigned Radio configurator.

KEY POINTS

- Reviews changes common to both RAIL and Connect.
- Describes RAIL-specific changes and how to migrate a RAIL application.
- Discusses Connect-specific changes and how to migrate a Connect application.

While these changes are a result of overall improvements in the SDK and in SSv5 it also means that developer flow changed between Flex SDK v2.x and v3.0. This document highlights the main changes, so someone with experience on Flex SDK v2.x can quickly adjust to the new developer flow.

The document also provides the recommended way to migrate existing projects to Flex SDK v3.x. However, since application development on Flex SDK v2.x was not standardized, especially for RAIL, project migration might be very different from case to case.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

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 plugins from the Eclipse Marketplace.

Gecko Software Development Kit suite version 3.0.0 (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 3.0, Proprietary 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 are 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 3.x development environment include:

- Project source management options (link to SDK sources or copy all contents to user folder)
- Graphical pin configuration through the Pin Tool
- 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

2   Changes in Flex SDK

This section highlights the SDK-level changes that apply to both RAIL and Connect development.
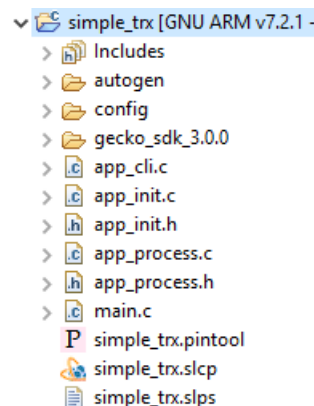
2.1   Project Generation

After creating a project, you no longer have to generate it. Generation happens automatically. None of the new Simplicity IDE project configuration tools (such as Project Configurator or Component Editor) have **Generate** controls: Saving the file triggers generation.

2.2   Project Structure

Projects always include the following parts:

- **autogen** folder: Only the autogen folder includes generated code. It includes the PHY configuration (*rail_config.c*), init code, the linker script, and other generated code used by components, like the command descriptors for the CLI interface.
- **config** folder: Component configuration headers are located in this folder. These can be edited with the Simplicity IDE Component Editor, but directly editing the header file is also possible. The Component Editor is available through the Project Configurator's **Configure** control, available only for configurable components.
- **gecko_sdk** folder (with version number): Contains source and binary files added by components.
- **files in the root folder**: Only the application specific files should be in the root folder, including source files, the project configurator (.*slcp*) file and the Pin Tool (.*pintool*) file.

```
∨ 📂 simple_trx [GNU ARM v7.2.1 -
   > 🗐 Includes
   > 📂 autogen
   > 📂 config
   > 📂 gecko_sdk_3.0.0
   > .c app_cli.c
   > .c app_init.c
   > .h app_init.h
   > .c app_process.c
   > .h app_process.h
   > .c main.c
     P simple_trx.pintool
     🔩 simple_trx.slcp
     📄 simple_trx.slps
```

Note that all files related to the project should be visible in the Project Explorer view, including header and library files. In Flex SDK 2.x, library files were never shown in the project explorer, and only some headers were shown.

All projects include *main.c*, which is not recommended to modify. Instead, add the initialization code to *app_init.c*, and implement the main loop in *app_process.c*. This way, the System components can initialize components, and call the "process" function of the components that require it. Additionally, enabling an RTOS will convert *app_process*'s main loop into an RTOS task.

These files are also available in Connect projects, but `emberAfInitCallback()` and `emberAfTickCallback()` are not directly called from the main framework, they are called by the stack. Doing so empowers the Connect stack to prioritize operations critical to the timely execution of stack and radio tasks.

2.3   Working with Custom Boards

Working with custom boards was somewhat complicated in Flex SDK v2, as most examples depended on Wireless Starter Kit (WSTK) peripherals and the known wiring of the WSTK. With Flex SDK v3.x, working with custom boards is very simple. Two methods are available, depending on your needs:

- Create an example on the WSTK, then remove the WSTK on the Project Configurator Overview tab using the **Change Target/SDK** control. This way, the wiring of the kit is kept, but you can change it using the Pin Tool.
- Create an example with just the part, no WSTK, and set up the wiring for all required peripherals of the example (like buttons or pins) using the Pin Tool.

Access the Pin Tool through the **Advanced Configurators > Pin Tool** component or by opening the <project>.pintool file.

Note that a WSTK can also depend on components (typically peripheral init components) that are removed if you remove the WSTK from the project.

2.4   Radio Configurator

The Radio Configurator follows a similar workflow to the old version, but with some significant changes.

To open the Radio Configurator, either open the **Advanced Configurators > Radio Configurator** component or open the file *config/rail/radio_settings.radioconf*.

In the previous configurator, setting up a custom PHY required selecting the PHY named Custom settings. In the new version, the Radio Configurator opens parameter tiles after the **Customized** switch is enabled. Changes from the originally selected PHY are highlighted next to the parameter, as shown in the following figure.
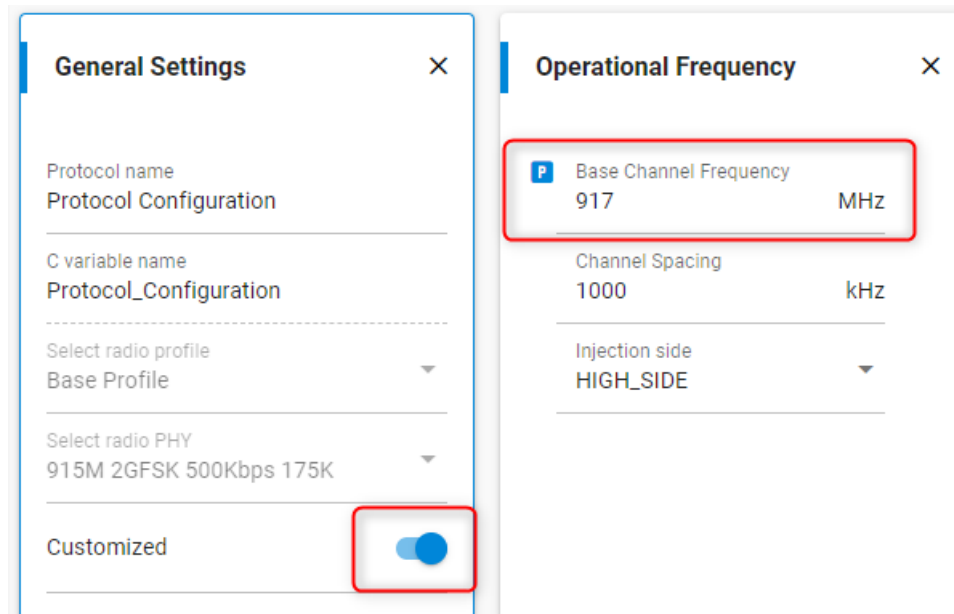
The Radio Configurator also has an updated look and feel, better embedded documentation, and a search tool, among other enhancements. For details, see *AN1253: EFR32 Radio Configurator Guide for RAIL in Simplicity Studio 5*.

Radio configs created with Simplicity Studio 4 (*.isc* files) can be imported into the new Radio Configurator through the **Import** control:
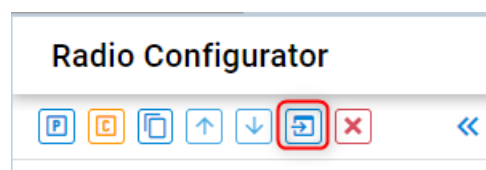
Note that the Radio Configurator is now installed as part of the SDK, not Studio. This means that projects created with Flex SDK 3.0.x and projects created with Flex SDK 3.1.x might use different versions of the Radio Configurator.

3 RAIL

While the tools to create projects are very different, the RAIL library itself and its API are backward-compatible.

3.1 Sample Applications

RAIL sample applications are updated to fit well into the component architecture and to use RAIL features that were introduced after the initial release of these applications.

3.2 RAIL Utility, Initialization Component

RAIL initialization and event handling was part of the application in Flex SDK v2.x. Now that functionality is moved to the **Platform > Radio > RAIL Utility, Initialization** component. This component is now responsible for the following:

- RAIL initialization
- Loading the PHY config
- Setting up auto state transitions
- Event configuration
- Event distribution

Of course, the application can still change event configuration in run-time, and event handling is still the responsibility of the application.

To make it possible for components to handle events, an event distribution system is required. The application should use the `sl_rail_app_on_event()` callback for event handling when this component is enabled.

This component is the recommended method to start RAIL and handle events, but not mandatory. When porting existing code to Flex SDK v3 it is simpler to avoid it.

For more details, see the component's documentation.

3.3 Other RAIL Utility Components

These components are available in the **Platform > Radio** group.

- **Radio Utility, Front End Module**: The FEM (front-end module, such as an external amplifier) control used to be implemented in some stacks. Now the same component can be used with any stack.
- **RAIL Utility, Co-existence:** Co-existence (typically with WiFi) was supported by some stacks, and now is also supported by RAIL through this component.
- **RAIL Utility, Packet Trace:** Packet trace interface (PTI) initialization was part of the **RAIL HAL** plugin, and it now in this component.
- **RAIL Utility, Power Amplifier:** PA initialization (formerly part of the application) and the **PA conversations** plugin functionality were combined into this component
- **RAIL Utility, RF Path:** RF Path configuration, formerly part of the application, was moved to this component. Note that currently only the EFR32 xG21 family supports this feature.

For more details, see the documentation for these components on docs.silabs.com.

While none of these components are mandatory, it is fairly simple to modify existing code to use them.

3.4 Power Manager Support

Gecko SDK v3.0 introduces **Services > Power Manager**, a component that can be used to enter the lowest energy mode needed in a given application. Software modules can request higher energy modes or give up their requirements. For more details, see the Power Manager documentation.

RAIL can directly request the energy mode required for the current radio operation, which can result in slightly more energy efficient operation than setting the energy mode from the application.

This mode is not enabled in RAIL at initialization. If the Power Manager component is present in the project, it can be enabled by `RAIL_InitPowerManager()` or by enabling the **Flex > RAIL > Utility > Flex - RAIL Power Manager Sleep** component.

3.5 Migrating Existing RAIL Projects to Flex SDK v3.x

The recommended way to migrate an existing project to the new SDK is the following:

1. Start with a **Flex(RAIL) - Empty Example** and uninstall the RAIL-related components from it (except the library), as this is most likely part of your application already. PTI and PA components might be still needed, if you used the **RAIL HAL** or the **PA conversations** plugins. Note you can use the Installed Components filter at the top of the Software Components tab to easily find the installed RAIL components.

2. Import the old *.isc* file into the Radio Configurator.

3. Copy the application code into the project.

4. Test at this point, as you might need to update some peripheral init code.

5. Decide what RAIL-related components you want to use. Install one, and remove the application code that used to perform that function. Repeat this step for all components.

4   Connect

Connect also remained mostly the same at the API level, but replaces the features of Application Builder (the generator GUI in Simplicity Studio 4) with either a new tool or a new API. However, Connect is not 100% backwards compatible even at the API level (for example, channels are now stored on 2 bytes, not 1), although updating should be fairly simple with the API documentation.
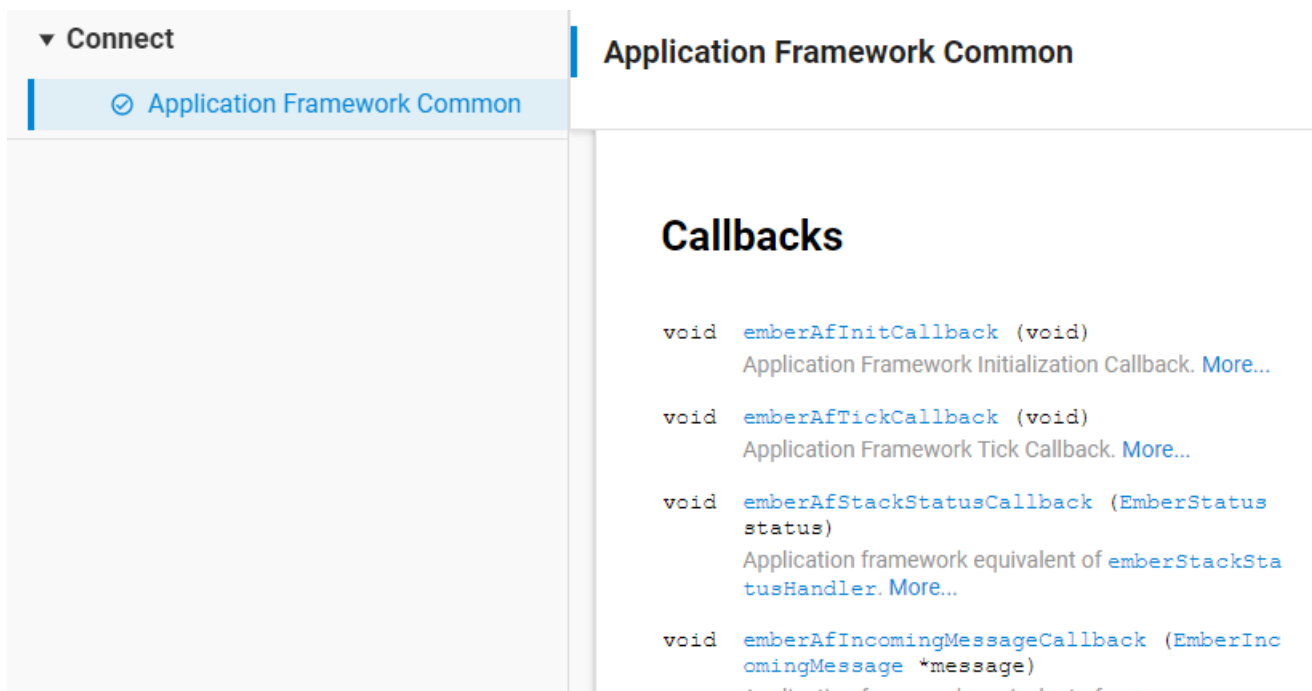
4.1   Components

Connect used to be configured as a set of plugins, now it is a set of components. The change for Connect components is straightforward. Basically every plugin has an equivalent component.

4.2   Callbacks

The Callbacks interface is replaced by the embedded API documentation for each component. The recommended method to add a callback to the code is to copy from the documentation and paste into the relevant C file (usually *app_process.c*).

Some components have their own callbacks (like the OTA components). The stack callbacks are all routed through the application framework, so they can be found under the **Application Framework Common** component:

4.3   Events

Events in Connect can be used to create scheduled and/or periodically-called functions. In Flex SDK v2.x, events were created by Application Builder, on the **Other** tab. This was replaced by the `emberAfAllocateEvent()` function, which is part of the **Application Framework Common** component. With this method, you do not have to regenerate the whole project just to add a new event, and events can be created at run-time, not just generation-time.

4.4   HAL Libraries

Connect used to be based on Ember HAL, the hardware abstraction layer library originally developed for the EmberZNet stack. With Gecko SDK v3.0, the Connect SDK has transitioned to a more uniform HAL and platform software. The component **Services > Legacy HAL** provides a translation layer from the old API to the current drivers. Currently, the Legacy HAL component is still required by the Connect stack itself, but in the long term it will be an optional component to support applications that depend on the legacy HAL API.

4.5   Sleep Management

Connect formerly used Ember HAL's sleep management functions through the Idle/sleep plugin, which was optional. In Flex SDK v3.x, Connect uses *Power Manager* for these functions, and it is mandatory. Connect interfaces to it through the *Application Framework Common* component, `gecko_sdk_3.0.0/protocol/flex/app_framework_common/app_framework_sleep.c`, which is very similar to the old Idle/sleep plugin. Both the legacy `emberAfCommonOkToEnterLowPowerCallback()` and Power Manager's API can be used to keep the MCU awake.

4.6   Command Line Interface

Although a Command Line interface is not part of the Connect stack directly, many Silicon Labs and customer applications relied on the command line interface available in Connect. This was replaced with the new, platform-level CLI, which has an incompatible API, and at this time has no user interface. Commands can, however, be added to the CLI system with APIs, shown as an example in the CLI component documentation.

4.7   Debug Print

The old print APIs were replaced. To print from the application, call `connect_stack_debug_print()`. Note that this is standard printf now, while the printf implemented in Connect v2.7 used slightly different formatting strings (for example, `%2x` in the old version is equivalent to `%04x` in the new).

4.8   Host-NCP Mode

Host-NCP mode is not supported in Connect v3.0. The functionality will be added in a later release. Communication between Host-NCP code developed on Connect v2.7 and SoC code developed on Connect v3.x should be possible.

4.9   Migrating Existing Connect Projects to Flex SDK v3.x

The recommended way to migrate an existing project to the new SDK is the following:
1.   Start with a **Flex(Connect) - SoC Empty** project.
2.   Import the old *.isc* file into the Radio Configurator.
3.   Enable the component equivalents of the plugins used in the original project.
4.   Copy the application code into the project.
5.   Register the events at init with the **emberAfAllocateEvent()** API.
6.   Update where needed for the Connect 3.0 API.
7.   Update print and CLI to the new API.

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