# UG235.04: Customizing Applications with Silicon Labs Connect v2.x

This chapter of the *Connect v2.x User's Guide* describes how to use Connect plugins, callbacks, and events to provide developer-configurable features and application behavior. The Connect stack is delivered as part of the Silicon Labs Proprietary Flex SDK. The *Connect v2.x User's Guide* assumes that you have already installed the Simplicity Studio development environment and the Flex SDK, and that you are familiar with the basics of configuring, compiling, and flashing Connect-based applications. Refer to *UG235.01: Developing Code with Silicon Labs Connect v2.x* for an overview of the chapters in the *Connect v2.x User's Guide*.

**KEY POINTS**

- Introduces the Connect Application Framework.
- Describes the plugins available in Connect.
- Describes how callbacks are used to customize application behavior.
- Describes timing application behavior using Events.

The *Connect v2.x User's Guide* is a series of documents that provides in-depth information for developers who are using the Silicon Labs Connect Stack for their application development. If you are new to Connect and the Proprietary Flex SDK, see *QSG138: Proprietary Flex SDK v2.x Quick Start Guide*.

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. Connect Application Framework

The Silicon Labs Application Builder (AppBuilder) allows a developer to start a new project based on an existing framework of best-practice application state machine code developed and tested by Silicon Labs. This framework sits on top of the Silicon Labs Connect stack to interface with the Hardware Abstraction Layer (HAL) and provides application layer functionality. To support this modular configurability, the Connect stack code has been structured to support optional functionality blocks called **plugins**. Each plugin is provided as a standalone library or set of source code. The corresponding stub library is also provided and is compiled in place of the full library if the plugin has not been selected for inclusion in the project within AppBuilder. This allows a developer to generally include only those capabilities essential to the target application, without paying a footprint/resource penalty for an unused feature set.

Underlying code checks at run time whether libraries are present or not. Common stack code performs some actions conditionally if a library is present or not. Plugins may depend on other plugins. Plugin dependencies are managed within AppBuilder. Each plugin implements one or more callbacks on top of the Application Framework. These can be implemented in the `flex-callbacks.c` file after AppBuilder generates the project. Callbacks are the places where developers can add custom application code on top of the Silicon Labs existing framework to give that application unique behaviors and determine how it will react.

Within the callback implementations, developers can utilize the entire HAL and stack APIs as well as a complete set of Application Framework-specific APIs that often provide high-level wrappers around complex HAL or stack functionality. These APIs are documented in an Application Framework API Guide (under “SDK Documentation” in the Launcher perspective in Simplicity Studio, also online at https://docs.silabs.com/connect-stack/latest), and examples of their usage can also be found in the Connect sample code.

# 2. Plugins

This chapter describes some of the plugins available, grouped by the stack layer in which they reside. For a complete list, see the Plugins tab of a current Connect stack release. Click on a plugin in AppBuilder to review additional information (description text, common source files, defined callbacks, etc.) about that plugin (see the following figure for an example).

**Note:** The Plugins tab also includes some options specific to RAIL (Radio Abstraction Interface Layer) that are not compatible with Connect applications. These should not be enabled (the RAIL Library API plugin is an exception to this rule).

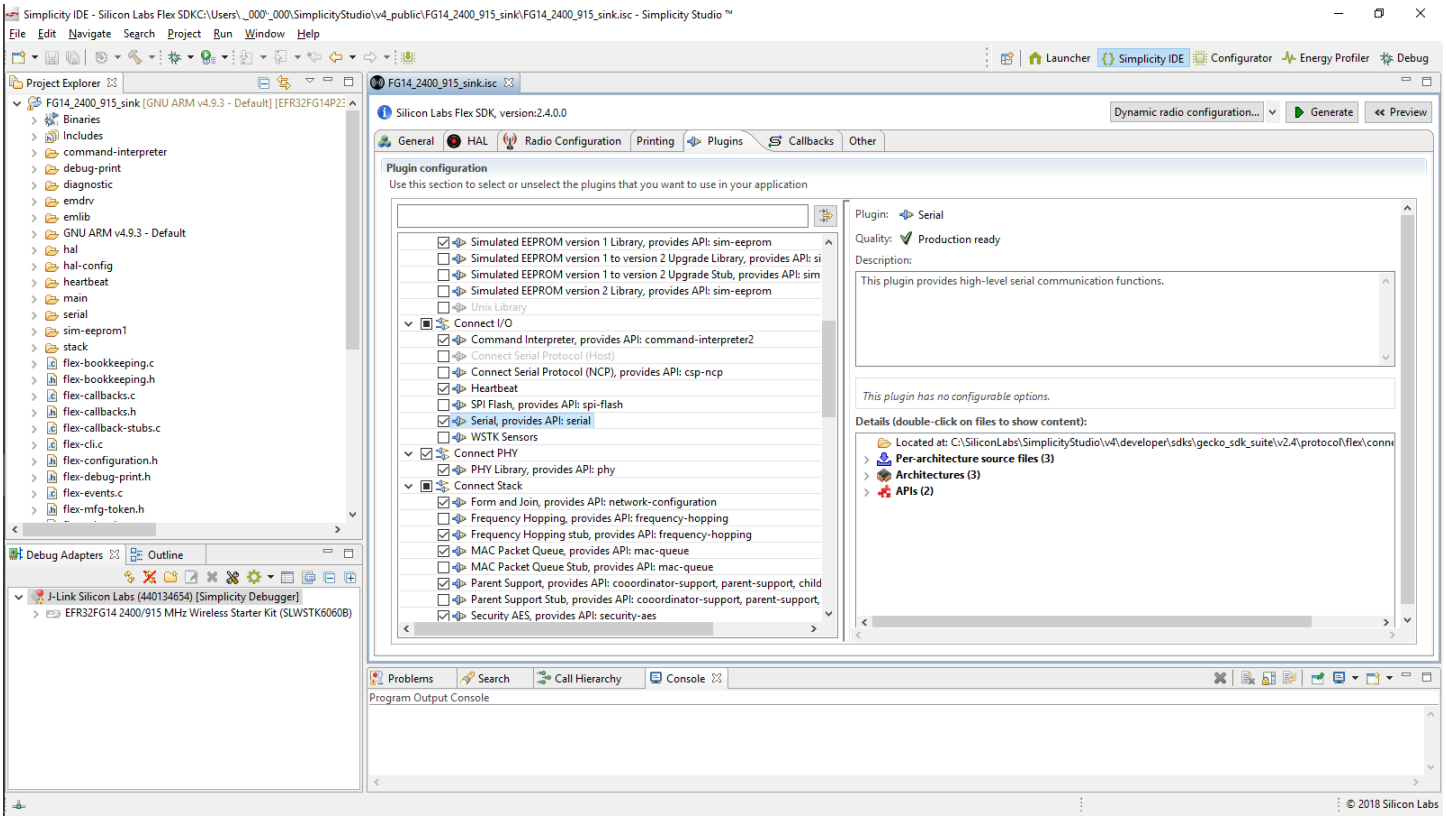



**Figure 2.1. The Plugins Tab of a Connect Example**

### 2.1 MAC and Network Layer Plugins

Media Access Control (MAC) and Network layer optional plugins found in the Connect Stack group on the Plugins tab include:

- Frequency Hopping
- MAC Packet Queue
- Parent Support
- Security plugins
- AES
- XXTEA
- Stack Common

**Frequency Hopping:** Allows nodes to communicate while rapidly switching channels in a pseudo-random fashion, thereby reducing channel interference which aids in regulatory compliance. For more information, see *UG235.03: Architecture of the Silicon Labs Connect Stack*.

**MAC Packet Queue:** Some applications need to submit multiple messages to the Connect stack. This plugin provides dynamic memory allocation functionality and permits messages to be queued and sent out as soon as possible according to the submission order and/or priority. By disabling this plugin, the Connect stack can only handle one message at a time.

**Parent Support** (requires MAC packet queue): Provides parent functionalities such as indirect communication (communication with sleepy devices), child table and routing table, and should be included for any coordinator or range extender node intended to support multiple end-device and/or sleepy end-device nodes. Star end devices, direct devices, or MAC devices should select the Parent Support Stub plugin instead to save flash/RAM. The Parent Support plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect*.

**Child table:** Allows a star coordinator or a star range extender to support multiple child devices. Child devices are aged and eventually removed. The child information table is stored in non-volatile memory (NVM) and requires (n*11) Bytes token space, where n is the configured child table size. (For more information, see the Simulated EEPROM discussion in section 2.2 Application Framework Plugins.) The child table size of a coordinator is limited to 64, while the range extender child table sizes are limited to 32.

**Indirect queue:** Buffers packets destined to sleepy child devices.

**Routing table:** Is needed for star coordinator applications if the network includes star range extenders. It stores the information collected from star range extenders.

**Security, AES:** Enables nodes to exchange secured messages with IEEE 802.15.4 mode-5-like or mode-5 MAC encryption/authentication scheme. This takes advantage of the AES hardware acceleration block available on EFR32 devices. Use this plugin only if the application will use/supports AES security. Otherwise, select the **Security AES Stub** plugin instead. (Uses the CRYPTO0 peripheral directly, or through mbedTLS if that plugin is enabled.)

**Security, XXTEA:** Provides legacy XXTEA-based security. Should be included only if the application will use/supports XXTEA security. Otherwise, select the Security XXTEA Stub plugin instead. Silicon Labs does not recommend this for new designs because it is less secure then AES.

**Stack Common:** Provides the Connect stack basic common functionality. Should be included in any Connect application.

### 2.2 Application Framework Plugins

The Connect Application Framework plugins are used to manage application layer functionality as follows.

**Connect Common group**
- Main
- Main (NCP)

**Connect Debug group**
- Diagnostic
- Debug Print
- Stack Packet Counters

**Connect Utility group**
- BLE
- Idle / Sleep
- Mailbox
- OTA-related plugins
- Poll
- mbedTLS

**Connect I/O group**
- Serial
- Command Interpreter
- Heartbeat
- WSTK (Wireless Starter Kit) sensors

**Connect HAL group**
- Bootloader interface
- HAL Library
- Micrium RTOS
- Simulated EEPROM version *
- NVM3

**Connect Common – Main:** Defines the `main()` function for System on Chip (SoC) applications. It calls all the required initialization functions. It also implements the stack handlers and dispatches them to every plugin that subscribes to them by calling into the bookkeeping auto-generated callbacks. If the diagnostic plugin was selected, prints out reset information upon reset.

**Connect Common – Main (NCP):** Defines the `main()` function for Network Coprocessor (NCP) applications. It calls all the required initialization functions.

**Connect Debug – Diagnostic:** Provides program counter diagnostic functions and, if the software crashes and the system resets, prints out on the serial port reset information including the call stack.

**Connect Debug – Debug print:** Manages each plugin's `printf()` debug APIs. These APIs are auto-generated by the Application Framework (as determined by the user-defined Debug Configuration section of the **Printing** tab in AppBuilder). The application can also define its own debug printf types. This makes it possible to easily configure which plugins have debug print routines included or excluded and turned on or off at runtime. If this plugin is not selected, the API calls have no effect.

**Connect Debug – Stack packet counters:** Provides stack packet counters functionality. If this plugin is enabled, the Connect stack keeps track of successful and failed transmissions as well as successful received packets and dropped incoming packets.

**Connect Utility – Idle/sleep:** Manages idle and sleep mode.

**Idle mode:** The main application loop is halted while the radio stays on. An incoming packet causes the node to get out of idle mode. Other interrupts are also served.

**Sleep mode:** The MCU processing is halted and the radio is disabled.

The plugin includes the logic for determining if the device can idle or sleep and initiates idle/sleep whenever it is possible. It attempts to sleep first, but if that is not possible, then it attempts to idle. It queries the Connect stack, Application Framework plugins, and the application. This plugin can only be enabled for a bare-metal (without Micrium OS) SoC application. The idle/sleep plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect*.

**Connect Utility – Mailbox:** A server/client service that allows client nodes to submit messages to, and retrieve messages from, a server node. An application-layer protocol (as opposed to the indirect queue, which is provided by the MAC layer), the Mailbox plugin supports much longer timeouts (default 1 hour, can be set to days) than the indirect-queue (between 8 and 30 seconds). The Mailbox plugin is not supported in MAC mode. The Mailbox plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect*.

**Connect Utility – Poll:** Manages periodic polling for end devices. Regular end devices need to exchange some sort of traffic with the parent as a **keep-alive** mechanism, also referred to as a **long poll interval**. Star sleepy end devices need to poll the parent for incoming packets, also referred to as a **short poll interval**. Star end devices are in long poll mode by default. The application can switch to short poll mode when appropriate using the plugin. For instance, a star sleepy end device sends out a packet that expects a response. The application then switches to short poll mode until the expected response is received. The Polling plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect*.

**Connect Utility – mbedTLS:** The application will use mbedTLS for CRYPTO operations. This plugin is required to share the CRYPTO hardware between the Connect stack and the application or the Bluetooth stack.

**Connect I/O – Serial:** Provides high-level read/write serial communication functionality, including `readByte()`, `readData()`, `readLine( )`, `writeByte()`, `writeHex()`, `writeString()`, `writeData()`, `writeBuffer()`, `printf()`, `guaranteedPrintf()`, `printfLine()`, `printCarriageReturn()`, and `printVarArg()`. Relies on the HAL low-level UART APIs. (Uses the serial peripheral configured in hardware configurator and two Linked Direct Memory Access (LDMA) channels.)

**Connect I/O – Command interpreter:** Provides a common framework for defining Command Line Interface (CLI) commands and for parsing serial input. Each CLI command is defined by the command string, the set of parameters, and the corresponding function to be called when a command and its parameters are successfully parsed. This command set is depicted in the Command Line Configuration section of the **Printing** tab in AppBuilder. Application developers can easily define a custom set of CLI commands. See *UG235.08: Using the Command Line Interface with Silicon Labs Connect* for details.

**Connect I/O – Heartbeat:** For use with the Wireless Starter Kit (WSTK). Periodically toggles an LED on the WSTK board. The application can set which LED to toggle and the toggling period. The Heartbeat plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect*. (Uses the HAL APIs to toggle board LEDs. (Uses the GPIO peripheral which drives the selected LED.)

**Connect I/O – SPI Flash Plugin:** Provides a set of APIs to perform read/write/erase operations on external flash parts. Supported parts are:

- Macronix MX25R8035F (8-Mbit)
- Macronix MX25R6435F (64-Mbit)

The SPI Flash plugin is also discussed in *UG235.07: Energy Saving with Silicon Labs Connect.* (Uses the SPI peripheral configured in hardware configurator.)

**Connect I/O – WSTK sensors:** The WSTK board is equipped with a temperature and humidity sensor. This plugin provides APIs to read humidity and temperature values. It initializes the sensor and reads values using the low-level HAL APIs. (Uses the $I^2C$ peripheral configured in the hardware configurator.)

**Connect HAL – HAL Library:** The HAL for the target device.

**Connect HAL – Micrium RTOS:** When enabled, the application will run within Micrium OS. For more information, see *UG235.05: Using Micrium OS (RTOS) with Silicon Labs Connect*.

**Connect HAL – Simulated EEPROM version * Library/Stub and NVM3:** Implements the supported NVM storage solutions for the Connect stack. These libraries simulate an EEPROM within the internal flash of the chip to maximize the lifetime of flash pages and reduce write cycles by wear leveling writes across the flash. This NVM is used for persistent storage of tokens for the network (automatically managed by the stack) and application layers (application can add its own tokens).

Hardware requirements:

- SimEEv1: 8kB flash is used to provide 2kB token space
- SimEEv2: 36kB flash is used to provide 8kB token space
- NVM3: Configurable storage with at least three flash pages
- All versions: 45 bytes of token space are consumed by the Connect stack. The parent support plugin might require additional space, up to 749 bytes.

For more information, see the AppBuilder description for each plugin, *AN703: Using Simulated EEPROM Version 1 and Version 2 for the EM35x and EFR32 SoC Platforms*, *AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage*, and *AN1154: Using Tokens for Non-Volatile Data Storage*.

### 2.3 Bootloader-related Plugins

Connect also includes the following plugins that either directly implement or support the Connect Over the Air (OTA) bootloader functionality. For more in-depth coverage of this feature set, see *UG235.06: Bootloading and OTA with Silicon Labs Connect*. Unless otherwise noted, all plugins are in the Connect Utility group.

**Note:** OTA Bootloader plugins are not supported in MAC mode.

**Bootloader Interface** (in the Connect HAL group)**:** Provides a set of APIs for interacting with the Gecko Bootloader.

**OTA Bootloader Server Plugins:** The server side of the OTA bootloader protocol. It includes all the functionality to distribute an image to one or multiple target devices (clients) and to instruct one or more target devices (clients) to perform an image bootload operation at a certain time in the future.

**OTA Broadcast Bootloader Server:** The broadcast server version, meant for multiple clients.

**OTA Unicast Bootloader Server:** The unicast server version, meant for a single client.

**OTA Bootloader Client Plugins**: The client side of the OTA bootloader protocol. It includes all the functionality to download an image from an OTA bootloader server and to be instructed to perform an image bootload at a certain time in the future.

**OTA Broadcast Bootloader Client:** The broadcast client version that assumes there are multiple clients receiving the same image simultaneously.

**OTA Unicast Bootloader Client:** The unicast client version that assumes only one client is receiving the image from the server.

**OTA Bootloader Test Plugins:** Provides test code to demonstrate how to perform flash read/write/erase operation locally and how to use the OTA Bootloader Server/Client plugins to perform an OTA bootloading operation. This plugin also comes with a set of CLI commands.

**OTA Bootloader Test Common:** The common part of the bootloader tests which contains the CLI commands for the external Flash part operations as well as functions that work the same way for broadcast and unicast (for example, setting the image tag on the client side).

**OTA Broadcast Bootloader Test:** The broadcast-specific bootloader test functions, such as CLI commands for setting up the target devices, starting the distribution, and requesting a bootload from the targets.

**OTA Unicast Bootloader Test:** The unicast-specific bootloader test functions. The same as above but different CLI commands with some differences to those in the broadcast commands.

### 2.4 DMP-related Plugins

Starting with Flex 2.6, Connect supports dynamic multiprotocol with Bluetooth. To enable it, you need the following plugins:

- **BLE:** Includes the Bluetooth stack itself.
- **Micrium RTOS:** Required by dynamic multiprotocol.
- **NVM3:** The Connect and Bluetooth stacks will use the common NVM3 storage.
- **mbedTLS:** When mbedTLS is not present, Connect requires direct access to the security hardware.
- **RAIL Multiprotocol Library:** Enables the dynamic multiprotocol backend in RAIL.

With dynamic multiprotocol enabled, Bluetooth will be automatically initialized during boot with the information setup in the GATT Configurator (BLE tab in Application Builder). You can implement the Bluetooth event handler in the `emberAfPluginBleEventCallback`. All callbacks that call the Connect and Bluetooth APIs—including `emberAfPluginBleEventCallback`—run as an App framework task. Calling Bluetooth and Connect Stack APIs from the App framework task is thread-safe.

# 3. Callbacks

In a typical Connect-based application the **Connect Common – Main** plugin is enabled, which defines the `main()` function for SoC applications. It calls all the required initialization functions, but also implements the stack handlers and dispatches them to every plugin that subscribes to them by calling into the bookkeeping auto-generated callbacks. While Silicon Labs provides all the source code for the Connect Application Framework (the stack and RAIL are provided as pre-compiled libraries), user-created code should live outside the framework and should interact with the framework through the Connect Application Framework API exposed by the framework utilities and callbacks.

AppBuilder generates a stub callback file called `flex-callbacks.c` that contains default implementations of all callbacks you have selected to include in your project. Hence, rather than modifying `main()` or the application framework source, in general your application code should be written within `flex-callbacks.c`. Commonly, `emberAfMainInitCallback` (for initialization) and `emberAfMainTickCallback` (iterations through the application main loop) are utilized for this purpose.

**Note:** Unused callbacks reside by default in stub form in `flex-callbacks-stubs.c`. For more information, see section 3.1 Callbacks & Stubs AppBuilder Interface.

Stack callbacks are routed through and distributed by the Application Framework. This can be observed by noting the prefix on relevant API function names (`ember` being associated with the stack API and `emberAf` associated with the Application framework API). For example, when the receipt of an incoming message causes the stack to raise `emberIncomingMessageHandler`, the message is both passed to `emberAfIncomingMessageCallback` (for processing by the application) and `emberAfIncomingMessage` (to be consumed by plugins), as dictated by `flex-bookkeeping.c`).

### 3.1 Callbacks & Stubs AppBuilder Interface

The **Callbacks** tab in AppBuilder contains a listing of available callbacks within the application framework. The origin column indicates the plugin that created and relies upon that callback. For most callbacks, you are also provided an option to set or clear two checkboxes labeled “Is used?” and “Stub?” as shown in the following figure.

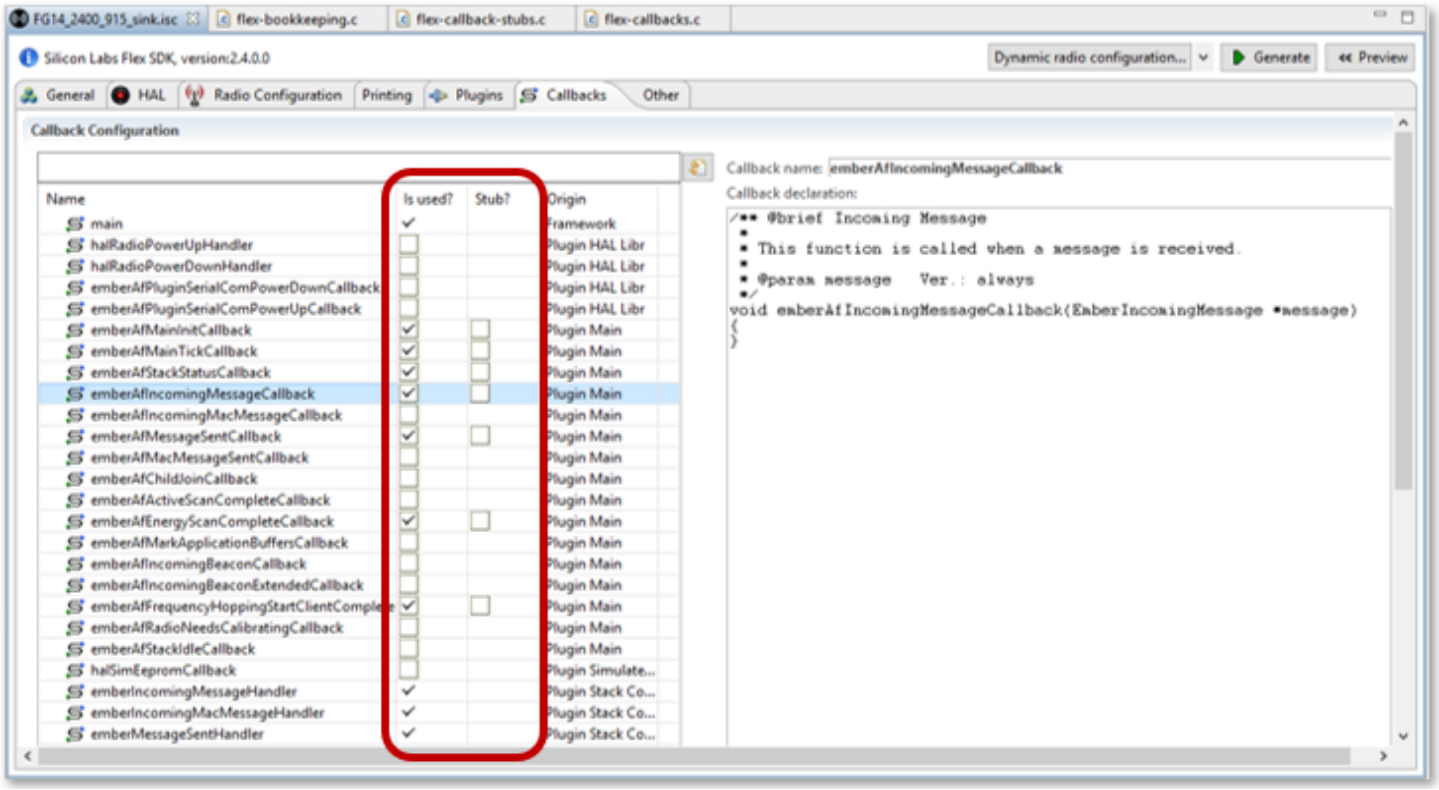



**Figure 3.1. Silicon Labs AppBuilder Callbacks Tab**

The appropriate setting for each of these checkboxes depends on the current stage of your development effort:

1. **Starting a new project**

   In this case, AppBuilder has not yet generated its output components to populate your workspace. Check both “Is used?” and “Stub?” for desired callbacks. This will ensure that AppBuilder not only expects a given callback to be present in your application (by removing its instance from `flex-callbacks-stubs.c` and creating an instance in `flex-callbacks.c`), but it will also auto-generate the callback declaration (in `flex-callbacks.c`) which provides a convenient template within which you can write your custom callback content.

2. **Editing callback configuration on an existing project**

   In this case, AppBuilder has already generated its output files at least once. When adding a new callback that was not previously included in your project, check “Is used?” (so the default instance will be purged from `flex-callbacks-stubs.c`). Avoid setting **any** of the “Stub?” checkboxes, as a selection here will cause AppBuilder to try and overwrite your existing callbacks content with the empty stubs again. You will need to directly edit `flex-callbacks.c` to add in any newly-introduced callbacks to your existing project.

3. **Removing a callback from an existing project**

   Like #2 above, you want to avoid selecting any checkboxes in the “Stub?” column to prevent overwrites to your existing `flex-callbacks.c` content. When removing an existing callback, you simply clear the associated “Is used?” checkbox and re-generate. AppBuilder will restore the default instance of the callback to `flex-callbacks-stubs.c`. You will need to ensure that the unwanted callback is manually removed from `flex-callbacks.c` to avoid conflicting with the restored default instance.

# 4. Events

The Connect Application Framework provides a simple event scheduler framework, which can be considered an extension of the call-back mechanism. You can use this to set up delayed (or immediate) events without directly using a timer—similar to how you might accomplish this behavior using an RTOS task. However, Connect does not provide mutexes, semaphores, or queues without a proper RTOS (see Note below). The events scheduler cannot be disabled, as the Connect stack also uses this feature to schedule stack events (like periodic beacon transmission).

**Unless you need tight scheduling, Silicon Labs highly recommends that you use events instead of timers in Connect.**

**Note:** Events are still available when the Micrium OS (RTOS) plugin is enabled. Micrium OS tasks use a lot of memory, so you can only create a handful of RTOS tasks. Alternatively, you can have 256 Connect events (only limited by the type of `EmberTaskId`) and you only need to allocate an `EmberEventControl` (6 bytes) for each event in RAM.

## 4.1 Creating an Event

To create a new event, append a custom entry in the *Event Configuration* section (on the **Other** tab of AppBuilder) using the **Add new** button (as shown in the following figure).

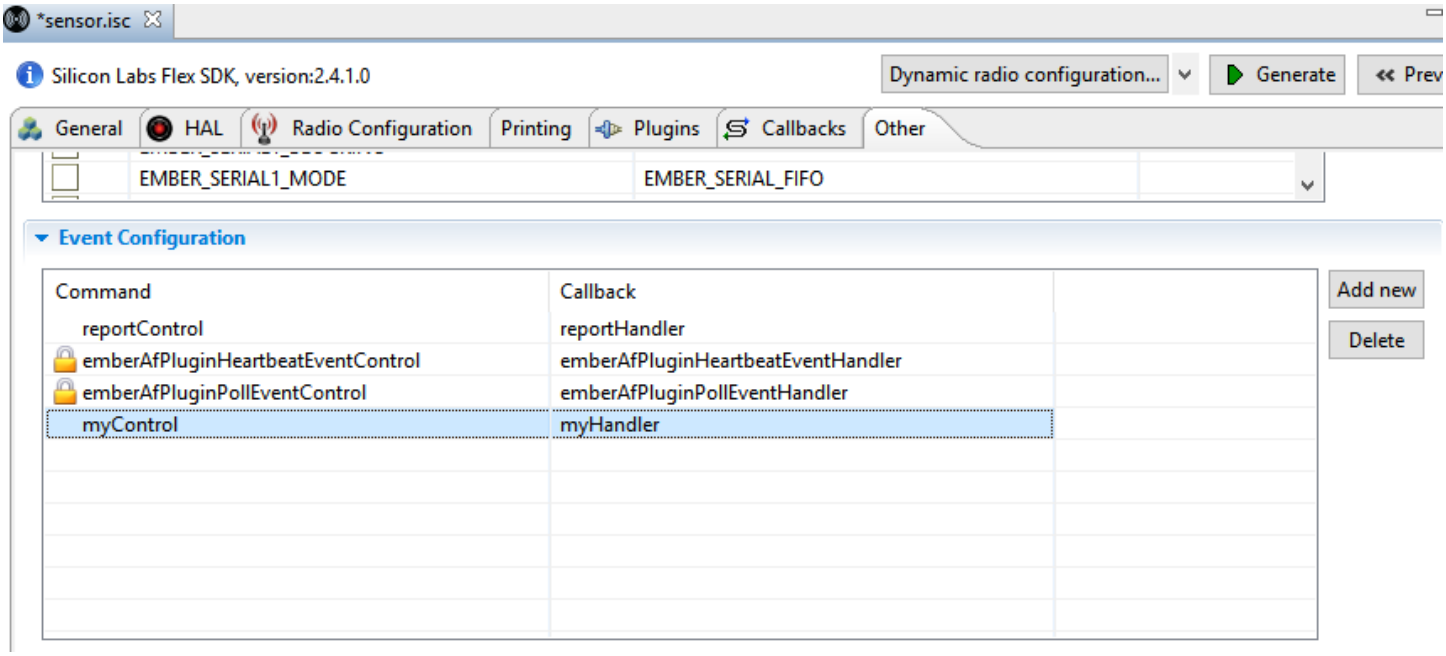



**Figure 4.1. The Event Configuration Section in AppBuilder**

This will add the event to the `emAppEvents` array in `flex-events.c`, which is used by the event scheduler. Each event requires a control (which is used to schedule it) and a callback (which is the function that will be executed when the event runs). Some events visible in the Event Configuration GUI are “locked” and cannot be deleted because they are used by plugins.

Once the event is created in the GUI, you should allocate memory for its control and implement its callback. Typically, this should be in `flex-callbacks.c` and will resemble the following:

```
EmberEventControl myControl;

void myHandler(void){
}
```

### 4.2 Scheduling Events

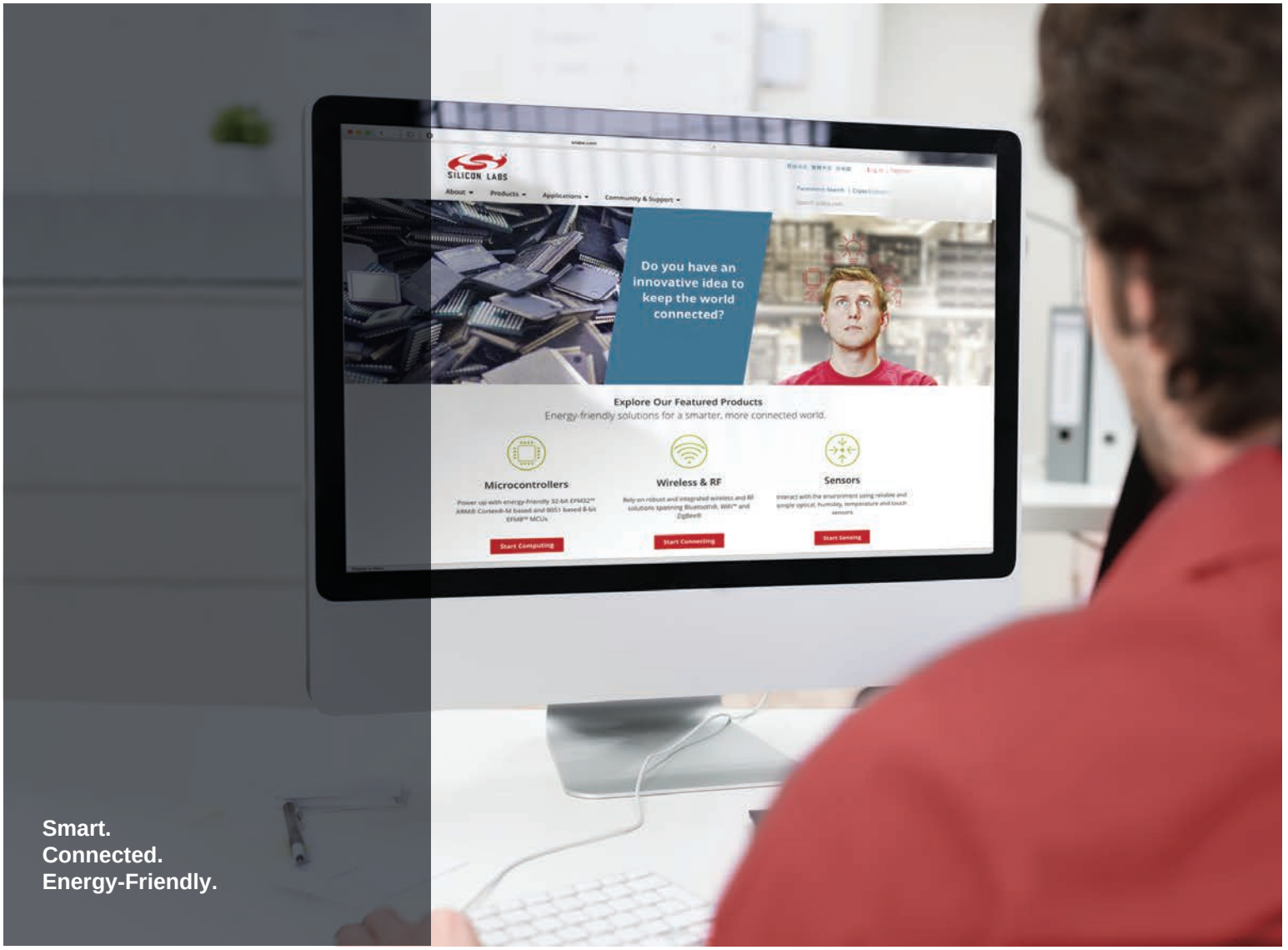
The following APIs are most commonly used to schedule events:

- `emberEventControlSetActive(EmberEventControl control)` – Schedules an event as soon as possible.
- `emberEventControlSetDelayMS(EmberEventControl control, uint32_t delay)` – Schedules an event to occur delay ms in the future.
- `emberEventControlSetInactive(EmberEventControl control)` – Turns off an event until it is scheduled again (either with `SetActive` or `SetDelay`).

  **Note:** Once the callback of the Handler is running, it will be recalled repeatedly until either it is set inactive or delayed. However, the callback function itself will run until it returns – it will not be stopped by an `emberEventControlSetDelay` command (like an RTOS). This means that a typical event handler will look something like this:

```
void myHandler(void){
  emberEventControlSetInactive(myControl); //make sure to set the event inactive, if we need to reschedule,
we'll do it later
  EmberStatus status = someTaskToPerform();
  if ( status != EMBER_SUCCESS ){ //task was unsuccessful, try again 1 second later
    emberEventControlSetDelayMS(myControl, 1000);
  }
}
```

For more details and the full available API, see the related API documentation.




**Products**
*www.silabs.com/products*

**Quality**
*www.silabs.com/quality*

**Support and Community**
*community.silabs.com*



SILICON LABS

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

**http://www.silabs.com**