UG435.04: Customizing Applications with Silicon Labs Connect v3.x

This chapter of the *Connect v3.x User's Guide* describes how to use Connect components, 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 v3.0 and higher. The *Connect v3.x User's Guide* assumes that you have already installed the Simplicity Studio® 5 development environment and the Flex SDK, and that you are familiar with the basics of configuring, compiling, and flashing Connect-based applications. Refer to *UG435.01: About the Connect v3.x User's Guide* for an overview of the chapters in the *Connect v3x User's Guide*.

The information in this chapter is designed for developers who are new to Connect. If you are experienced with Connect v2.x in Simplicity Studio 4, see *AN1254: Transitioning from the v2.x to the v3.x Proprietary Flex SDK*.

The *Connect v3x 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 *QSG168: Proprietary Flex SDK v3.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.

KEY POINTS

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

1. Connect Stack and Application Framework

Applications based on Silicon Labs Connect incorporate not only the Connect Stack, but also the Connect Application Framework. Together, they allow developers to begin a new project based on a foundation of best-practice application state machine code developed and tested by Silicon Labs. The framework sits on top of the Silicon Labs Connect stack to interface with other platform elements and provide application layer functionality. It consumes the stack handler interfaces and exposes its own more highly-abstracted and application-specific interface to developers.

Consistent with the platform architecture introduced in Gecko Software Development Kit (GSDK) v3.0, the Connect stack and application framework are structured to support optional functionality blocks called **components**. Each Connect component is provided as a standalone library or a set of source code. 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.

Each component registers one or more callbacks on top of the Application Framework, which are then implemented by the application (overriding default weakly-defined stubs). In a Connect-based project, callbacks are the places where developers add custom application code on top of the existing Silicon Labs framework to give that application unique behaviors and determine how it will react.

Within the callback implementations, developers can utilize the entire platform and stack APIs – as well as a complete set of Application Framework-specific APIs that often provide high-level wrappers around complex platform or stack functionality. These APIs are documented online at https://docs.silabs.com/connect-stack/latest (relevant pages of which are embedded within the Project Configurator in Simplicity Studio 5 (SSv5)), and examples of their usage can be found in the Connect sample code.

The following sections take a closer look at each of the fundamental elements that comprise applications based on Silicon Labs Connect: Components, Callbacks, and Events.

2. Components

The GSDK v3.x platform introduced a mechanism – called the **component library** – that empowers developers to leverage existing, proven implementations of desired functionality. The GSDK v3.x and Simplicity Studio 5 (SSv5) were architected from the ground up to support this modular component-based approach to rapid and robust application development.

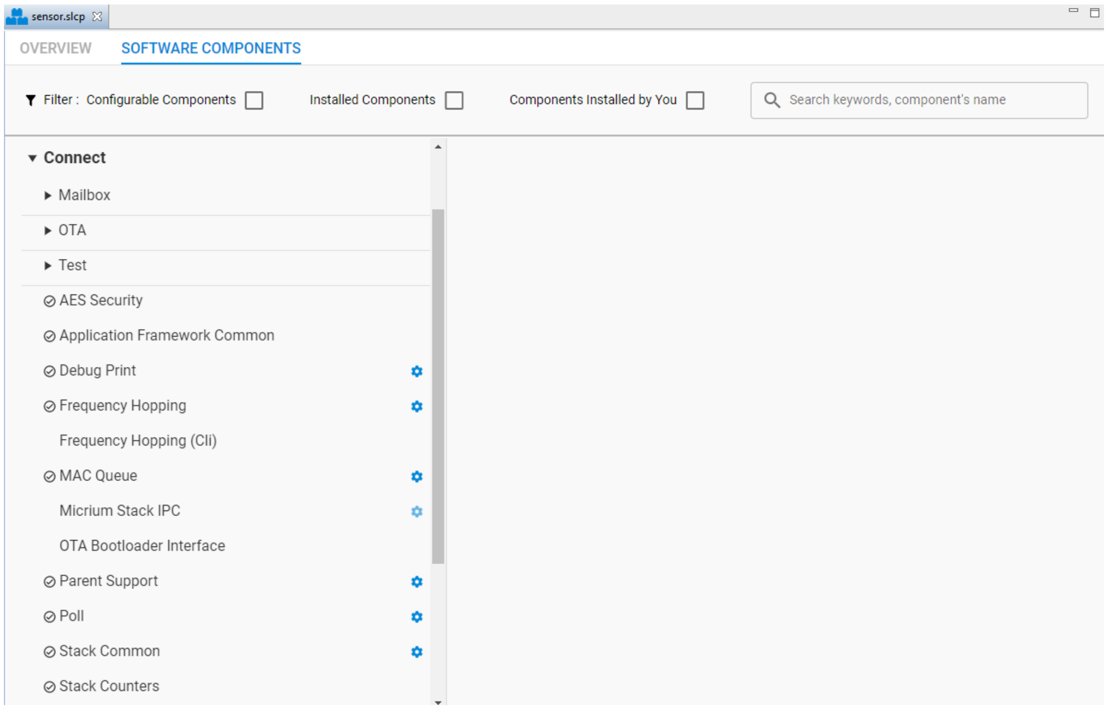
In Flex SDK v3.x, Silicon Labs Connect (both the stack and application framework) has been reformulated to integrate seamlessly with this component architecture – the Connect-specific elements are accessible to applications in the same way as other components (like the Power Manager for sleep mode transitions, board support for Silicon Labs development kit hardware, and so on) through the SOFTWARE COMPONENTS tab in the Project Configurator.

Provided as either a standalone pre-compiled library or a set of source code, each component adds a specific feature set to the project. Use the Project Configurator to install desired (or uninstall unwanted) components to add (or remove) the associated code and libraries to/from the build. Many components have configurable options, which can be adjusted directly in the underlying header files or conveniently managed using the Component Editor. Reference information on component descriptions and behaviors – including component dependencies and dependents – is also easily accessible from the Project Configurator.

More information on using the component library is available in the Simplicity Studio 5 User's Guide, available through SSv5 and online at https://docs.silabs.com.

2.1 Connect Features in the Component Library

This section describes some of the Connect components currently available. For a complete and current list (including inter-component dependencies), see the SOFTWARE COMPONENTS tab of a Connect project using the latest Flex SDK. These components can be found beneath the top-level “Connect” group, as shown in the following figure.



2.1.1 MAC and Network Layer Components

The component library includes the following Media Access Control (MAC) and Network layer Connect components:

- AES Security
- Frequency Hopping
- MAC Queue
- Parent Support
- Stack Common

AES Security: Enables nodes to exchange secured messages with IEEE 802.15.4 mode-5-like or mode-5 MAC encryption/authentication using Mbed TLS. This takes advantage of hardware-accelerated AES support on EFR32 devices. Install this component only if the application will use/supports AES security.

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 *U435.03: Architecture of the Silicon Labs Connect Stack v3.x*.

MAC queue: Some applications need to submit multiple messages to the Connect stack. This component 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. Without this component installed, the Connect stack can only process one message transmit request at a time.

Parent Support Provides parent features such as indirect communication (communication with sleepy devices), child table, and routing table support. It should be installed for any coordinator or range extender node intended to support multiple end-device and/or sleepy end-device nodes. To save flash/RAM, star end devices, direct devices, or MAC devices should not install this component. The Parent Support component is also discussed in *UG435.07: Energy Saving with Silicon Labs Connect v3.x*.

Indirect queue support buffers packets destined for sleepy child devices.

The 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 section 2.2 Connect Application Dependency on Non-Connect Components.) The child table size of a coordinator is limited to 64, while the range extender child table sizes are limited to 32.

A routing table is needed for star coordinator applications in extended star networks (that is, the network includes star range extenders). It stores the information collected from star range extenders.

Stack Common: Provides Connect Stack common functionalities such as 15.4-like MAC layer, events system, dynamic memory allocation, and other required infrastructure functions. It also provides the stack configuration file, which allocates all the sizable RAM data structures. Should be installed in any Connect application.

2.1.2 Application Framework Components

The optional Connect Application Framework components are used to manage application layer functionality as follows:

- Mailbox
  - Mailbox client
  - Mailbox server
- Application Framework Common
- Debug Print
- Frequency Hopping (CLI)
- CMSIS Stack IPC
- Poll
- Stack Counters

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 component supports much longer timeouts (default 1 hour, can be set to days) than the indirect queue (between 8 and 30 seconds). The Mailbox component is not supported in MAC mode. The Mailbox feature is also discussed in *UG435.07: Energy Saving with Silicon Labs Connect v3.x*.

Application Framework Common: Declares all the required application framework globals, initializes the Connect stack and dispatches stack callback calls as needed to the application components.

Debug Print: Provides APIs for adding serial debugging capability (printf-style printing from the application).

Frequency Hopping (CLI): Provides frequency hopping CLI commands.

CMSIS Stack IPC: Provides an IPC (Inter-process communication) for the Connect stack APIs. If this component is installed, the stack will be running within an RTOS (Real time operating system) as an RTOS task and the application framework will run as a separate task. APIs are invoked at the application framework using an IPC. Similarly, stack callbacks are communicated to the application framework task using an IPC. For more information, see *UG435.05: Using Real Time Operating Systems with Silicon Labs Connect v3.x*.

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 component API. 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 component is also discussed in *UG435.07: Energy Saving with Silicon Labs Connect v3.x*.

Stack Counters: Provides stack packet counters functionality. If this component is installed, the Connect stack keeps track of successful and failed transmissions as well as successful received packets and dropped incoming packets.

2.1.3 Bootloader-Related Components

Connect provides the following components that either directly implement or support the Connect Over-the-Air (OTA) bootloader functionality. For more in-depth coverage of this feature set, see *U435.06: Bootloading and OTA with Silicon Labs Connect v3.x*.

Note: OTA Bootloader components are not supported in MAC mode.

- OTA
  - OTA Broadcast Bootloader Client
  - OTA Broadcast Bootloader Server
  - OTA Unicast Bootloader Client
  - OTA Unicast Bootloader Server
- Test
  - OTA Bootloader Test Common
  - OTA Broadcast Bootloader Test
  - OTA Unicast Bootloader Test
- OTA Bootloader Interface

Bootloader Interface : Provides a set of APIs for interacting with the Gecko Bootloader.

OTA Bootloader Server components implement the server side of the OTA bootloader protocol. They include all the functionality to distribute an image to one or multiple target devices (clients) and to instruct one or more 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 components implement the client side of the OTA bootloader protocol. They include 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 components provide test code to demonstrate how to perform flash read/write/erase operation locally and how to use the OTA Bootloader Server/Client components to perform an OTA bootloading operation. These components also support 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.

2.2 Connect Application Dependency on Non-Connect Components

With the advent of GSDK v3.x and its unified approach to platform modularization and configuration, prior support for some Connect-specific implementations of common features has been replaced by universal stack-agnostic solutions elsewhere in the component library. This includes the (former) Idle/Sleep and NVM “plugins”. Note: Developers who have worked previously with Connect in SDKs prior to v3 should consult *AN1254: Transitioning from the v2.x to the v3.x Proprietary Flex SDK*.

Device energy mode transitions are now overseen by the Power Manager component (for more information, see *UG435.07: Energy Saving with Silicon Labs Connect v3.x*).

Connect applications require emulated nonvolatile memory support. Silicon Labs provides a number of such solutions, located in the Services component group (NVM3 is strongly recommended). 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 (the application can add its own tokens).

These tokens incur the following hardware requirements:

- **SimEEv1**: 8 kB flash is used to provide 2 kB token space
- **SimEEv2**: 36 kB flash is used to provide 8 kB token space
- **NVM3** (strongly recommended): Configurable storage with at least three flash pages
- All versions: 47 bytes of token space are consumed by the Connect stack. For end devices this represents the complete token storage footprint. Coordinators and range extenders can require up to an additional 704 bytes due to the Parent Support component (n * 11 bytes, to support up to n children, where n is between 0 and 64).

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

3. Callbacks

Connect-based applications include the Stack Common and Application Framework Common components, which not only provide the required initialization actions but also implement the stack handlers and dispatch them to every dependent component callback. 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.

GSDK v3 projects include a *main.c* file (Connect-based applications should leave this unaltered) that initializes the stack and runs its main loop. The stack will call `emberAfInitCallback()` (in *app_init.c*) at the end of its initialization and `emberAfTickCallback()` (in *app_process.c*) from the main loop. Using these functions for init and main loop actions empowers the Connect stack to prioritize operations critical to the timely execution of stack and radio tasks.

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 components), as dictated below in *app_framework_stack-cb.c* (from gecko_sdk_3*/protocol/flex/app-framework-common/).

```
void emberIncomingMessageHandler(EmberIncomingMessage *message)
{
  emberAfIncomingMessage(message);
  emberAfIncomingMessageCallback(message);
}
```

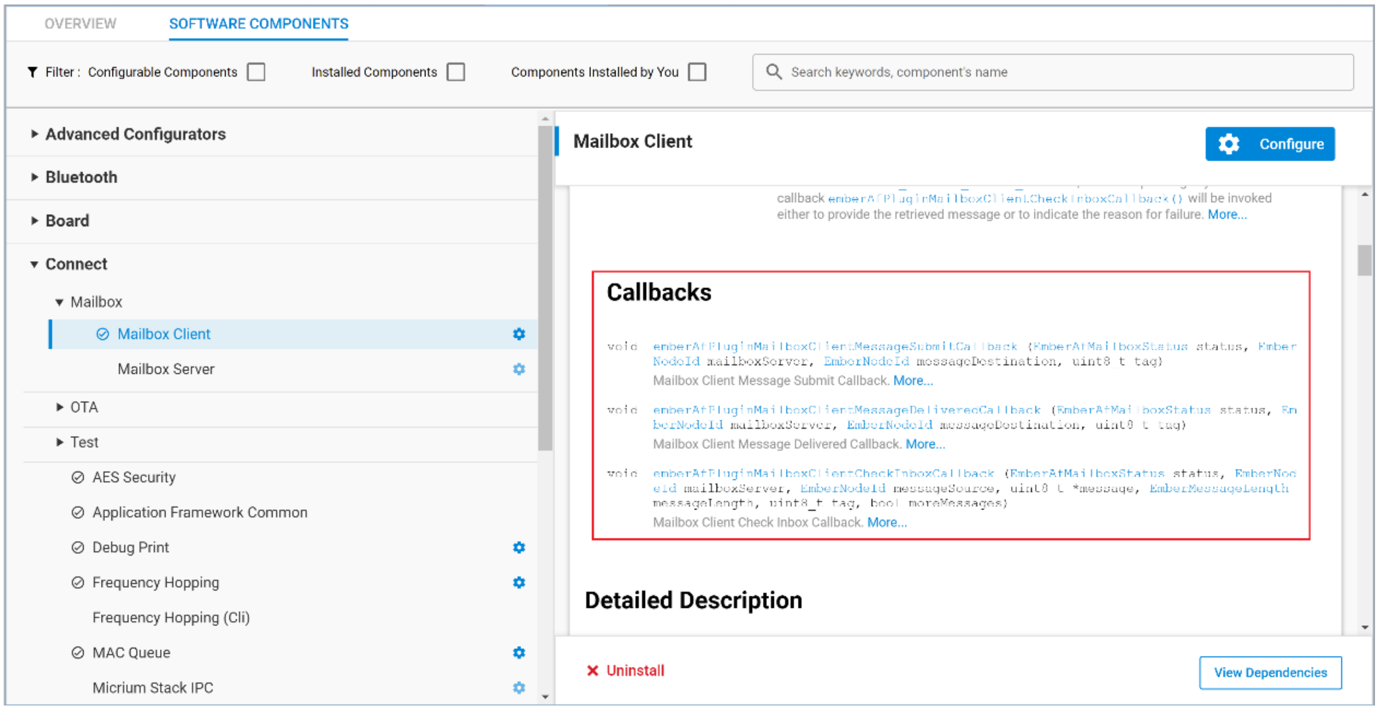
When installing a component, note the callback(s) prototype code provided to the right of the component list in the Project Configurator. Use these reference examples to implement the necessary callbacks in a .c file, typically *app_process.c* for your installed components, as illustrated in the following example.

3.1 Implementing Callbacks to Support Installed Components

As an example, when installing the Mailbox Client component in the Software Configurator, a number of changes are automatically generated within the project to support the new component. These include:

- A configuration file (*mailbox-client-config.h*) is copied into the /config directory.
- The /mailbox SDK directory is copied or linked into /gecko_sdk_3.0.0/protocol/flex.
- Necessary include directories are added to the project settings.
- Files under /autogen are modified where necessary to support the functionality introduced by the component.

Although they are weakly defined in the SDK (in this case, */gecko_sdk_3.0.0/protocol/flex/mailbox/mailbox-client/mailbox-client-cb.c*), before you can make use of a newly-installed component the application often must implement the associated callbacks. To facilitate this task, prototypes are provided where necessary in the embedded documentation displayed in the Project Configurator:



callback emberAfPluginMailboxClientCheckInboxCallback() will be invoked either to provide the retrieved message or to indicate the reason for failure. More...

Callbacks

```
void  emberAfPluginMailboxClientMessageSubmitCallback (EmberAfMailboxStatus status, EmberNodeId mailboxServer, EmberNodeId messageDestination, uint8_t tag)
```
Mailbox Client Message Submit Callback. More...

```
void  emberAfPluginMailboxClientMessageDeliveredCallback (EmberAfMailboxStatus status, EmberNodeId mailboxServer, EmberNodeId messageDestination, uint8_t tag)
```
Mailbox Client Message Delivered Callback. More...

```
void  emberAfPluginMailboxClientCheckInboxCallback (EmberAfMailboxStatus status, EmberNodeId mailboxServer, EmberNodeId messageSource, uint8_t *message, EmberMessageLength messageLength, uint8_t tag, bool moreMessages)
```
Mailbox Client Check Inbox Callback. More...

Detailed Description

The figure shows three callbacks available to the application when Mailbox Client is installed:

- `emberAfPluginMailboxClientMessageSubmitCallback`,
- `emberAfPluginMailboxClientMessageDeliveredCallback`, and
- `emberAfPluginMailboxClientCheckInboxCallback`.

To complete the installation of the Mailbox Client component, copy these prototypes into *app_process.c* and implement the callback code required by your application.

4. Events

The Connect Application Framework provides a simple event scheduler, which can be considered an extension of the callback 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 RTOS kernel component is enabled. RTOS tasks usually use a lot of memory, so you can only create a handful of RTOS tasks. Alternatively, you can have 6 Connect events, and you only need to define a pointer variable to an `EmberEventControl` structure for each event.

4.1 Creating an Event

Events are created at run-time using `emberAfAllocateEvent()`, which is included with the Application Framework Common component.

This will add the event to the `emAppEvents` array (defined in *autogen/app_framework_event_table.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 scheduled event runs). Note: The events that populate `emAppEvents` in source originate from the stack and installed components, and are regenerated upon component install/uninstall operations – overwriting any user edits.

Be sure to define an EmberEventControl type pointer variable in your application for the control and implement the callback for any events you plan to use. Typically, this should be in *app_process.c*, and will resemble the following:

```
EmberEventControl *myControl;

void myHandler(void){
}
```

See https://docs.silabs.com/connect-stack/3.0/group-app-framework-common#gaf720508a9ada3dece4864ea2b320c9d7 for more information.

4.2 Scheduling Events

Once the callback of the scheduled event handler is running, it will be re-called repeatedly until either 1) it is set inactive, or 2) it is delayed. However, the callback function itself will run until it returns – it will not be stopped by an `emberEventControlSetDelayMS` command (like an RTOS).

Therefore, a primary responsibility for an event callback is to carefully manage the scheduling of the next repetition of the source event. 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 `SetDelayMS`).

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 at https://docs.silabs.com/connect-stack/latest/group-event.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com