



# QSG138: Proprietary Flex SDK v2.x Quick Start Guide

---

This quick start guide provides basic information on configuring, building, and installing applications for the EFR32 using the Silicon Labs Flex SDK (Software Development Kit). The Flex SDK provides two paths to application development. The first uses Silicon Labs Connect, which provides a fully-featured, easily-customizable wireless networking solution optimized for devices that require low power consumption and are used in a simple network topology. The second begins with Silicon Labs RAIL (Radio Abstraction Interface Layer), which provides an intuitive, easily-customizable radio interface layer that is designed to support proprietary or standards-based wireless protocols.

This guide is designed for developers who are new to the Silicon Labs Flex SDK, the Simplicity Studio development environment, and Silicon Labs development hardware. It provides instructions to get started using both the Connect and RAIL examples provided with the Flex SDK used on the EFR32.

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

---

- Product overview
- Setting up your development environment
- Discovering your SDK
- Working with example applications.
- Using the Hardware Configurator

# 1 Flex SDK Product Overview

The Silicon Labs Flex SDK supports developers who wish to take advantage of configurable protocol functionality provided in Silicon Labs Connect and the underlying RAIL library, as well as those who wish to start application development on top of RAIL but develop custom lower-level radio and network protocols.

This section covers:

- Background information on RAIL, Connect, and the example applications included with the Flex SDK
- Background information on the Gecko Bootloader
- Prerequisites for application development using the Flex SDK
- Support
- Documentation

## 1.1 About Connect and RAIL

The Flex SDK provides two paths to application development. The first uses Silicon Labs Connect, which provides a fully-featured, easily-customizable wireless networking solution optimized for devices that require low power consumption and are used in a simple network topology. Connect example functionality is provided through easily-configurable plugins that can be turned on or off as desired. The second begins with Silicon Labs RAIL (Radio Abstraction Interface Layer), which provides an intuitive, easily-customizable radio interface layer that is designed to support proprietary or standards-based wireless protocols.

Whether you begin development with Connect or RAIL is determined by the example application you select as a starting point in Simplicity Studio. Silicon Labs recommends that you start from a Connect example if you want to include the following functions without further development:

- MAC layer functionality including frequency hopping and security
- Network formation and, for star networks, routing support
- Application-level functionality, including diagnostics, I/O, mailbox, and sleepy end device management
- Bootloading, including serial and Broadcast or Unicast OTA (over-the-air)
- Host and NCP mode

The following sections provide additional detail on Connect and RAIL, including brief descriptions of the example applications. When you create a project based on an example, the description on the Simplicity Studio IDE General tab provides additional detail about the example and interfacing with it.

### 1.1.1 Silicon Labs Connect

Silicon Labs Connect functionality for the EFR32 is implemented on top of the RAIL library. Silicon Labs Connect supports many combinations of radio modulation, frequency and data rates. The stack includes all MAC layer functions such as scanning and joining, setting up of a point-to-point or star network, device types such as sleepy end nodes, routers or coordinators, radio configuration, frequency hopping and LBT (Listen Before Talk) protocols required for regulatory compliance in each geographical region, and PHY configuration for each of these regions. With all this functionality already implemented in the stack, developers can focus on their application development and not worry about the lower level radio and network details.

The Flex SDK includes the following Connect example applications. Not all examples are accessible through the list on the Launcher perspective. Specifically, most host examples are only accessible through the **New Project** project flow.

**Connect (SoC): Empty Example:** A minimal Connect project structure, used as a starting point for custom applications.

**Connect (SoC): Empty Example - BLE:** A dynamic multiprotocol minimal project structure, used as a starting point for custom applications that run both Connect and Bluetooth protocols.

**Connect (SoC): Commissioned Device:** Demonstrates direct communication between nodes in range. The network parameters are commissioned by the application.

**Connect (SoC): Demo Connect Light:** Demonstrates a light application that can be turned on/off by a switch application. This example is part of the Connect Light/Switch dynamic multiprotocol demo setup. See *AN1209: Dynamic Multiprotocol Development with Bluetooth and Connect* for more information on working with the Dynamic Multiprotocol examples.

**Connect (SoC): Demo DMP Connect Switch:** Demonstrates a switch application using dynamic multiprotocol (Connect + Bluetooth). This example is part of the Connect Light/Switch dynamic multiprotocol demo setup.

**Connect (SoC): MAC Mode Device:** Demonstrates direct MAC mode communication between nodes in range.

**Connect (SoC): Sensor** and **Connect (SoC): Sink:** The sensor and sink applications demonstrate how to set up a star network topology in which communication occurs in both directions between the sink and the sensor(s) nodes.

**Connect (SoC): Wire-Replacement:** Demonstrates point-to-point bi-directional direct or indirect communication between two nodes.

**Connect (Host): Commissioned Direct Device:** Demonstrates direct communication between nodes in range. The network parameters are commissioned by the application. It runs on Unix UART Host, with EFR32 NCP.

**Connect (Host): Empty Example:** A minimal Connect project structure, used as a starting point for custom applications. It runs on Unix UART Host, with EFR32 NCP.

**Connect (Host): MAC Mode Device:** Demonstrates direct MAC mode communication between nodes in range. It runs on the Unix UART host, with EFR32 NCP.

**Connect (Host): Sensor** and **Connect (Host): Sink:** Demonstrates a star network topology setup. Bi-directional communication is possible between the sensor(s) and the sink nodes. It runs on Unix UART Host, with EFR32 NCP.

**Connect (Host): Wire-Replacement:** Demonstrates point to point bi-directional direct or indirect communication between two nodes. It runs on Unix UART Host, with EFR32 NCP.

**Connect (NCP): UART HW (Hardware Flow Control):** This network coprocessor (NCP) application supports communication with a host application over a UART interface with hardware flow control. It runs on an EFR32.

**Connect (NCP) UART SW (Software Flow Control):** This network coprocessor (NCP) application supports communication with a host application over a UART interface with software flow control. It runs on an EFR32. For more information about XON/XOFF software flow control, see *AN844: Guide to Host/Network Co-Processor Communications Using Silicon Labs Thread and Connect*.

### 1.1.2 Silicon Labs RAIL

Silicon Labs RAIL provides an intuitive, easily-customizable radio interface layer designed to support proprietary or standards-based wireless protocols. RAIL is delivered as a library that you can link to your applications. A description of library functions is provided in the development environment. The RAIL API is documented in an online API reference available through Simplicity Studio.

RAIL:

- Implements commonly-used radio functionality, so that it does not have to be written in the application or stack.
- Eliminates the need for developers to become expert in RF register details of complex wireless SoCs.
- Simplifies code migration to new wireless ICs and the development of new stacks by providing a common radio interface layer.
- Allows lab evaluation in addition to application code development.

The RAIL library supports APIs for:

- General Radio Operation
- Channel definition and selection
- Output power configuration
- Transmit
- Clear Channel Assessment before Transmit
- Scheduled Transmit
- Energy Detection
- Receive
- Packet Filtering
- Calibration
- CW (Carrier Wave) Transmission
- Modulated Transmission
- RFSense configuration as wake source

The Flex SDK includes example RAIL application code to demonstrate the capabilities of the device and the RAIL library. These examples are provided as source code to offer a starting point for application development. The following examples are included in the current release.

## RAIL: RAILTEST

RAILtest is a general test tool for the RAIL library. RAILtest is developed by the core engineering team working on the RAIL library. As each RAIL library feature is implemented, a RAILtest serial command is added to allow scripted testing and ad hoc experimentation. Many of the RAILtest serial commands can be used for lab evaluation.

RAILtest includes commands to:

- Transmit and receive packets.
- Schedule transmits at a specific time in the RAIL timebase.
- Configure RAIL address filtering to receive only specific packets.
- Enable CCA mechanisms (CSMA/LBT) to validate that a channel is clear before transmit.
- Set a timer callback in the RAIL timebase to see how the RAIL timer API works.
- Change the transmit channel within the current configuration's band.
- Change the transmit power level.
- Enable RF energy sensing of specified duration across the 2.4 GHz and/or Sub-GHz bands, and sleep to wake on this event.
- Output a continuous unmodulated tone for debugging.
- Output a continuous modulated PN9 stream for debugging.
- Enter into direct mode where data can be sent and received using asynchronous GPIOs as input and output.

## RAIL:Range Test

The Range Test examples enable over-the-air range testing between two devices customized with user-defined parameters. Range Test is designed to be run on the Silicon Labs WSTK hardware without the need for commands from a host computer. This capability allows for mobility during range testing activities.

**RAIL: Switch:** Demonstrates the simplest exchange of transmit and receive operation between a light and a switch. May be used with the Bluetooth Light application in the Bluetooth/RAIL multiprotocol example, as described in *QSG155: Using the Silicon Labs Dynamic Multiprotocol Demonstrations*. A precompiled binary demo of this example is also available for some platforms.

**RAIL: Light:** Demonstrates the simplest exchange of transmit and receive operation between a RAIL light and a RAIL switch. The light is capable of periodically and on change reporting its status back to the switch. This is not the dynamic multiprotocol light example application.

**RAIL: Simple TRX Multi-PHY:** Demonstrates the use of multiple PHYs selectable by channels. By default, channel 0 is configured to 2.4 GHz, 250 kbps, and channel 1 is configured to 915 MHz, 500 kbps (both packets are receivable by a single-PHY application using the correct pre-configured PHY). For details see *AN971: EFR32 Radio Configurator Guide*.

**RAIL: Simple TRX:** Demonstrates the simplest transmit and receive functions based on RAIL.

**RAIL Simple TRX with ACK (Software):** Demonstrates the simplest exchange transmit and ack operation between two nodes, based on RAIL.

**RAIL Simple TRX with FIFO (Long Packet):** Demonstrates the simplest FIFO data transmission operation between two nodes, based on RAIL.

**RAIL: Connected Motion for EFR32 Thunderboard:** Demonstrates communication between nodes where lost packets is of no concern. In this demonstration, each node has a different color and lights up when it becomes active via motion detection. The active node propagates its color to nearby nodes. When all nodes are stationary, a master node coordinates a green light fading from board to board.

**RAIL: Duty Cycle:** Includes three modes of operation (Duty Cycle (both nodes are in the same mode), Master, and Slave) and demonstrates low power applications using the EFR32.

**RAIL: Energy Mode:** Demonstrates the low power modes (EM0-Active, EM1-Sleep, EM2-Deep Sleep). of the EFR32.

**RAIL: Simple RAIL with HAL:** Simple RAIL with HAL example.

**RAIL: Simple RAIL without HAL:** Simple RAIL without HAL example.

**RAIL WMBus Meter:** Implements a Wireless M-Bus meter application. For details, see *AN1119: Using RAIL for Wireless M-Bus Applications with EFR32*. Uses the multi-PHY configurator and is capable of limited multi-PHY features, like asymmetric bidirectional modes. For details see *AN971: EFR32 Radio Configurator Guide*.

**RAIL WMBus Collector:** Implements a Wireless M-Bus collector application. For details, see *AN1119: Using RAIL for Wireless M-Bus Applications with EFR32*. Uses the multi-PHY configurator. For details see *AN971: EFR32 Radio Configurator Guide*.

## 1.2 The Gecko Bootloader

A bootloader is a program stored in reserved flash memory that can initialize a device, update firmware images, and possibly perform some integrity checks. Silicon Labs networking devices use bootloaders that perform firmware updates in two different modes: standalone (also called standalone bootloaders) and application (also called application bootloaders). An application bootloader performs a firmware image update by reprogramming the flash with an update image stored in internal or external memory. Silicon Labs recommends that you always flash a bootloader image along with your application, so that flash memory usage is appropriately allocated from the beginning. For more information about bootloaders see *UG103.6: Bootloader Fundamentals*.

In March of 2017, Silicon Labs introduced the Gecko Bootloader, a code library configurable through Simplicity Studio's IDE to generate bootloaders that can be used with a variety of Silicon Labs protocol stacks. The Gecko Bootloader can be used with EFR32MG1/EFR32BG1 (EFR32xG1) however, beginning with the EFR32MG12/ EFR32BG12/ EFR32FG12 (EFR32xG12) platform, it and all future EFR32MG, EFR32FG, and EFR32BG releases will use the Gecko Bootloader only. Legacy Ember bootloader applications for use with specific protocols such as Silicon Labs Thread and platforms including the EM3x will continue to be provided for use with those platforms.

The bootloaders work with specialized firmware update image formats. The legacy Ember bootloader update images end in extension .ebi; Gecko Bootloader update images end in extension .gbl. When you build an application both .s37 and update image files are generated. The update image file format depends on the hardware you selected. EBL files are generated for EM3x and EFR32xG1. GBL files are generated for EFR32xG12 and later devices. If you want to use the Gecko Bootloader on EFR32xG1 devices, you must convert the .S37 file using Simplicity Commander, as described in *UG162: Simplicity Commander User Guide*.

Examples provided for the EFR32xG12 and newer parts include Silicon Labs Gecko Bootloader examples. Examples are provided for all compatible Simplicity Studio SDKs. For more information on using the Gecko Bootloader see *UG266: Silicon Labs Gecko Bootloader User Guide*.

**Note:** When working with the Gecko Bootloader, you must use Simplicity Commander to enable some configuration options such as security features.

## 1.3 Gecko Platform

The Gecko Platform is a set of drivers and other lower layer features that interact directly with Silicon Labs chips and modules. Gecko Platform components include EMLIB, EMDRV, NVM3, and mbedTLS. The RAIL library, accessed through the RAIL example applications included in this SDK, is also considered part of the Gecko Platform. For more information see the Gecko Platform release notes found in Simplicity Studio's Launcher Perspective under **SDK Documentation > Flex SDK 6.6.n.n > Release Notes**.

## 1.4 Prerequisites

Before following the procedures in this guide you must have

- Purchased one of the Wireless Gecko (EFR32) Portfolio Wireless Kits (As of this writing, all EFR32FG kits and EFR32MG 434/868/915 are supported), although some features may only function on a subset of these.
- Downloaded the required software components. A card included in your development hardware kit contains a link to a Getting Started page, which will direct you to links for the Silicon Labs software products. See the Flex SDK release notes for version restrictions and compatibility constraints for Connect and RAIL and these components. To develop Silicon Labs Connect- or RAIL-based applications, you will need the following software components. See section [2 Setting Up Your Development Environment](#) for details.
  - The Simplicity Studio v4 development environment, which incorporates AppBuilder. AppBuilder is an interactive GUI tool that allows you to configure a body of Silicon Labs-supplied code to implement applications. Online help for AppBuilder and other Simplicity Studio modules is provided.
  - The Silicon Labs Flex SDK, installed through Simplicity Studio.
  - (optional) IAR Embedded Workbench for ARM (IAR EWARM). See the Release Notes for the IAR version supported by this version of the Flex SDK. This can be used as a compiler in the Simplicity Studio development environment as an alternative to

GCC (The GNU Compiler Collection), which is provided with Simplicity Studio. Download the supported version from the Silicon Labs Support Portal, as described at the end of section [2.2 Install Simplicity Studio and the Flex SDK](#). Refer to the “QuickStart Installation Information” section of the IAR installer for additional information about the installation process and how to configure your license.

- Simplicity Commander, installed along with Simplicity Studio. A GUI with limited functionality can be accessed through Simplicity Studio’s Tools menu. Most functions are accessible through a CLI invoked by opening a command prompt in the Simplicity Commander directory (\SiliconLabs\SimplicityStudio\v4\developer\adapter\_packs\commander). See *UG162: Simplicity Commander Reference Guide* for more information.

While Simplicity Studio and Simplicity Commander can be run on a Mac OS or Linux machine, these instructions assume you are working with a Microsoft Windows-based PC. If you are using a non-Windows system and using IAR-EWARM, IAR-EWARM must be run via WINE or some other form of emulator or virtual machine.

## 1.5 Support

You can access the Silicon Labs support portal at <https://www.silabs.com/support> through Simplicity Studio’s Resources tab, as described in section [3.4 Accessing Documentation and Other Resources](#). Use the support portal to contact Customer Support for any questions you might have during the development process.

## 1.6 Documentation

Flex SDK documentation is accessed through Simplicity Studio, as described in section [3.4 Accessing Documentation and Other Resources](#). See section [6 Document Summary for Connect and the RAIL API](#) for a review of related documentation. Simplicity Studio also provides links to hardware documentation and other application notes. See the release notes for details on other documentation available.

## 2 Setting Up Your Development Environment

### 2.1 Connect your Hardware

Connect your WSTK, with radio board mounted, to the PC on which you will install Simplicity Studio. By having it connected when Simplicity Studio installs, Simplicity Studio will automatically obtain the relevant additional resources it needs.

**Note:** For best performance in Simplicity Studio, be sure that the power switch on your WSTK is in the Advanced Energy Monitoring or “AEM” position, as shown in the following figure.

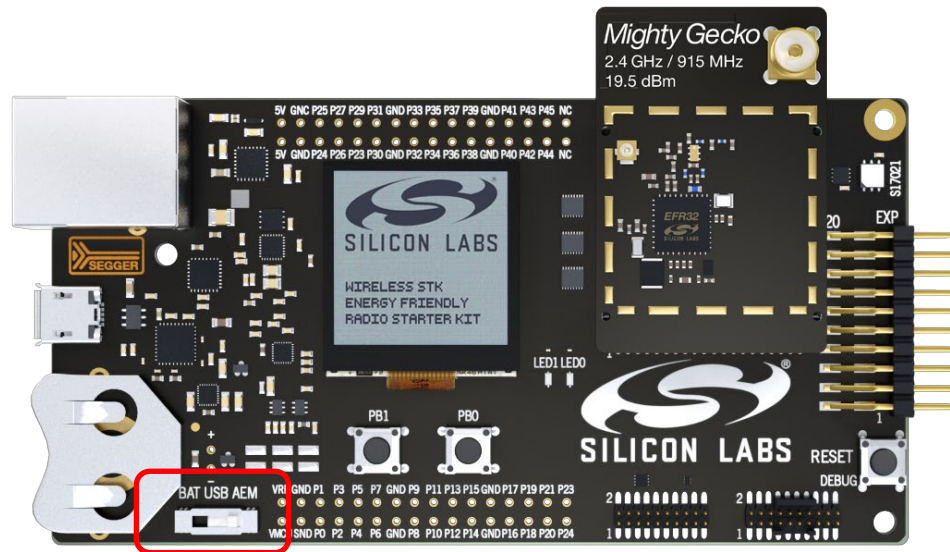
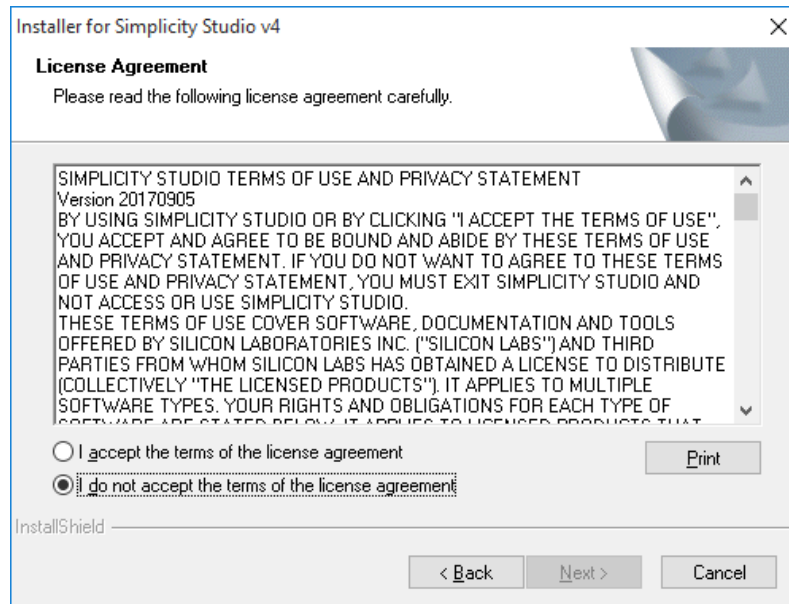


Figure 1. EFR32MG on a WSTK

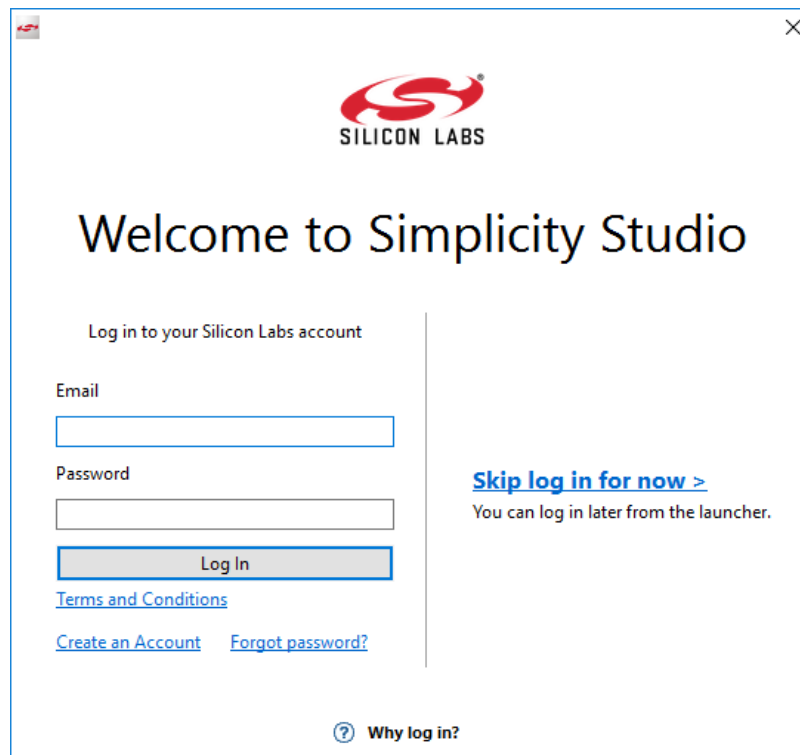


## 2.2 Install Simplicity Studio and the Flex SDK

1. Run the Simplicity Studio installation application.
2. When the Simplicity Studio installer first launches, it presents a License Agreement dialog. Accept the terms of the agreement and click **Next >**.

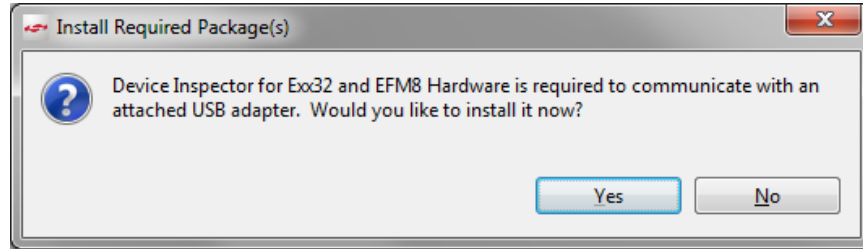


3. Choose a destination location, click **Next >** and then click **Install**.
4. When the application launches, you are invited to log in. log in using your support account username and password. Although you can skip log in here, you must log in later to download the Flex SDK.

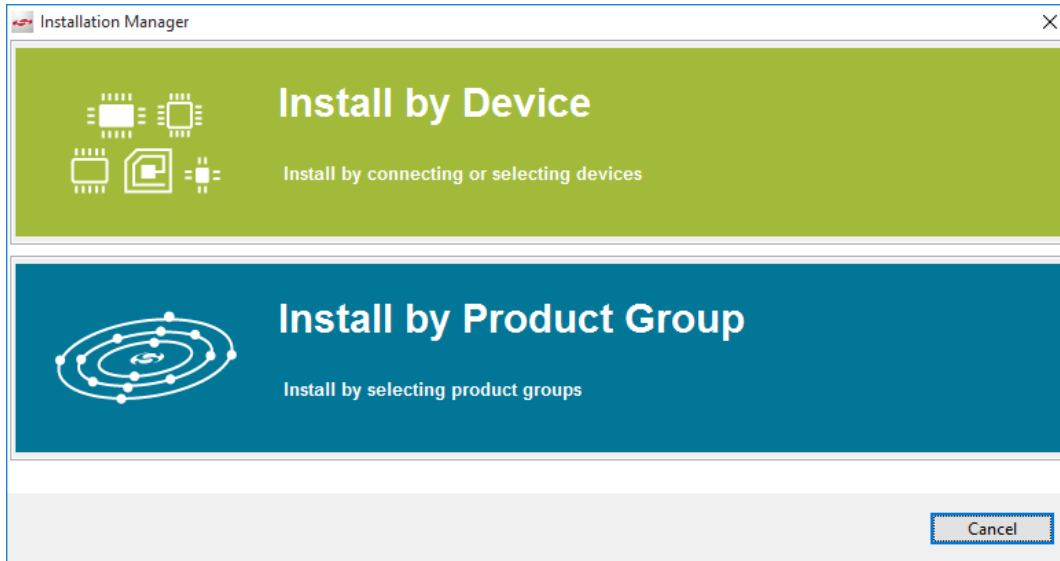




- After login, Simplicity Studio adds software information. Once initial software installation is complete, Simplicity Studio checks for connected hardware. If you have the WSTK connected by USB cable, Simplicity Studio will detect the USB cable and prompt you to download a Device Inspector. Click **Yes**.

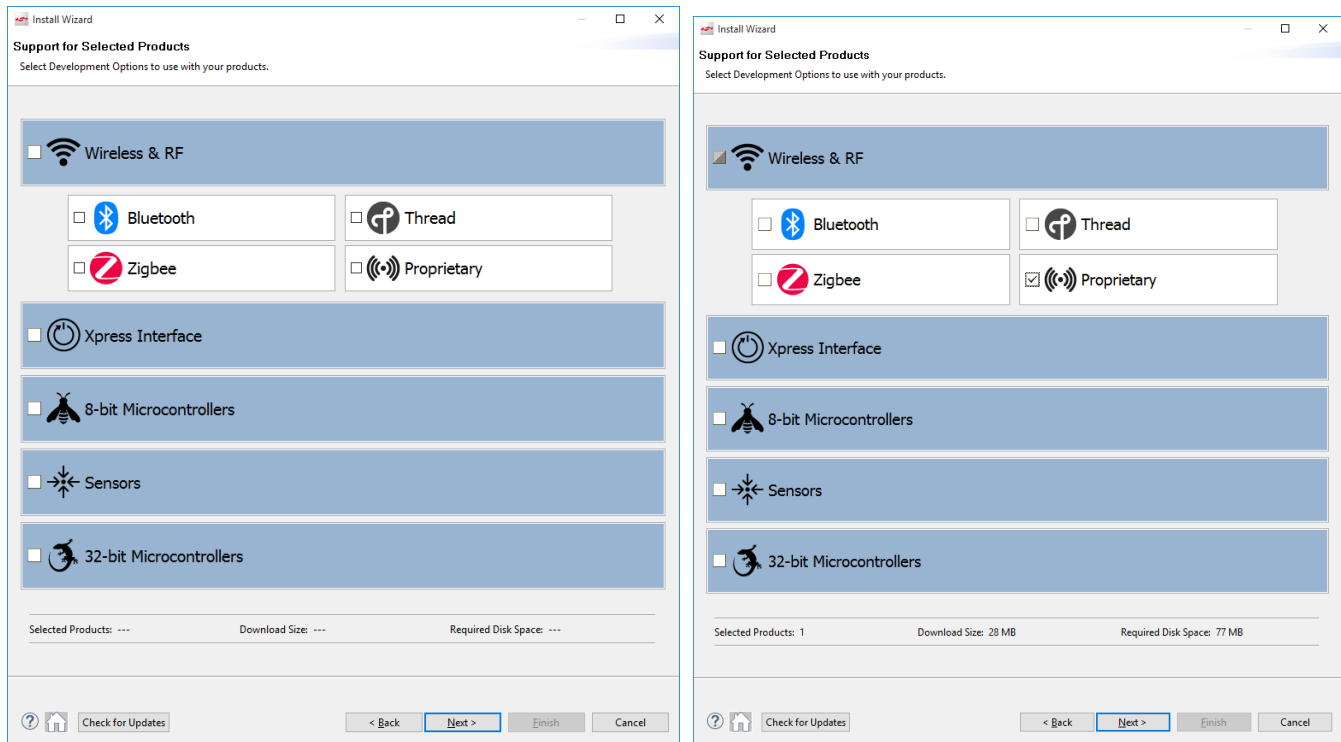


- After some additional items are installed, you are offered the option of installing by device (steps 8 and 9) or installing by product group (step 7).

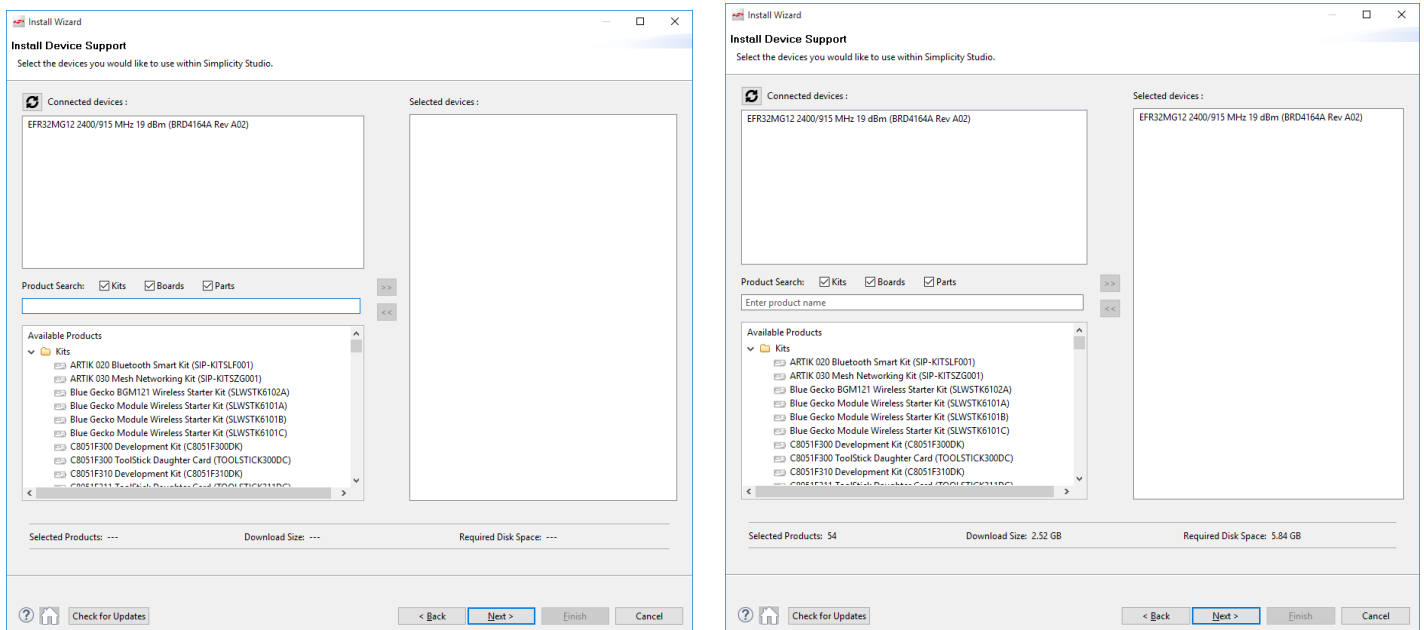


Throughout these procedures at any time you can click Home (🏠) to return to this dialog.

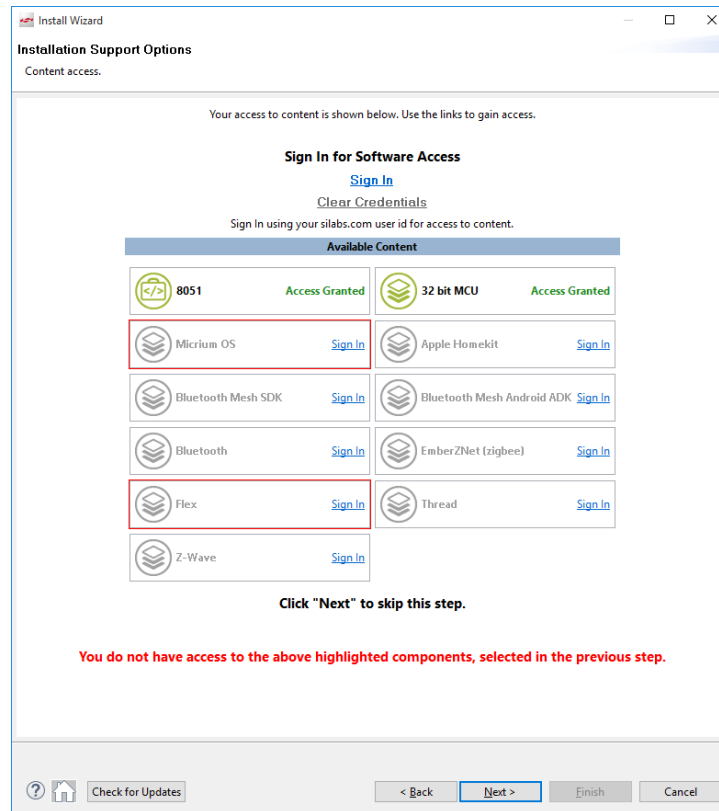
7. If you click **Install by Product Group** you are offered a list of product groups. Click the SDKs you want to install (Proprietary installs the Flex SDK), or click Wireless & RF to check all. Click **Next** and go to step 9.



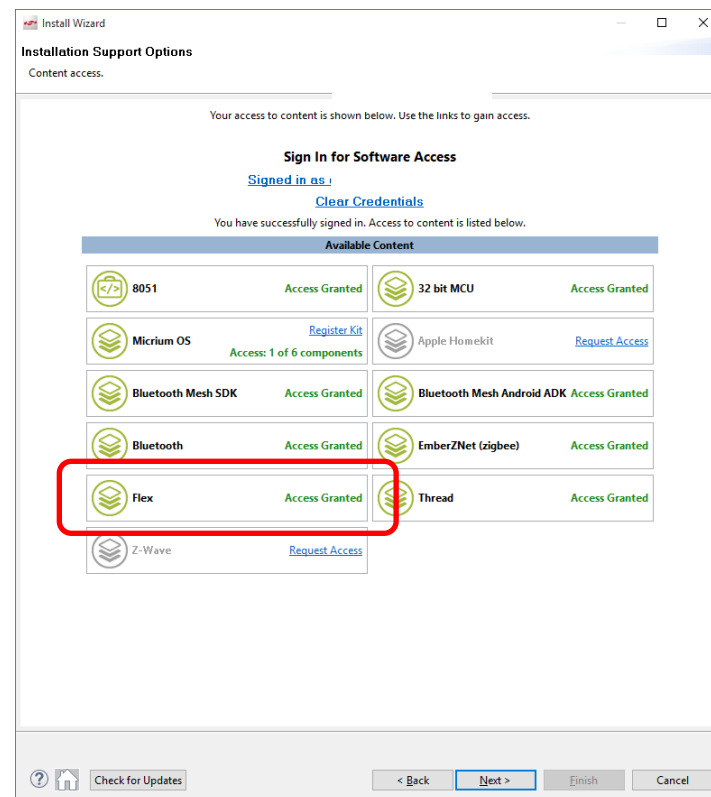
8. If you click **Install by Device**, an Install Device Support dialog appears. After a short delay, it shows your connected device. If the connected device does not show, click **Refresh**. Select either a connected device, or search for a product and select it. When a product is selected click **>>** to add it to the Selected Device pane. Simplicity Studio calculates available space required for installation. You can also click a selected device and click **<<** to remove it. Click **Next** to continue.



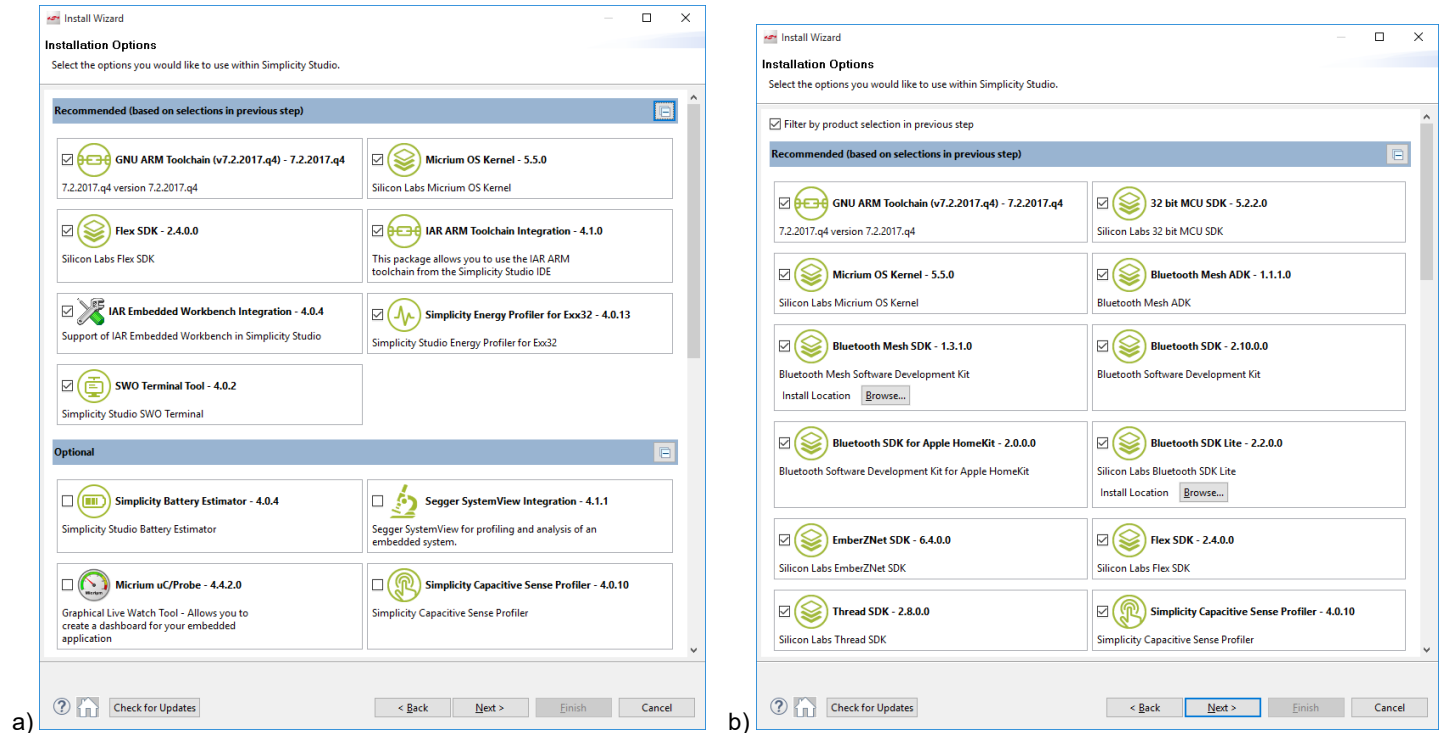
9. The next dialog varies depending on whether you have signed in. If you have not signed in, you have no access to restricted content and must sign in first.



When you have access to the Flex SDK, click **Next**.

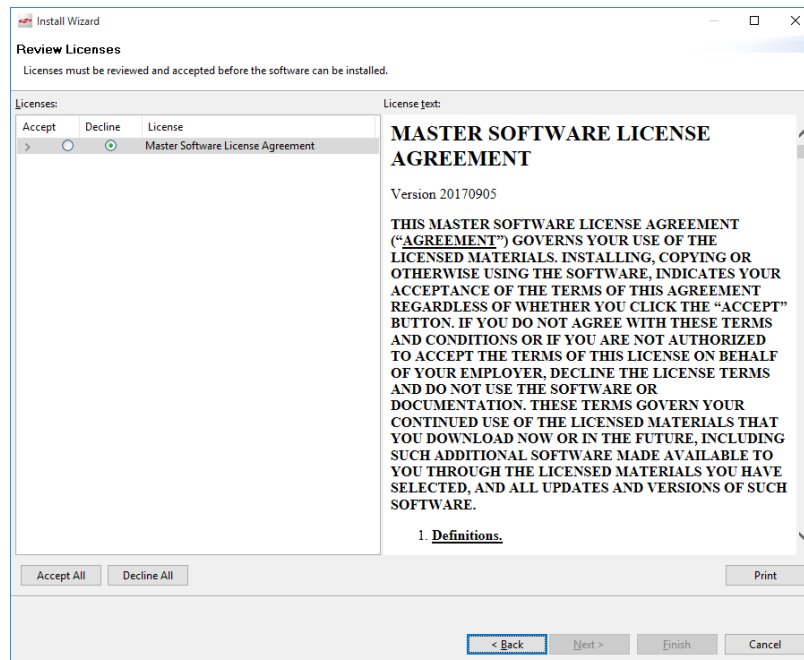


10. The **Installation Options** dialog shows the tools and software packages that can be installed (your versions may be different). The following shows Installation options after selecting the Proprietary product group (a), and after selecting an EFR32MG device (b). In both views you can uncheck anything you don't want to install. If you have installed by Product Group, the selection is filtered more specifically to your needs than if you have installed by device, and installing all checked options is recommended. If you have installed by device and are unchecking items, note that if you plan to use GCC as your compiler, be sure to leave the GNU ARM Toolkit checked. You can also install it later through the Upgrade Manager interface, Tools tab. Click **Next >**.

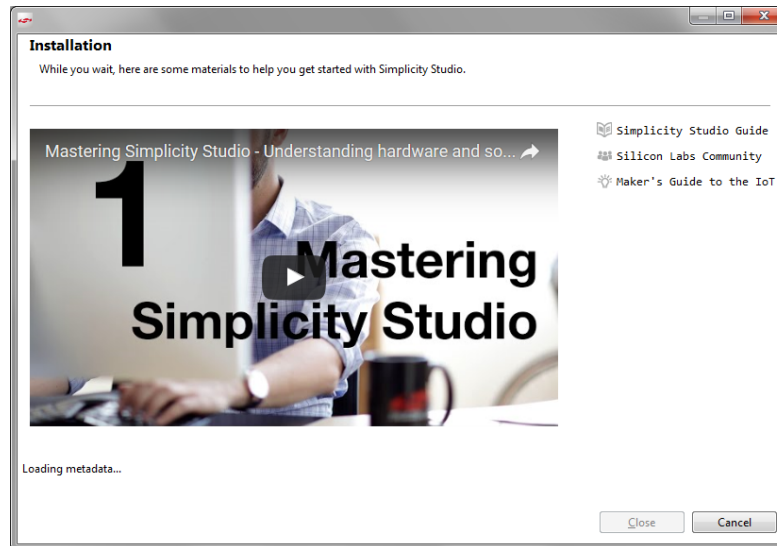


**Note:** Previous stack versions are shown under **Other Options**.

11. Studio displays a Review Licenses dialog. Accept the licenses shown and click **Finish**. Note that this dialog will present again if in the future you install a component with a separate license.

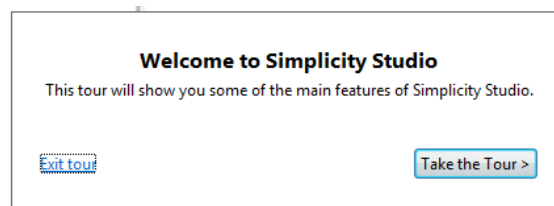


Installation takes several minutes. During installation, Simplicity Studio offers you viewing and reading options to learn more about the environment.

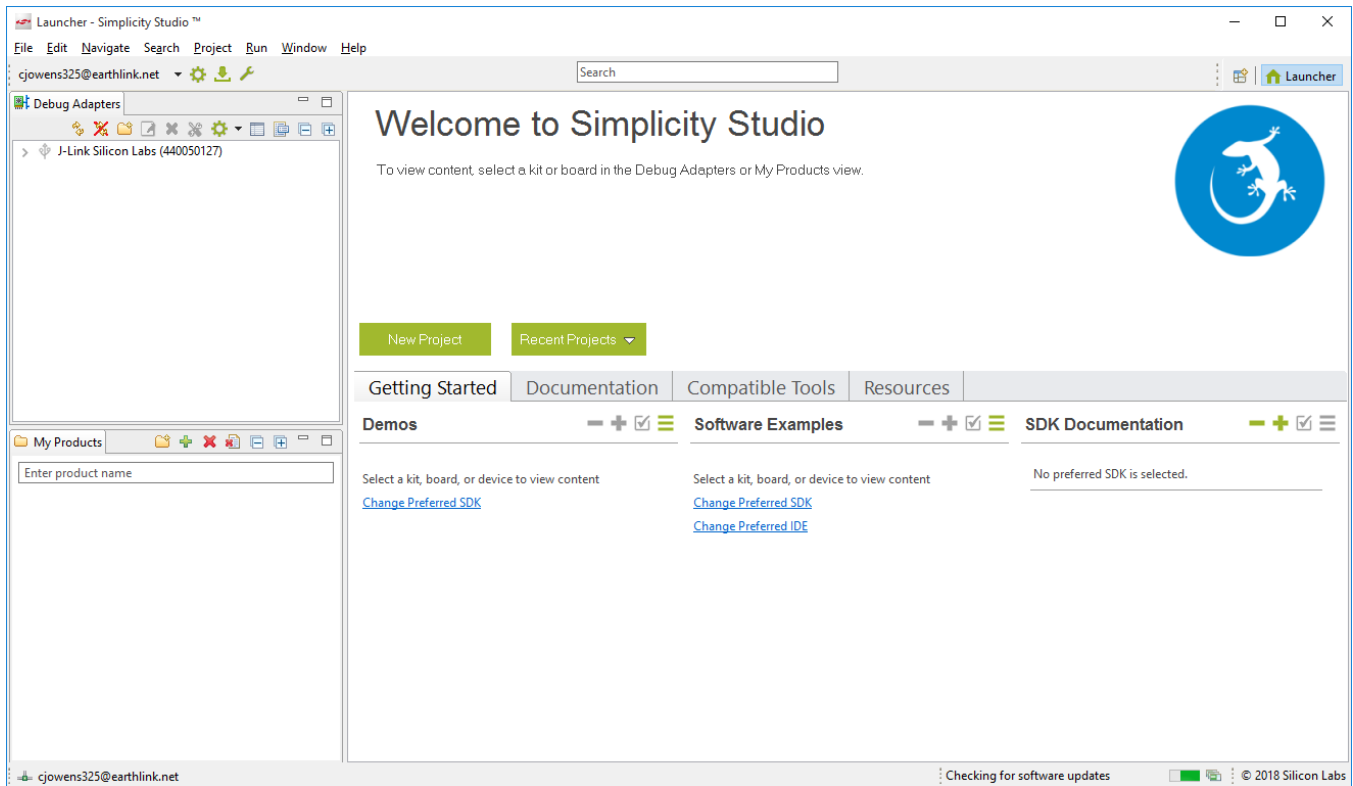


12. After installation is complete, restart Simplicity Studio.

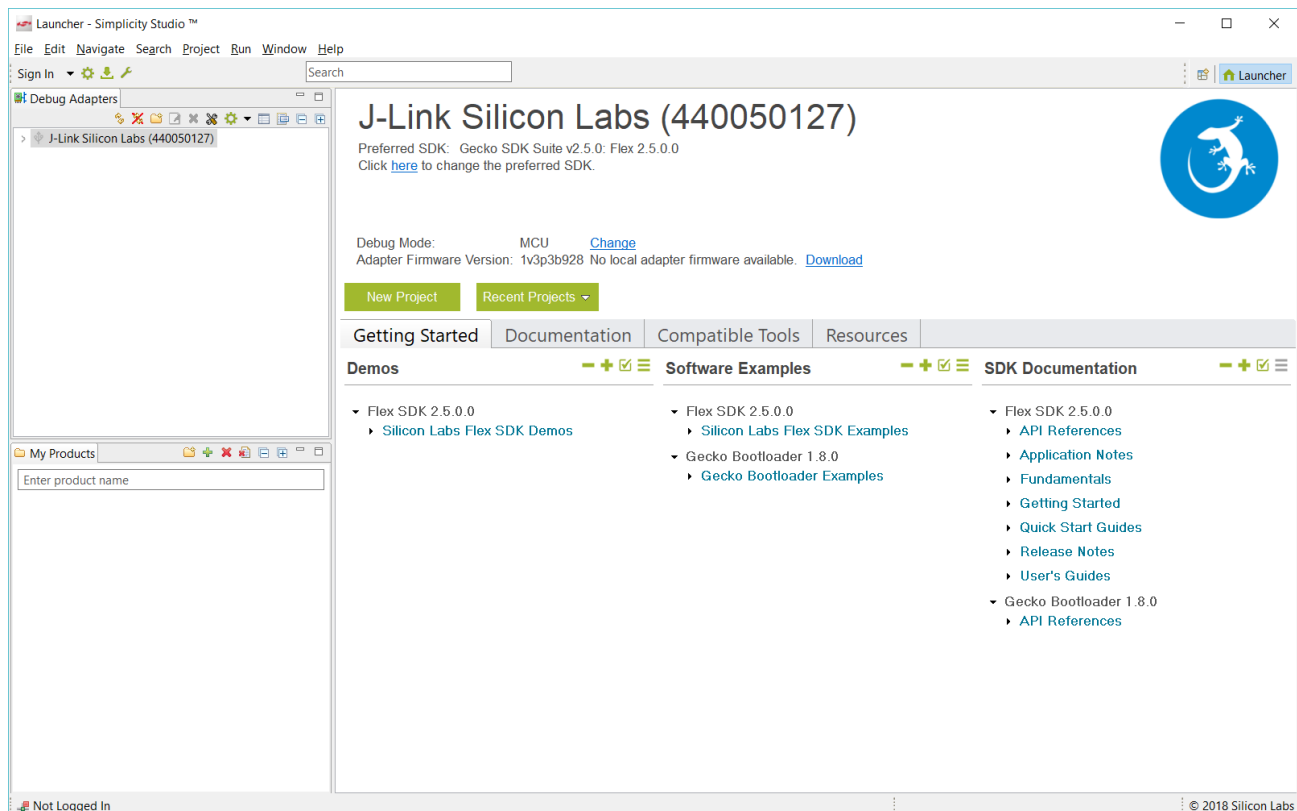
13. When Simplicity Studio restarts, you are invited to take a tour. To clear this option now or at any time during or after the tour, click **Exit tour**.



14. The Launcher perspective opens, but it is not yet fully populated. Click one of the devices in the Devices tab or find and select a part in the Solutions tab. Note that USB-connected WSTK devices are identified as J-Link devices as shown.



15. The Launcher perspective then is populated with the software components and functionality associated with your hardware and stack. Update your device firmware as described in section 3.3 Updating Adapter Firmware.



Finally, if you plan to use IAR as your compiler, find the Release Notes on the SDK Documentation list and check for software version requirements, in particular for IAR-EWARM. To install IAR-EWARM:

1. On the Launcher page's Resources tab, click **Technical Support**.
2. Scroll down to the bottom of the page, and click **Contact Support**.
3. If you are not already signed in, sign in.
4. Click the Software Releases tab, and In the View list select **\_Latest EmberZNet Software**. In the results is a link to the appropriate IAR-EWARM version.
5. Download the IAR package (takes approximately 1 hour).
6. Install IAR.
7. In the IAR License Wizard, click **Register with IAR Systems to get an evaluation license**.
8. Complete the registration and IAR will provide a 30-day evaluation license.



### 3 Functionality in the Launcher Perspective

Perspectives are made up of a number of tiles or panes, called views, as well as the content in those views. You can perform a number of functions in the Launcher Perspective, shown in the following figure. Additional information on some of these is provided later in the section. Note: Your installed version may be different than the version shown in the graphics in this section.

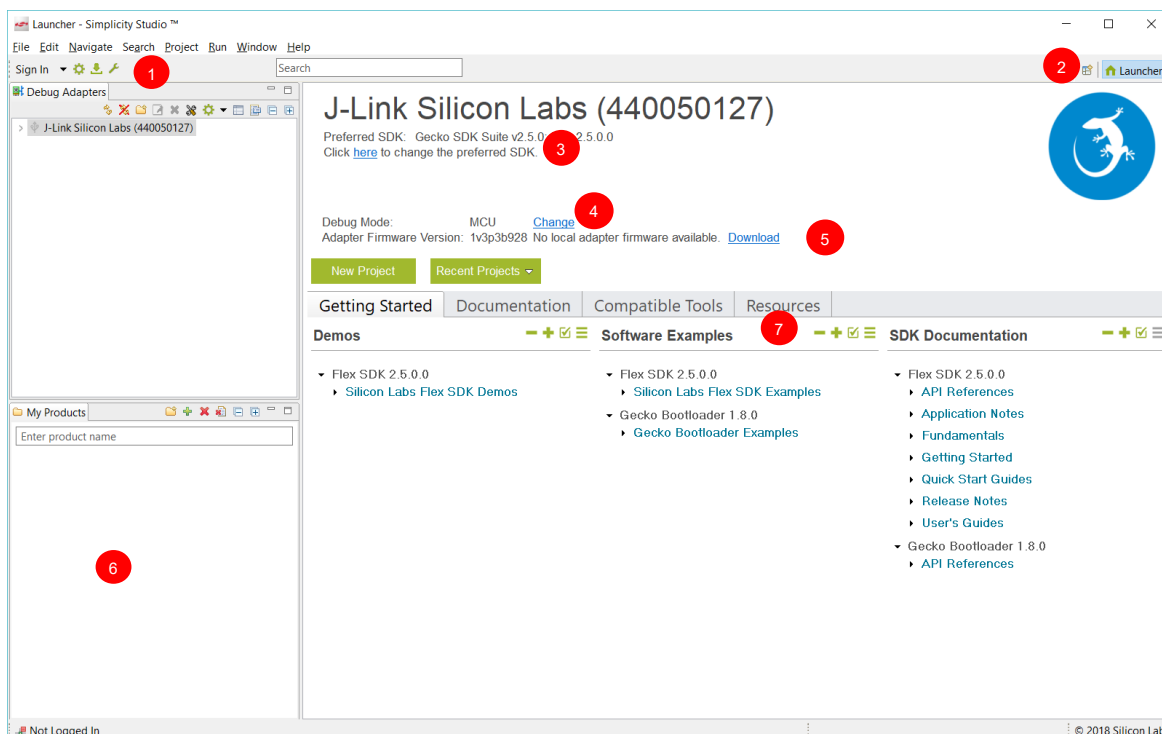
On the toolbar (1) you can:

- Sign in or out
- Open application settings (  )
- Update your software and firmware (  , see section [3.1 Downloading Updates or Installing Additional Components](#) for more information)
- Open the Tools menu (  ) to access tools such as Simplicity Commander or Energy Profiler.
- Search for information on line, including entries in the Community forums.
- Change perspectives (2). As you open the Simplicity IDE or other tools, buttons for their perspectives are displayed in the upper right. Use those buttons to easily navigate back to the Launcher perspective or to other perspectives. You can change the layouts of various perspectives by expanding or relocating views, or adding or removing views. To return to the default layout, right-click the perspective button in the upper right and select **Reset**.

In the main view you can:

- Change your preferred SDK (3, see section [3.2 Changing the Preferred SDK](#) for more information - legacy functionality, rarely needed).
- Change debug mode (4).
- Update adapter firmware (5, see section [3.3 Updating Adapter Firmware](#) for more information).
- Create solutions of multiple parts (6). If you are developing for complex networks with a number of different parts involved, you can add them all to the solution and then select the one you are working on from the list. You do not need to have the hardware connected to your computer.
- Access demos, examples, documentation, and other resources from the Getting Started and other tabs (7).

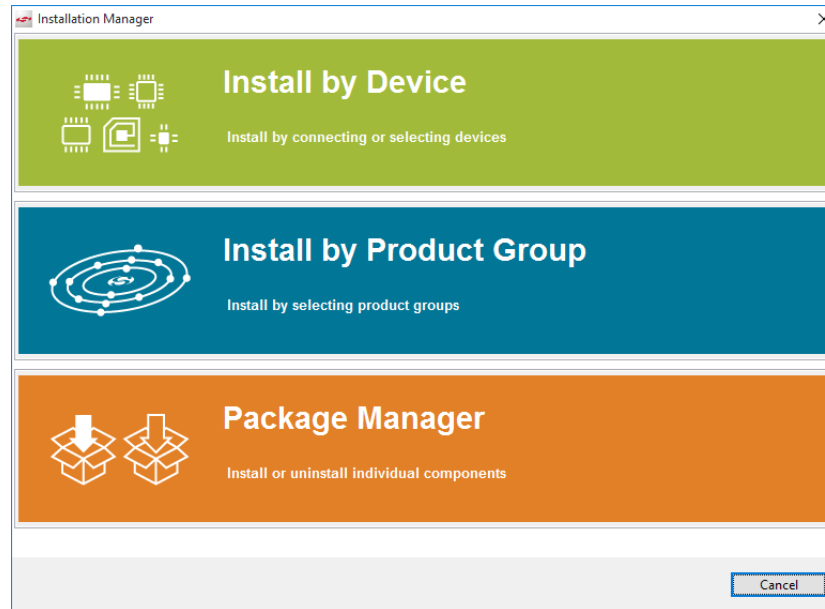
— + ☑ ☰ Use these controls to manage groups of items (Collapse All, Expand All, Customize, and Show All, respectively). See section [3.4 Accessing Documentation and Other Resources](#) for more information.



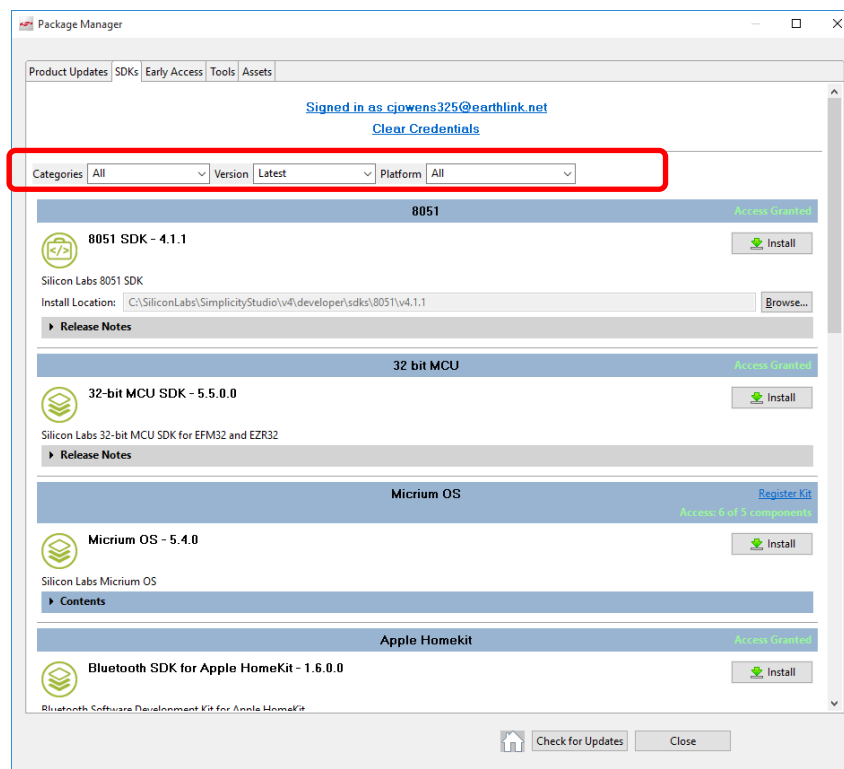
### 3.1 Downloading Updates or Installing Additional Components

The Update Software icon will be red if updates to installed components are available. If Simplicity Studio detects an available update, and you are in another perspective, you will be notified that an update is available.

To download a new or updated component, click the Update Software icon. Click Package Manager. Note: If you are installing based on a new device, or want to install a new product group, you can do so through this dialog as well. In subsequent dialogs, click Home (🏠) to return to this dialog. Note that Studio does not show you options, such as the GNU ARM Toolchain, that have already been installed.

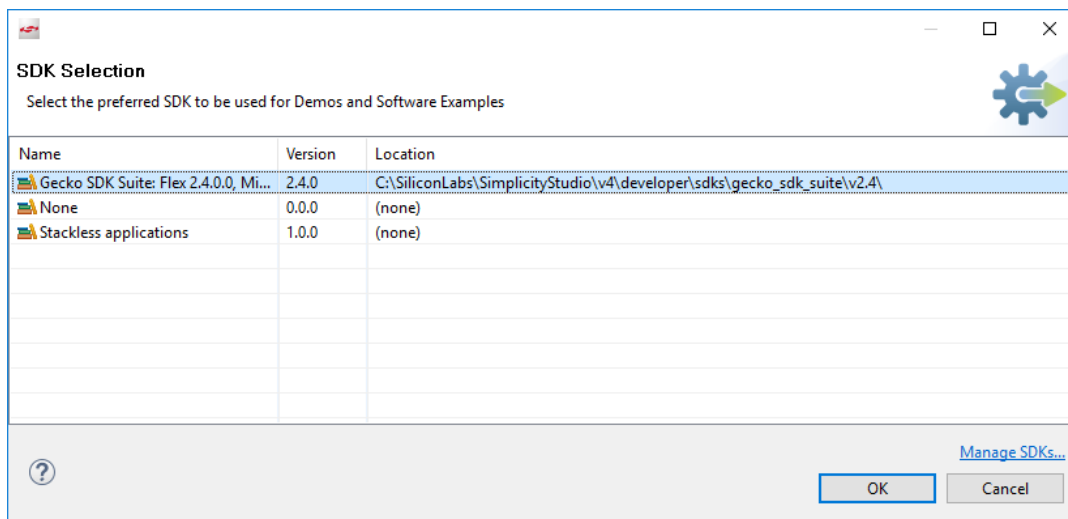


Simplicity Studio shows you available updates or SDKs in the Package Manager dialog. You can update all or select individual updates for installation. Click the tabs in the Package Manager dialog to see other components available for installation. Use the filters to reduce long lists.



### 3.2 Changing the Preferred SDK

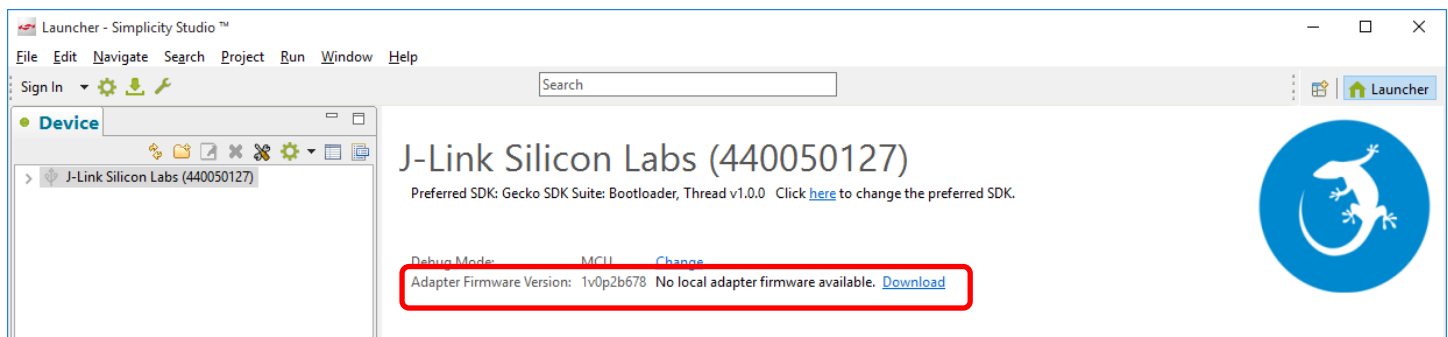
Use this function if on startup Simplicity Studio defaults to Stackless applications. Otherwise, most Silicon Labs protocol stack users will have one SDK available to them, the Gecko SDK Suite.



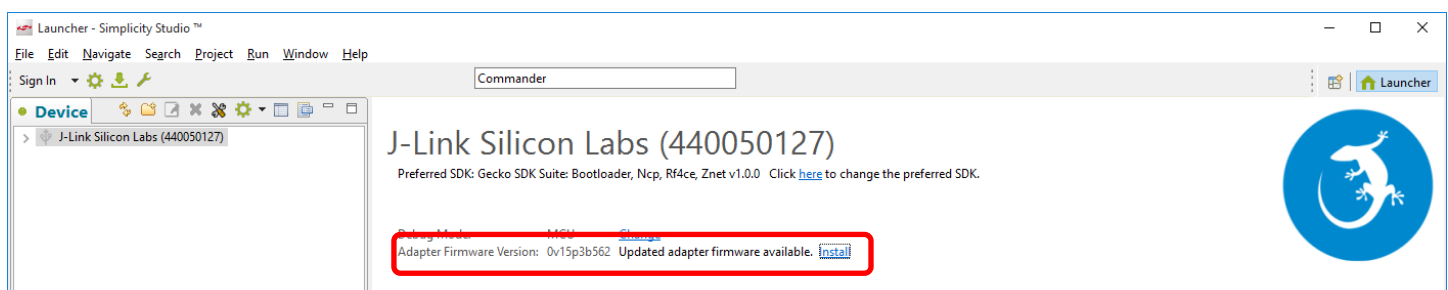
Within that suite you can have multiple protocols installed. The protocol used in any given instance is controlled either by the example you select, or the stack you select if you go through the 'New Project' interface. In general, you should add or remove protocol stacks through the Simplicity Studio update manager. If you need to install a stack or the Gecko SDK Suite outside of the normal installation process, you will receive separate instructions.

### 3.3 Updating Adapter Firmware

Initially the Launcher perspective may display "No local adapter firmware available." Click **Download** to download any updates.

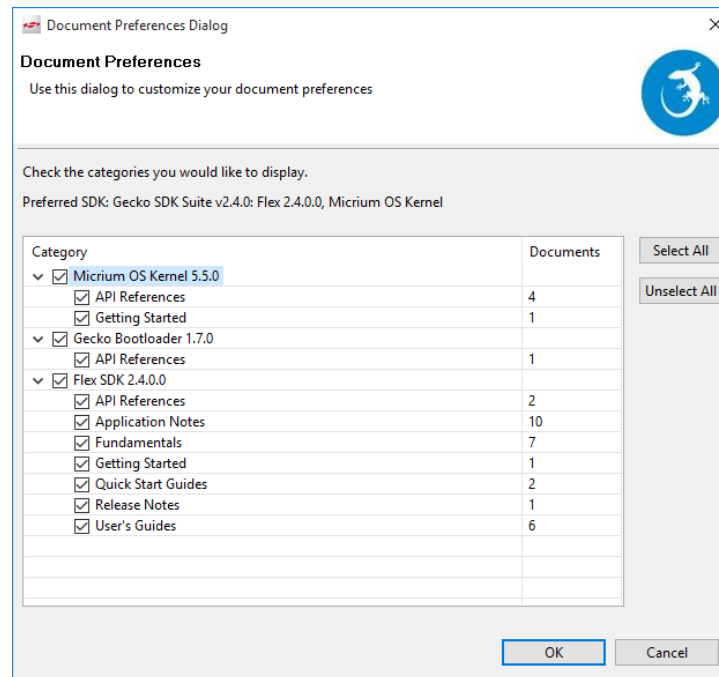


If an update is available, click **Install** to install the firmware.

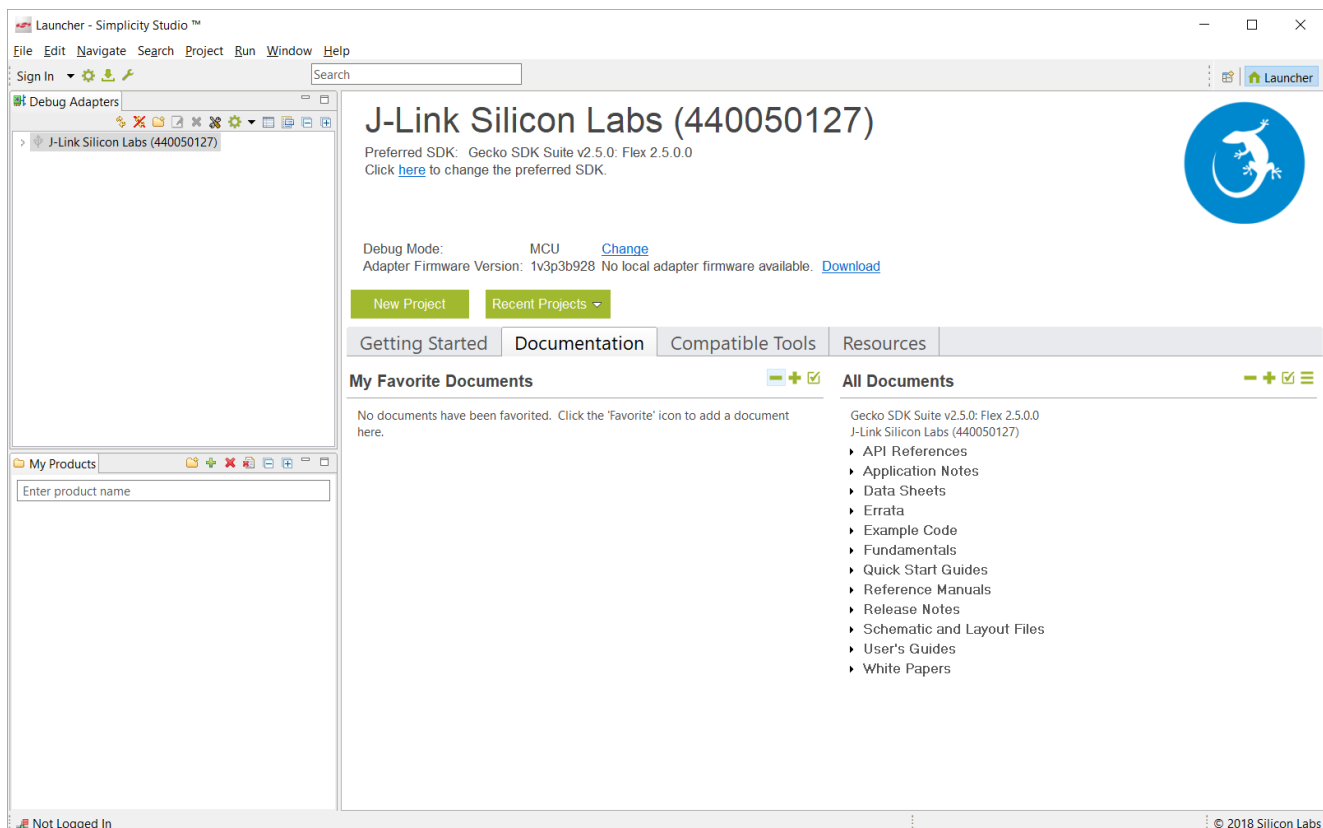


### 3.4 Accessing Documentation and Other Resources

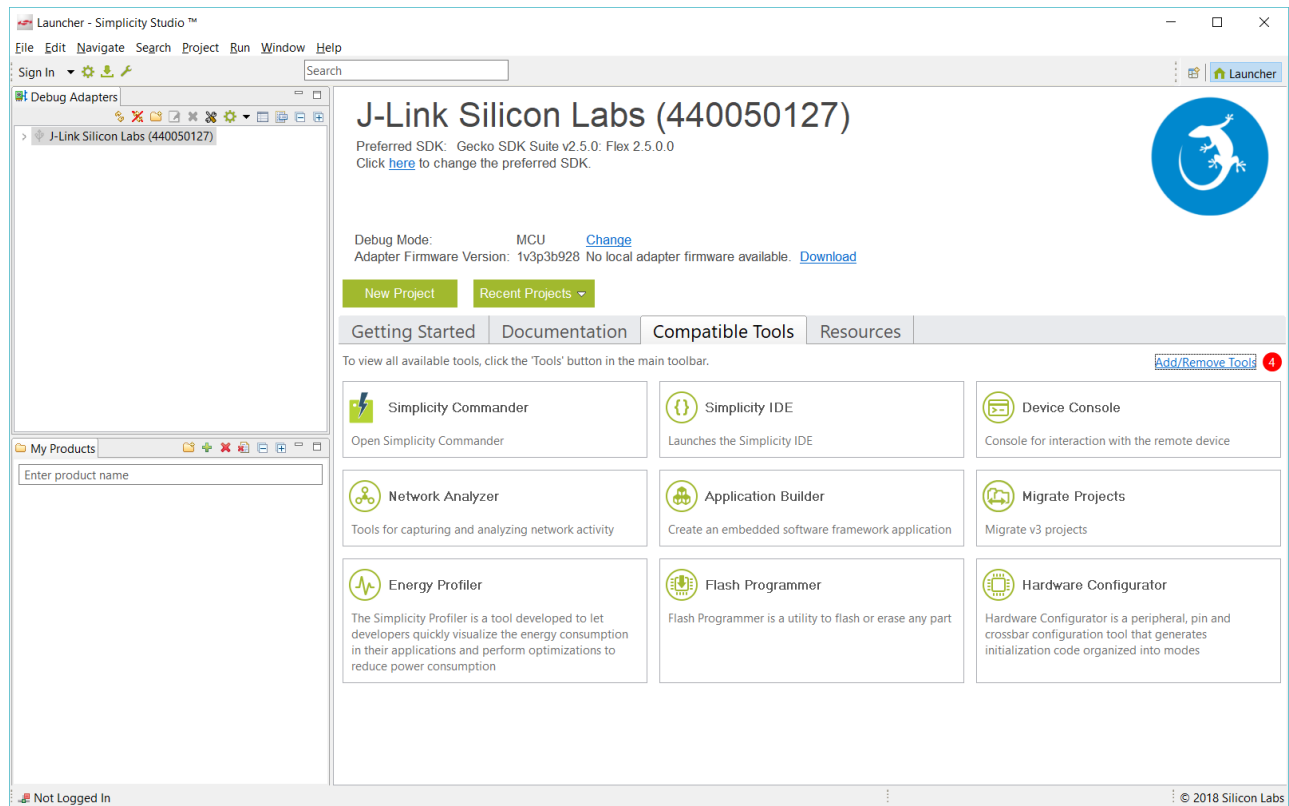
The **Getting Started** tab provides access to demos, example applications, and stack-related documentation. To show/hide specific categories, click . Select or deselect categories, then click **OK**.



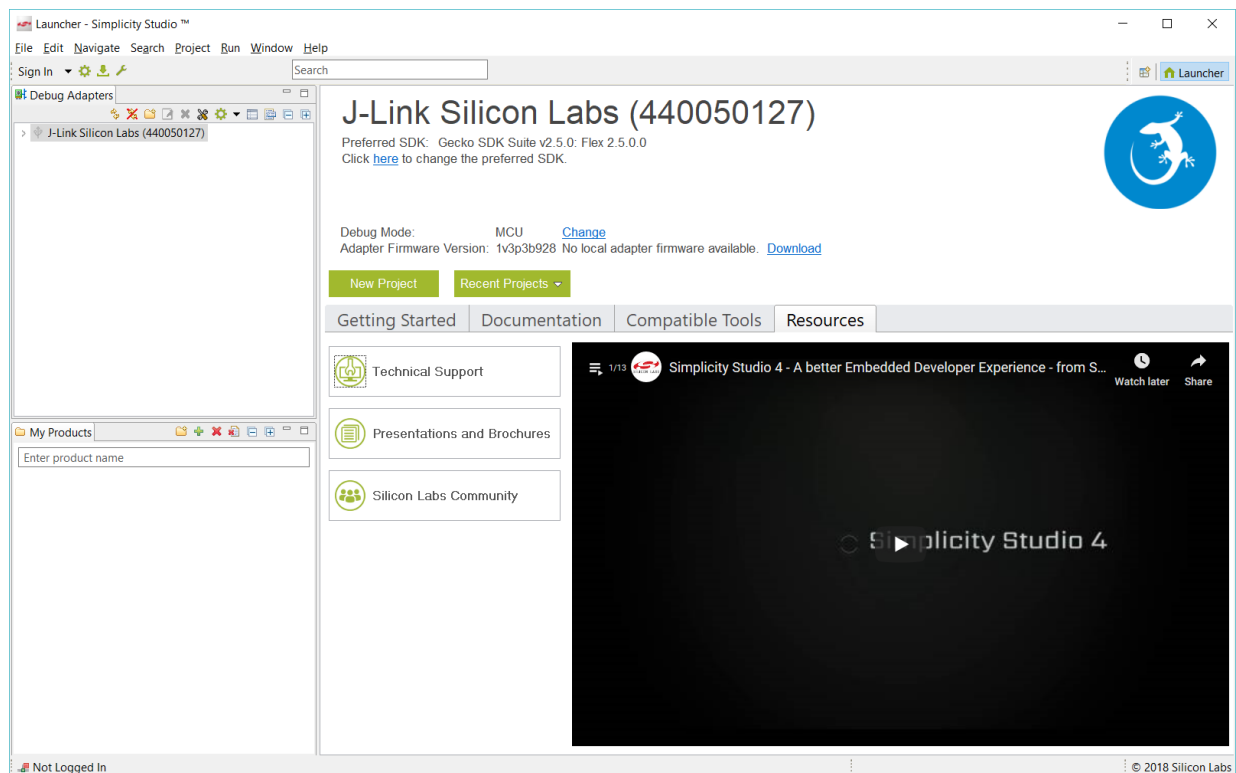
The **Documentation** tab combines documentation about the stack and the hardware on the right, and documents you select as favorites on the left. Click the star icon on any document to move it to the My Favorite Documents list.



The **Compatible Tools** tab is an alternative way to access the tools available through the Tools dropdown.



The **Resources** tab provides access to support, marketing collateral, and the Silicon Labs community.



## 4 Working with Example Applications

This chapter offers instructions on working with several types of examples.

- [Working with Connect SoC Examples](#)
- [Working with Connect Host/NCP Examples](#)
- [Working with RAIL Examples](#)
- [Starting with a Blank Application](#)

In the SoC (System on Chip) model the entire system (stack and application) resides on a single wireless device, whereas in the Host/NCP (Network Co-Processor) model the stack processing is done on the device and the application runs on a separate host microcontroller.

Simplicity Studio offers a variety of ways to begin a project using an example application. The easiest when you are getting started is through the Launcher page. Section [4.1 Working with Connect SoC Examples](#) and [4.3 Working with RAIL Examples](#) both use this method. As you get into development you may want to begin projects without leaving the Simplicity IDE (also known as AppBuilder). The instructions for section [4.2 Working with Connect Host/NCP Examples](#) starts new projects from the File menu.

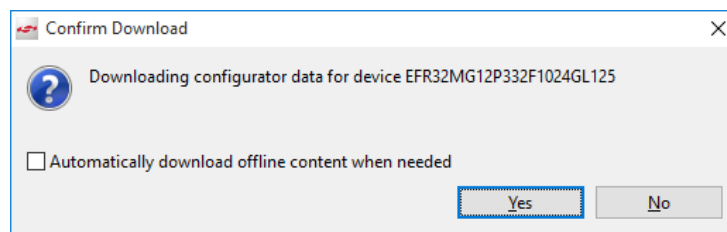
Regardless of how you begin, when working with example applications in Simplicity Studio, you will execute the following steps:

1. Select an example application.
2. Generate application files.
3. Compile and flash the application to the radio board.

These steps are described in detail in the following sections. **Note:** Your SDK version may be later than the version shown in the procedure illustrations.

Finally, although Silicon Labs strongly recommends against it, you can start with a blank application, as described in section [4.4 Starting with a Blank Application](#).

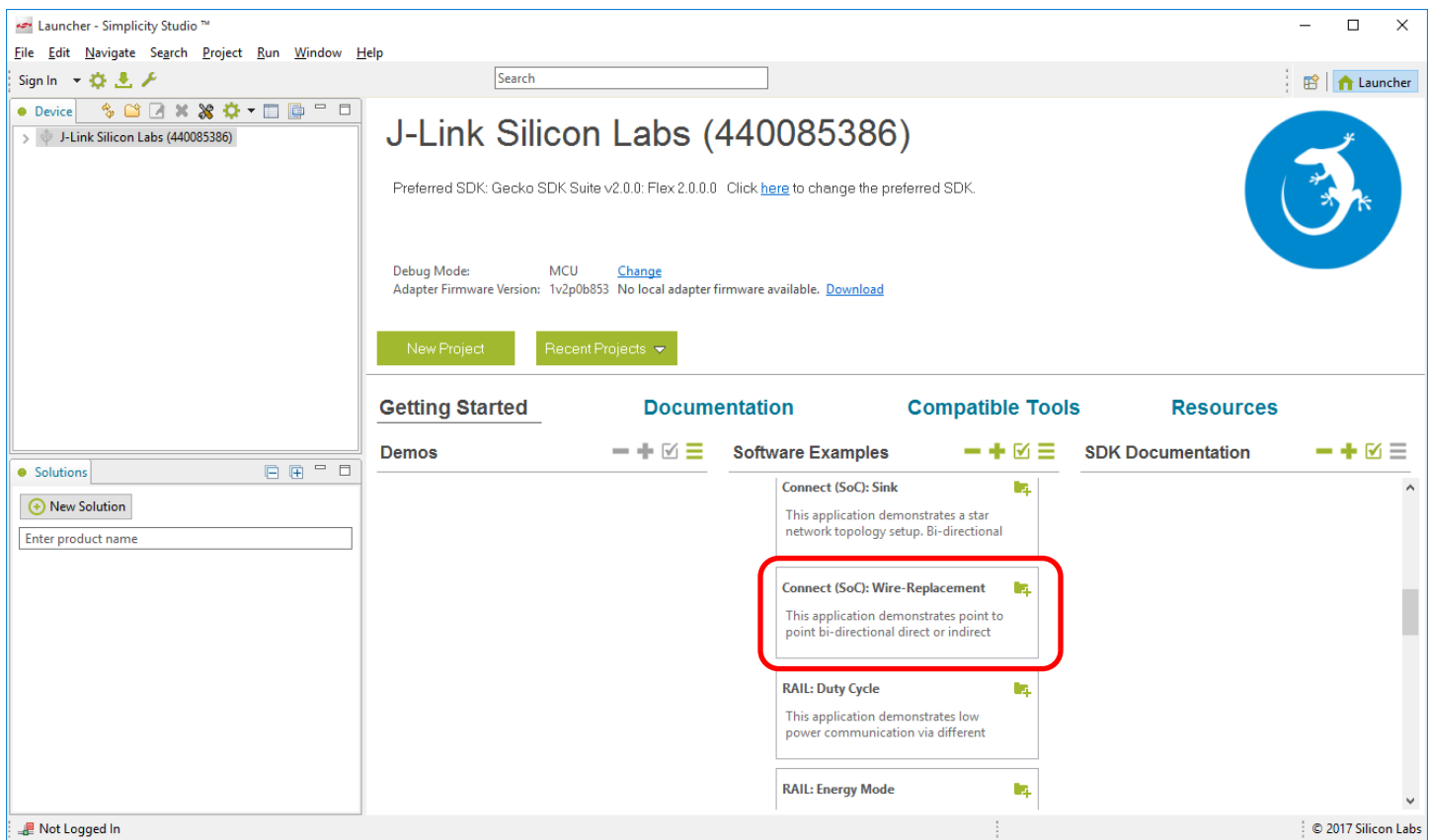
Note: SDK version 2.0.0 and later contain a number of changes to the way hardware peripherals are configured and managed. You may see the following dialog when you open an example or generate code. Always click **Yes**.



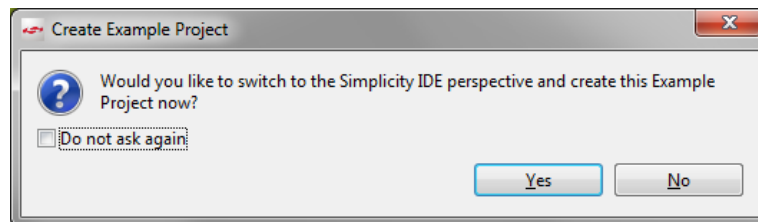
## 4.1 Working with Connect SoC Examples

### 4.1.1 Selecting a Connect Example Application

1. In the Launcher perspective, expand the Silicon Labs Flex SDK list and click on an example application, in this case the **Connect (SoC): Wire Replacement** example.



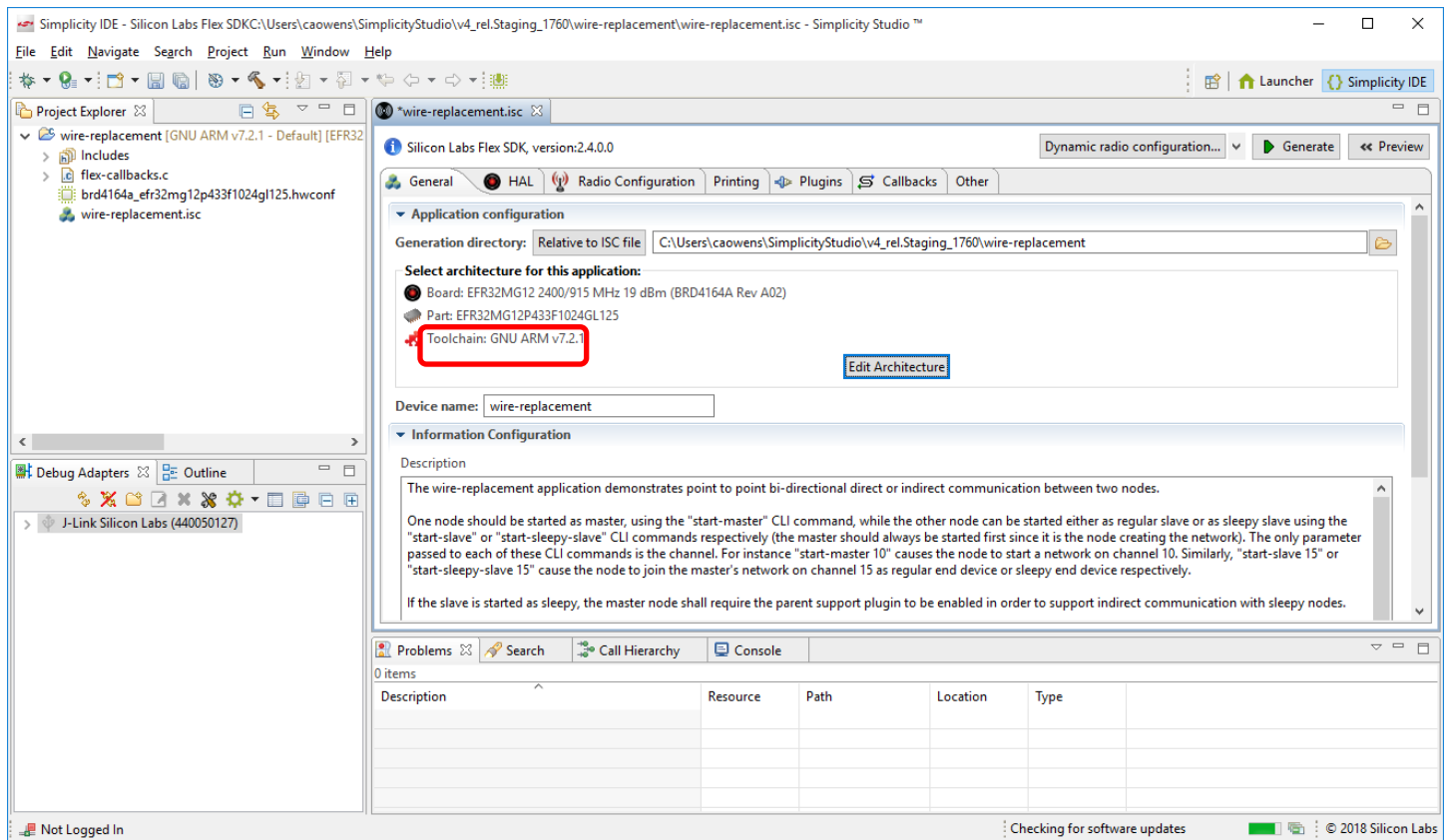
2. You are asked if you want to switch to the Simplicity IDE. Click **Yes**.



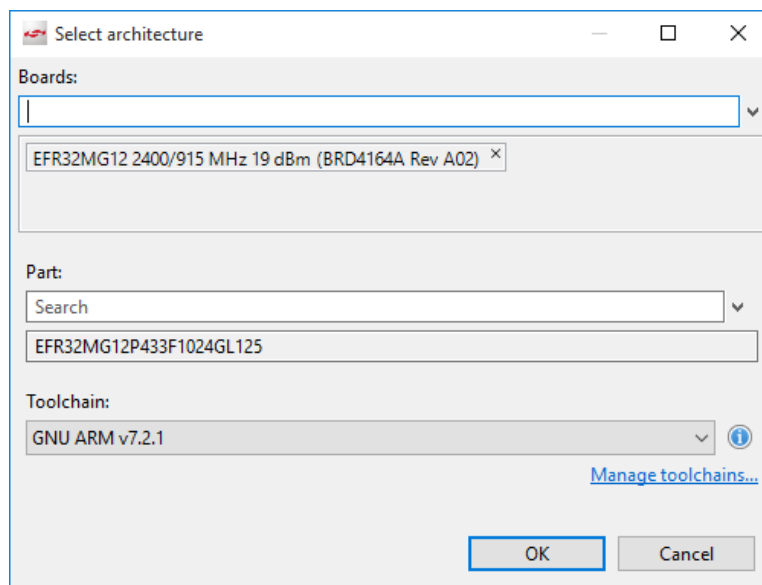
3. Simplicity IDE opens with the new project in AppBuilder view and the focus on the General tab.

**Note:** You now have a Simplicity IDE button next to the Launcher button in the upper right.

Make sure the toolchain shown is the one you want to use. If you have both toolchains installed and enabled in Preferences, GCC is the default.

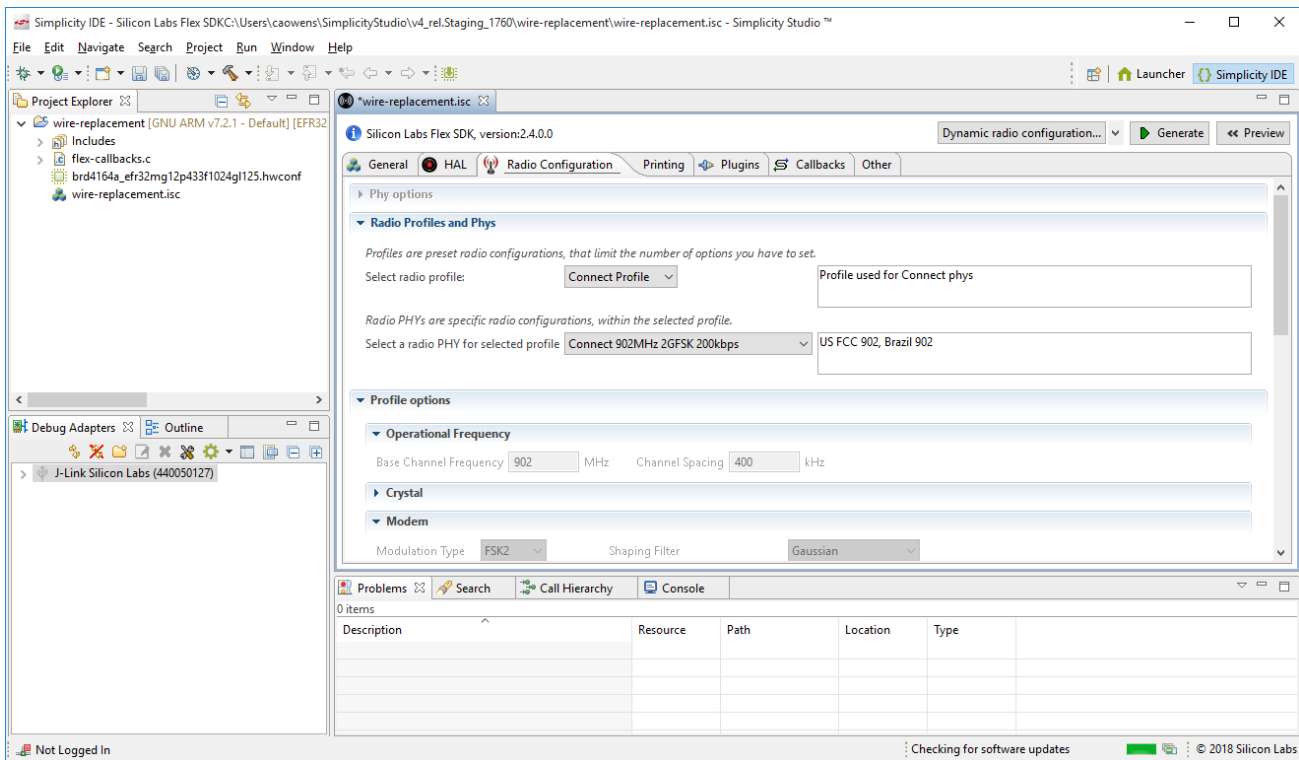


4. To change the toolchain click **Edit Architecture**. In the resulting dialog, select the desired toolchain and click **OK**.

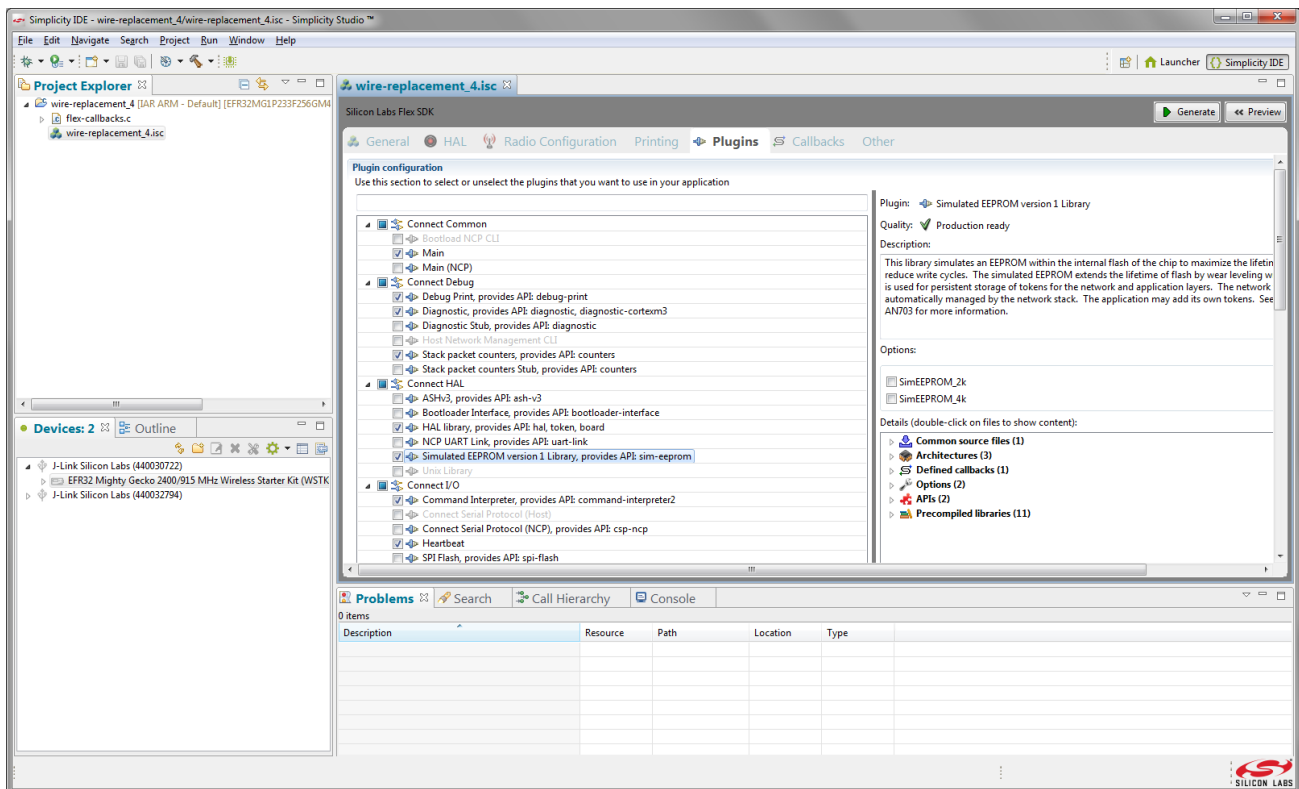




5. Check settings in the Radio Configuration tab to make sure that they are correctly configured for the selected board.

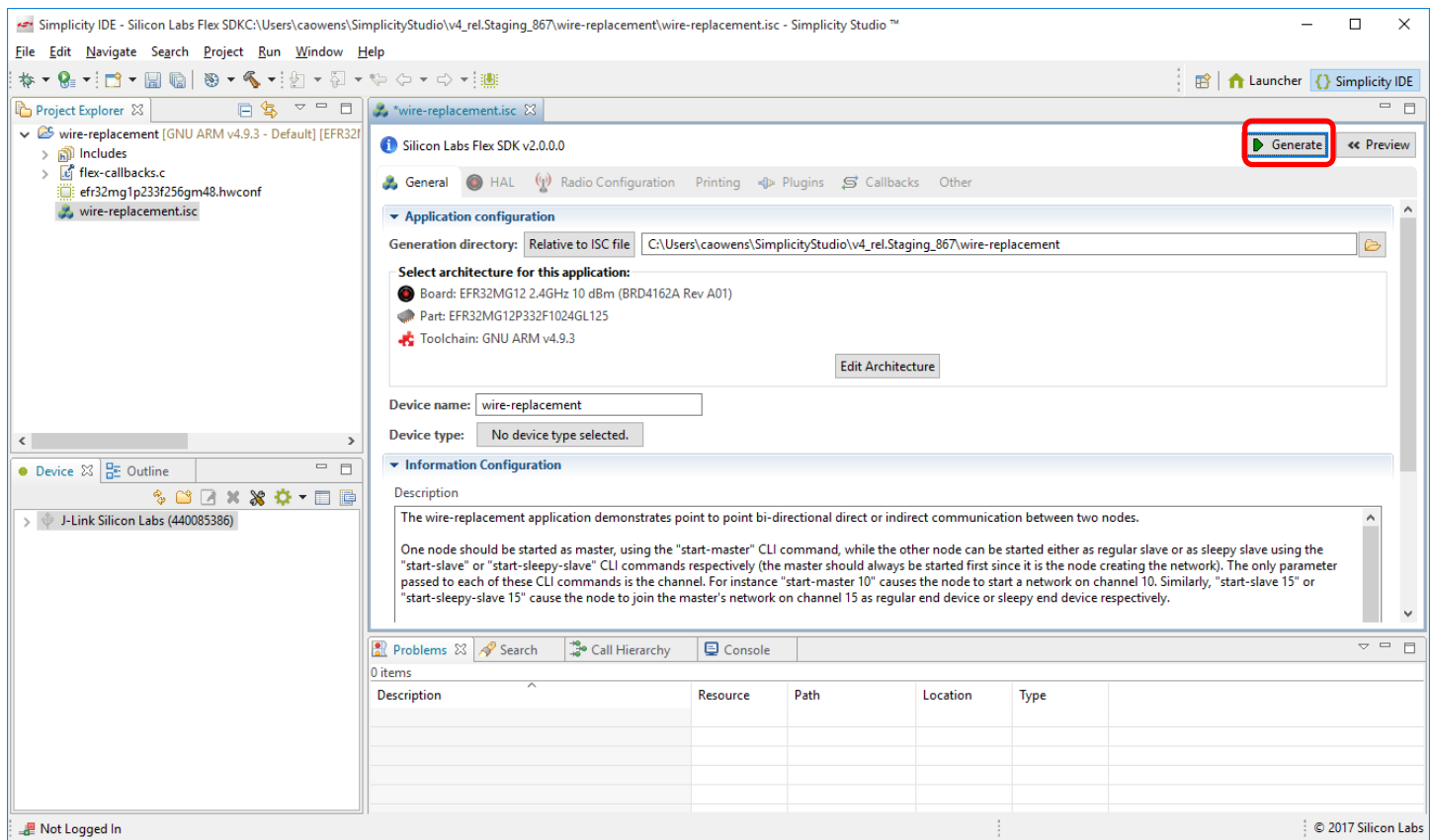


The Plugins tab displays the various plugins that are preselected to enable the example's functionality. Click on a plugin to see its configurable options (if any), along with more information about the plugin.

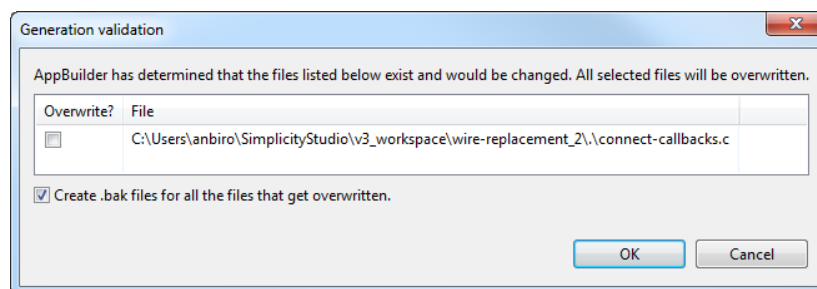


## 4.1.2 Generating Connect Application Files

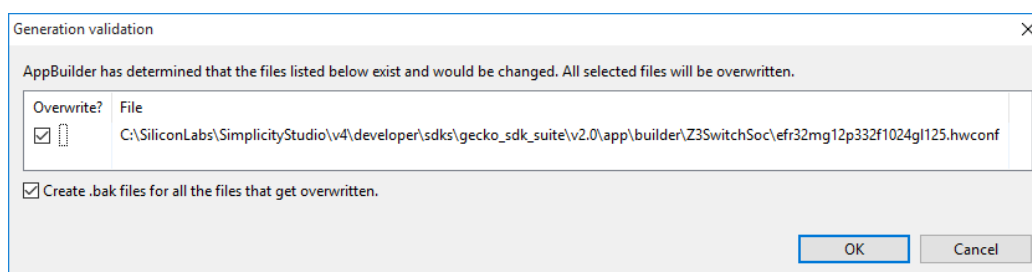
1. In the General tab, the Information Configuration block provides details about application setup and functionality (you may need to scroll down to see it). When you are ready to generate the sample project code, click **Generate**.



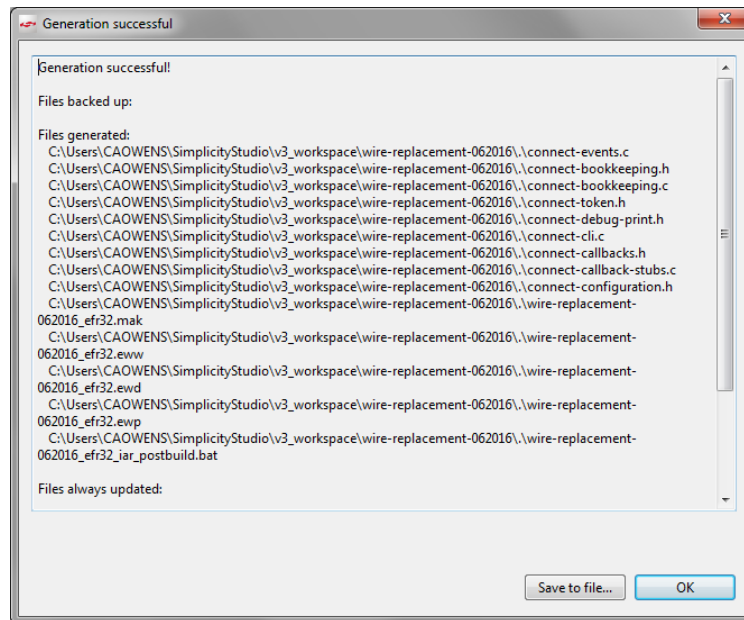
2. When asked about overwriting files, as long as you are working with examples do not check connect-callbacks.c. Only overwrite before starting your own coding, to create a file populated with stub callbacks for those you have indicated. Once you have made modifications to the callbacks in the file, be careful not to overwrite again.



If you get the following overwrite warning, click **OK**.



- Once generation is complete, a dialog reporting results is displayed. Click **OK**.

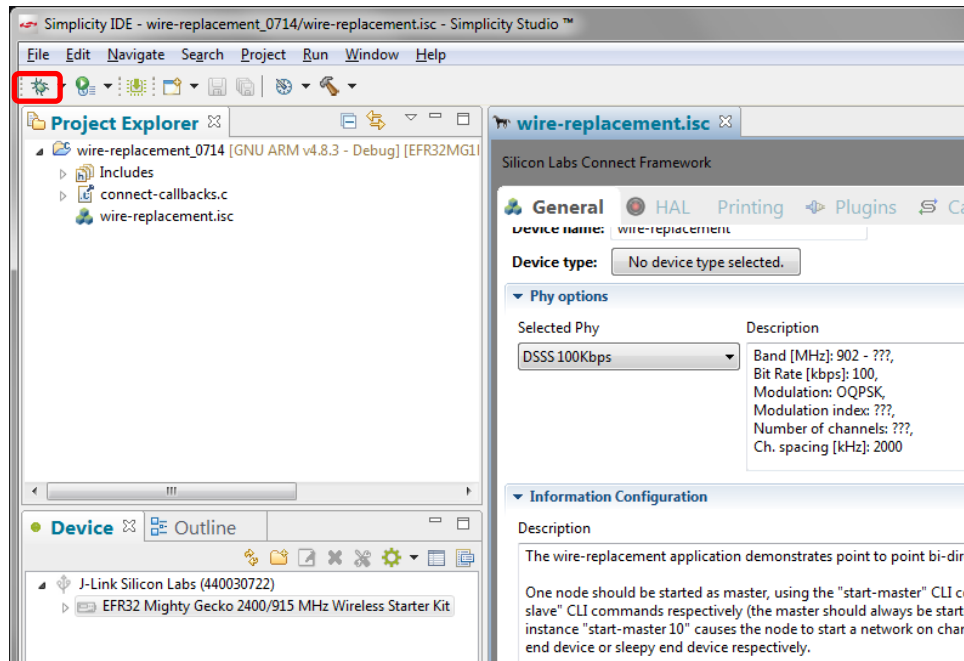


#### 4.1.3 Compiling and Flashing the Connect Application

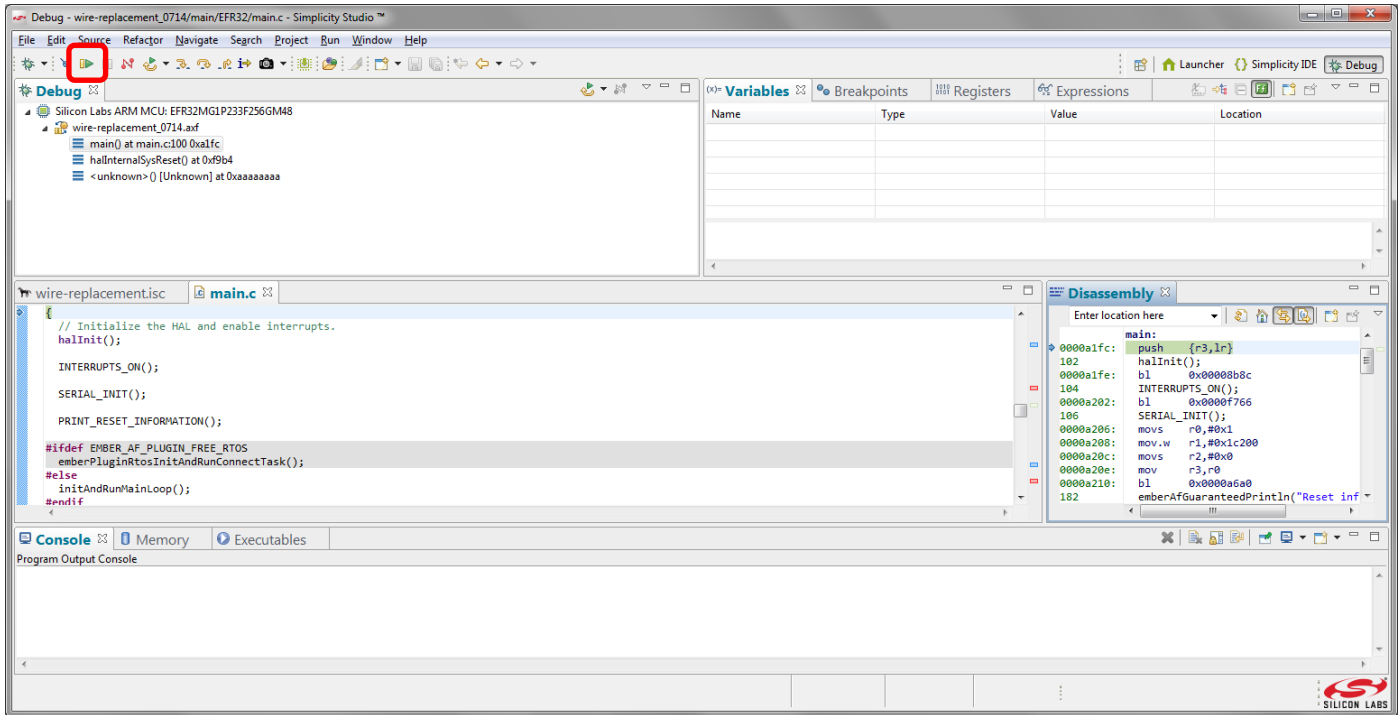
You can either compile and flash the application automatically, or manually compile it and then flash it.

##### Automatically Compile and Flash

- You can automatically compile and flash the application to your connected development hardware in the Simplicity IDE. After you click **OK** on the Generation Confirmation dialog, the Simplicity IDE returns. Click the **Debug** control.



- Progress is displayed in the lower left. Wait until flashing has completed and a Debug perspective is displayed. Click the **Resume** control to start the application running on the WSTK.



Next to the Resume control are Suspend, Disconnect, Reconnect, and stepping controls. Click the **Disconnect** control when you are ready to exit Debug mode.



## Manually Compile and Flash

After you generate your project files, instead of clicking Debug in the Simplicity IDE, click the **Build** control  in the top tool bar.

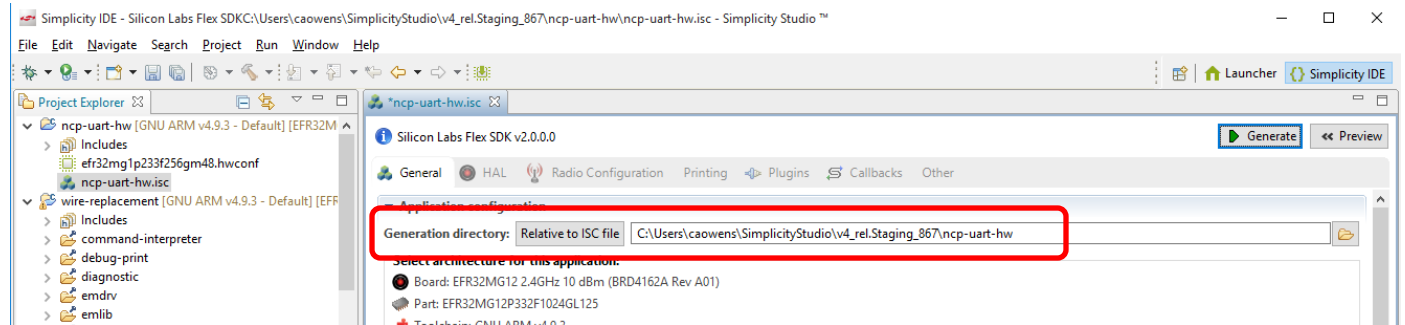
Your sample application will compile based on its build configuration. You can change the build configuration at any time by right clicking on the project and going to **Build Configurations > Set Active**.

You can also build your application directly in IAR-EWARM by opening IAR-EWARM and opening the generated project file inside IAR.

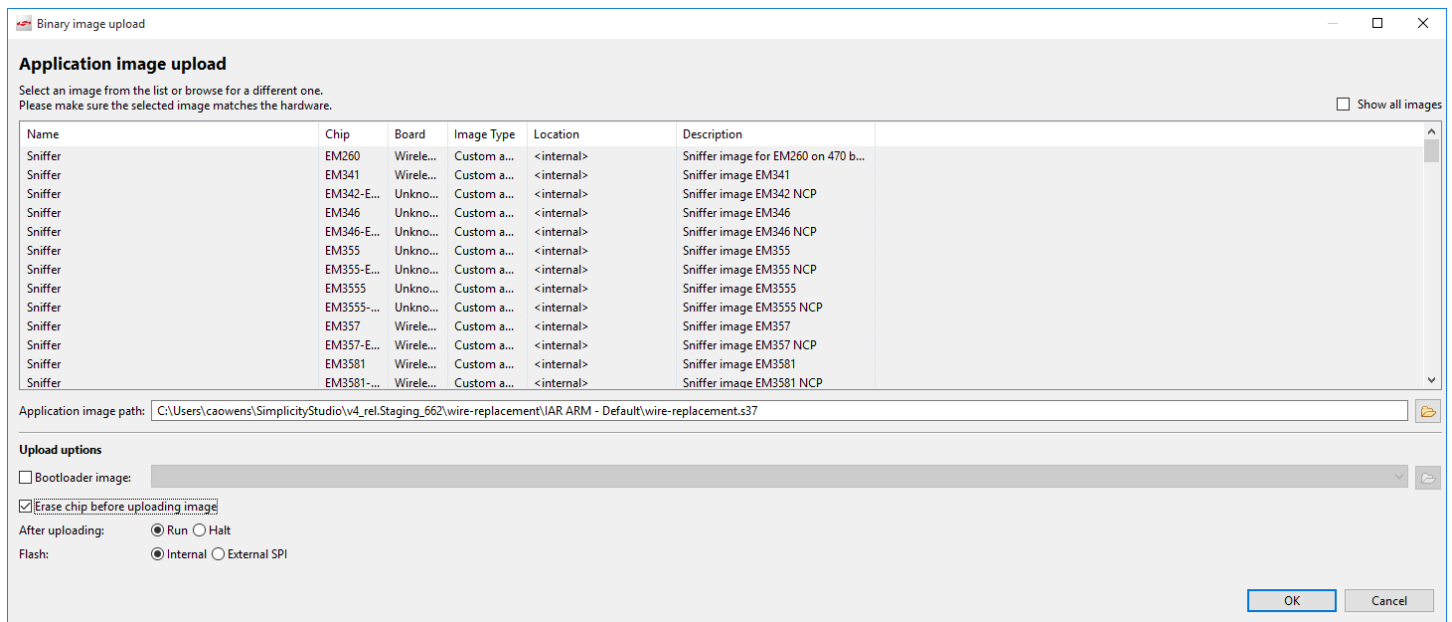
- Open IAR-EWARM.
- Select File > Open > Workspace and navigate to the location you selected for your sample application.
- Select the application .eww file and click **[Open]**.
- Select Project > Make or press F7. If the application builds without errors, you are ready to install the image on a device.

You can load the binary image through the Devices view in the Simplicity IDE perspective.

- Files are generated into the folder on the General tab.



- In the Devices view, right-click on the J-Link device and select **Upload Application**. The Select Binary Image dialog is displayed.

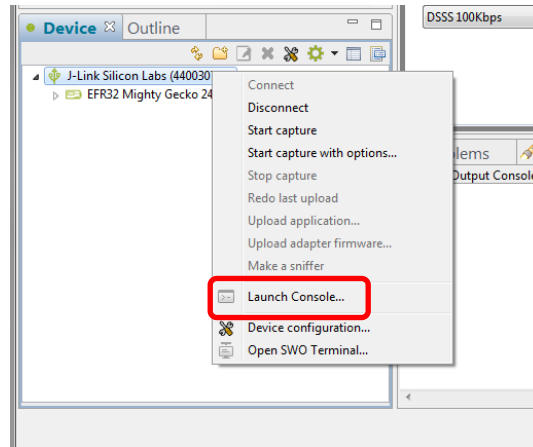


- Navigate to the .s37, .bin or .hex image you wish to upload. Files are located under the project folder on the General tab:  
If you compiled the image with GCC, the files are in <folder on General tab>\GNU ARM vn.n.n - Default  
If you compiled the image with IAR EWARM, the files are in <folder on General tab>\IAR ARM - Default
- Optionally, check **Erase Chip**, to make sure that any previous bootloader or other non-volatile data is erased before your new image is uploaded. New users will typically check this.
- The **After Load** options are **Run** (begin executing the code immediately) and **Reset** (wait for an event, such as a debugger to connect or manual initiation of a boot sequence). During initial development you will typically leave this set to **Run**.
- Click **OK**.

#### 4.1.4 Interacting with Connect Examples

Depending on the example application, you may be able to interact with it through your development environment's Console interface using a CLI (command line interpreter). The console interface allows you to form a network and send data.

To launch the Console interface, in the Simplicity IDE perspective right-click on your J-Link device in the Devices View. Choose **Launch Console**. Alternatively, from the Tools drop-down in the Launcher perspective select **Device Console**. To get a prompt on the Console Serial 1 tab, press Enter.

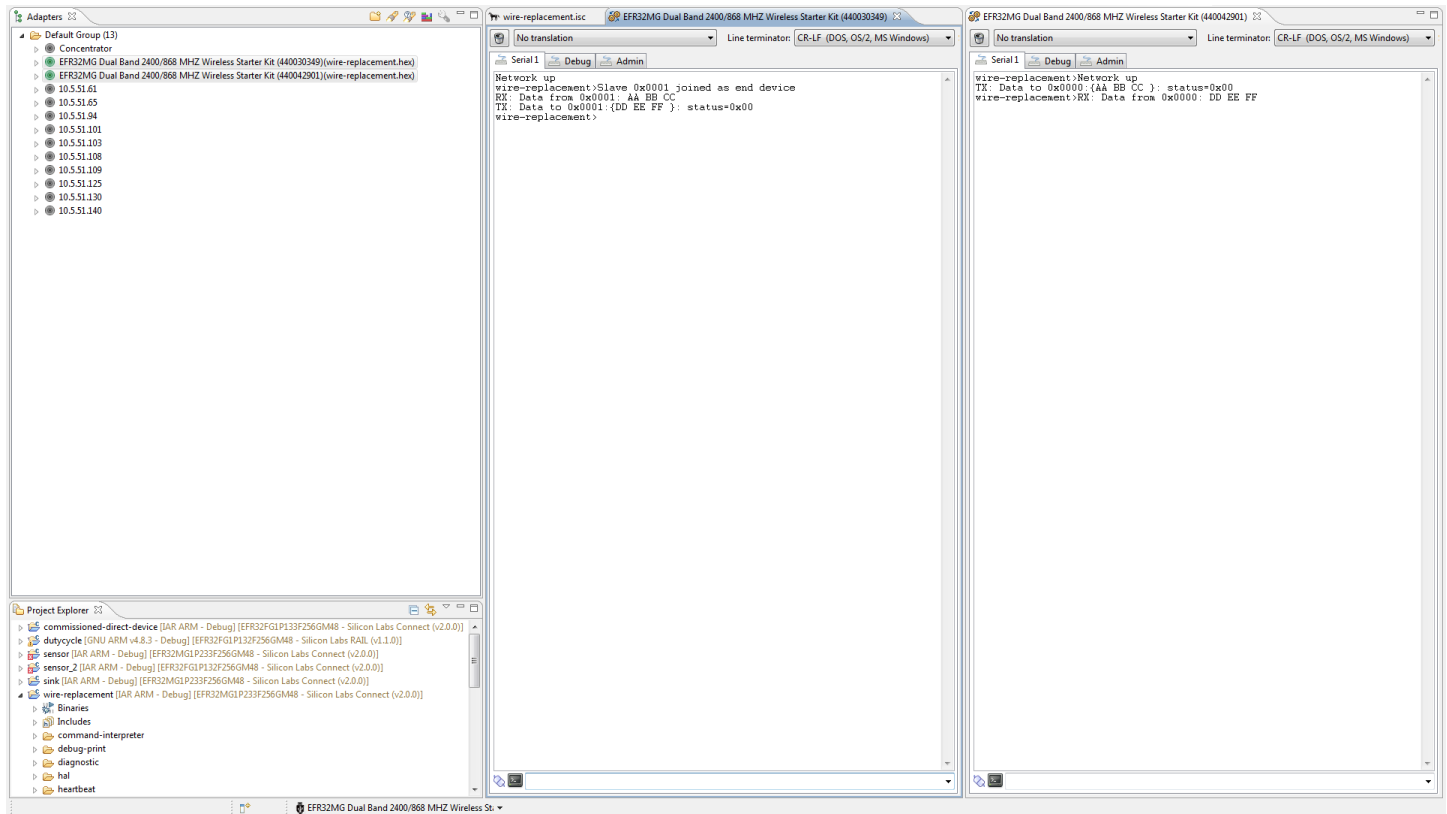


To create the network on the wire-replacement example (comments in parentheses), repeat the procedures in section [4.1.1 Selecting a Connect Example Application](#) through section [4.1.3 Compiling and Flashing the Connect Application](#) to compile and load the Wire Replacement example on a second device.

Then use the following procedure.

1. On Device A, enter `start-master 0` (where 0 is the channel number. You should see `Network Up`).
2. On Device B, enter `start-slave 0` (where 0 is the channel number). You should see:  
 From Device B: `Network Up`  
 From Device A: `Slave 0x0001 joined as end device from device A`
3. On device B, enter `data {AA BB CC}`  
 From Device B: `TX: Data to 0x0000:{AA BB CC } : status=0x00`  
 From Device A: `RX: Data from 0x0001: AA BB CC`
4. On Device A enter `data {DD EE FF}`  
 From Device A: `TX: Data to 0x0001:{DD EE FF } : status=0x00`  
 From Device B: `RX: Data from 0x0000: DD EE FF`

The following figure shows the console at the end of the procedure.

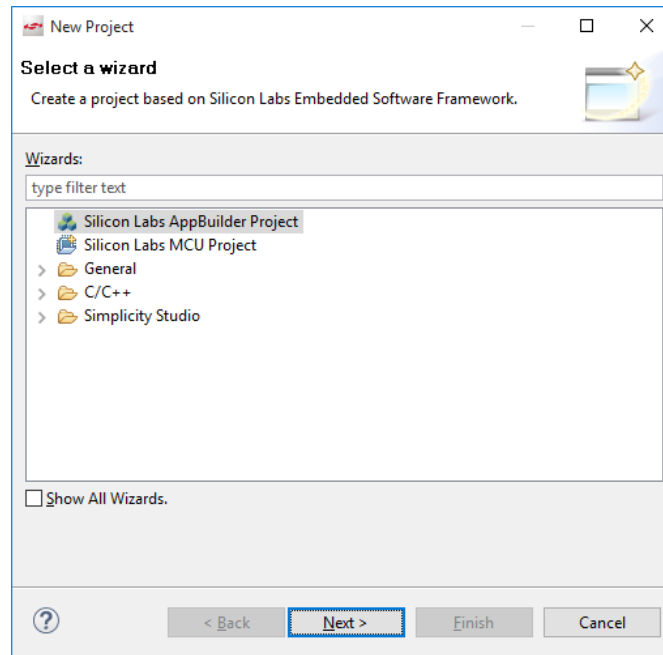


## 4.2 Working with Connect Host/NCP Examples

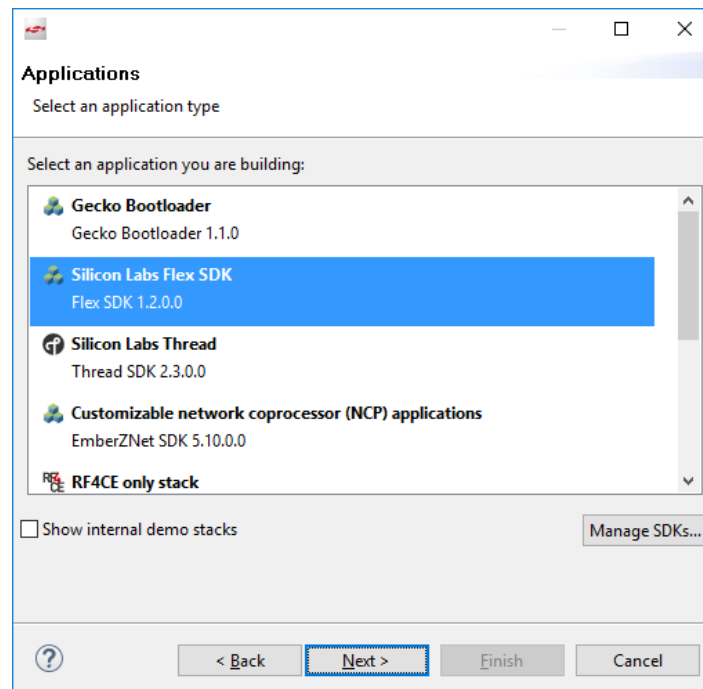
As well as describing how to work with Host/NCP applications, this section also illustrates how to create a new project without leaving the Simplicity IDE.

### 4.2.1 The NCP Application

1. In the Simplicity IDE (AppBuilder), select File > New > Project.
2. Select Silicon Labs AppBuilder Project, and click **Next**.

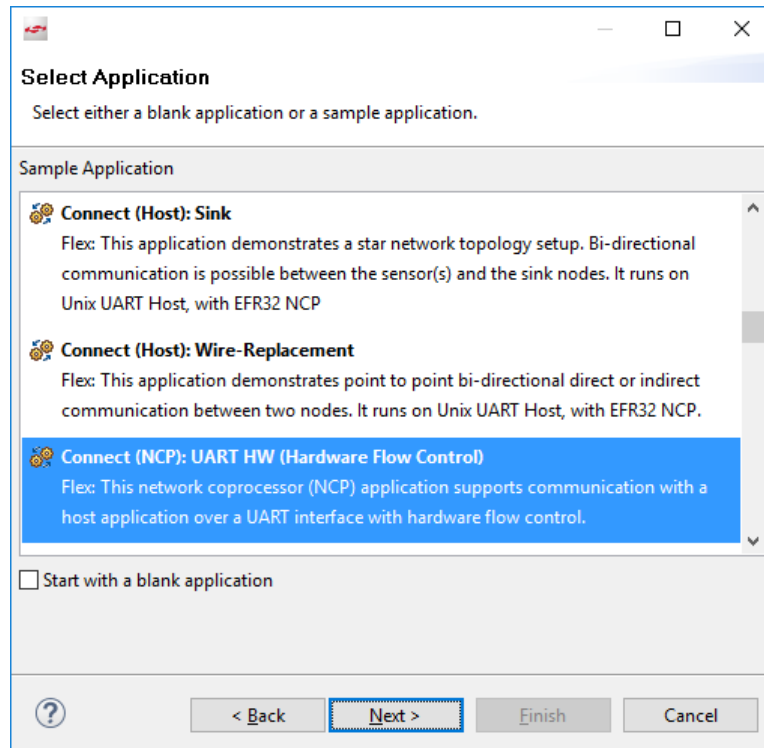


3. Select Silicon Labs Flex SDK and click **Next**.

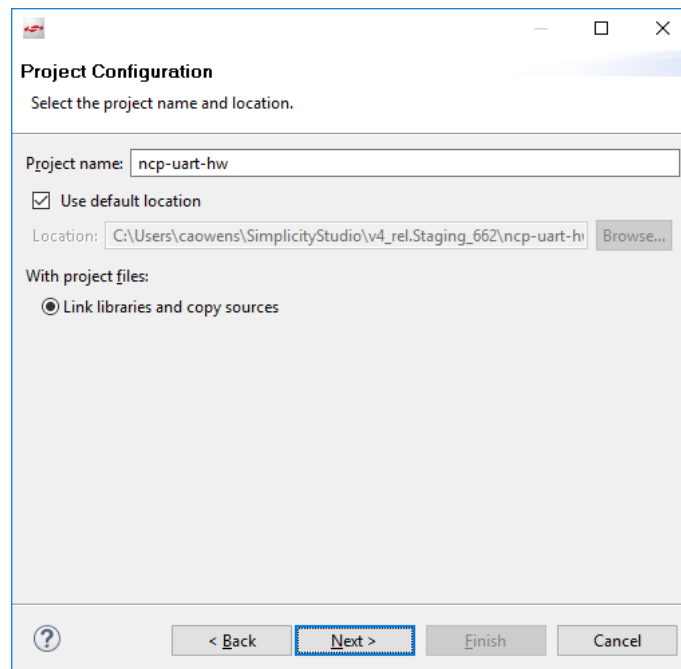




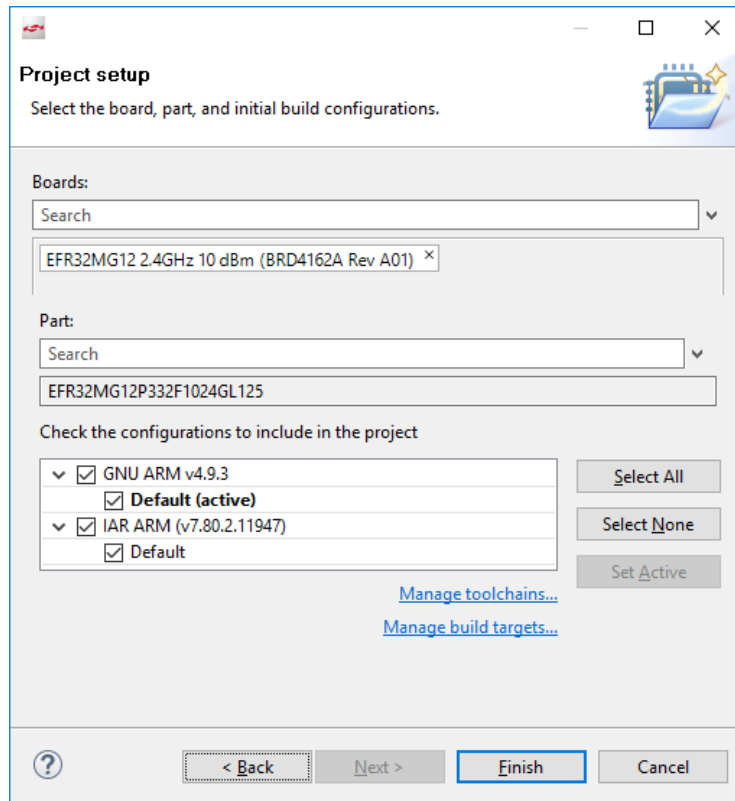
4. Select either **Connect NCP: UART HW Flow Control** or **Connect NCP: UART SW Flow Control** example application and click **Next**.



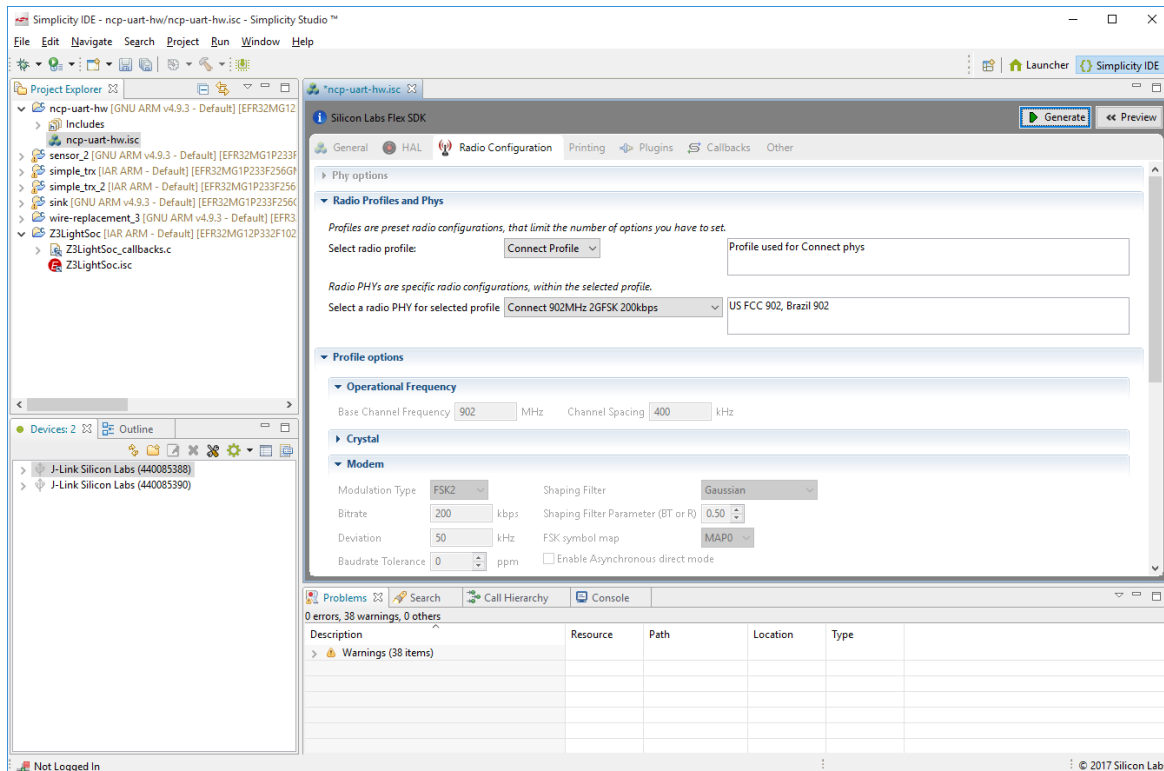
5. (optional) Edit the project name and/or location, then click **Next**.



- Verify that the board and parts are correct (if you have one connected it should be displayed). Select a toolchain if you have more than one. Click **Finish**.

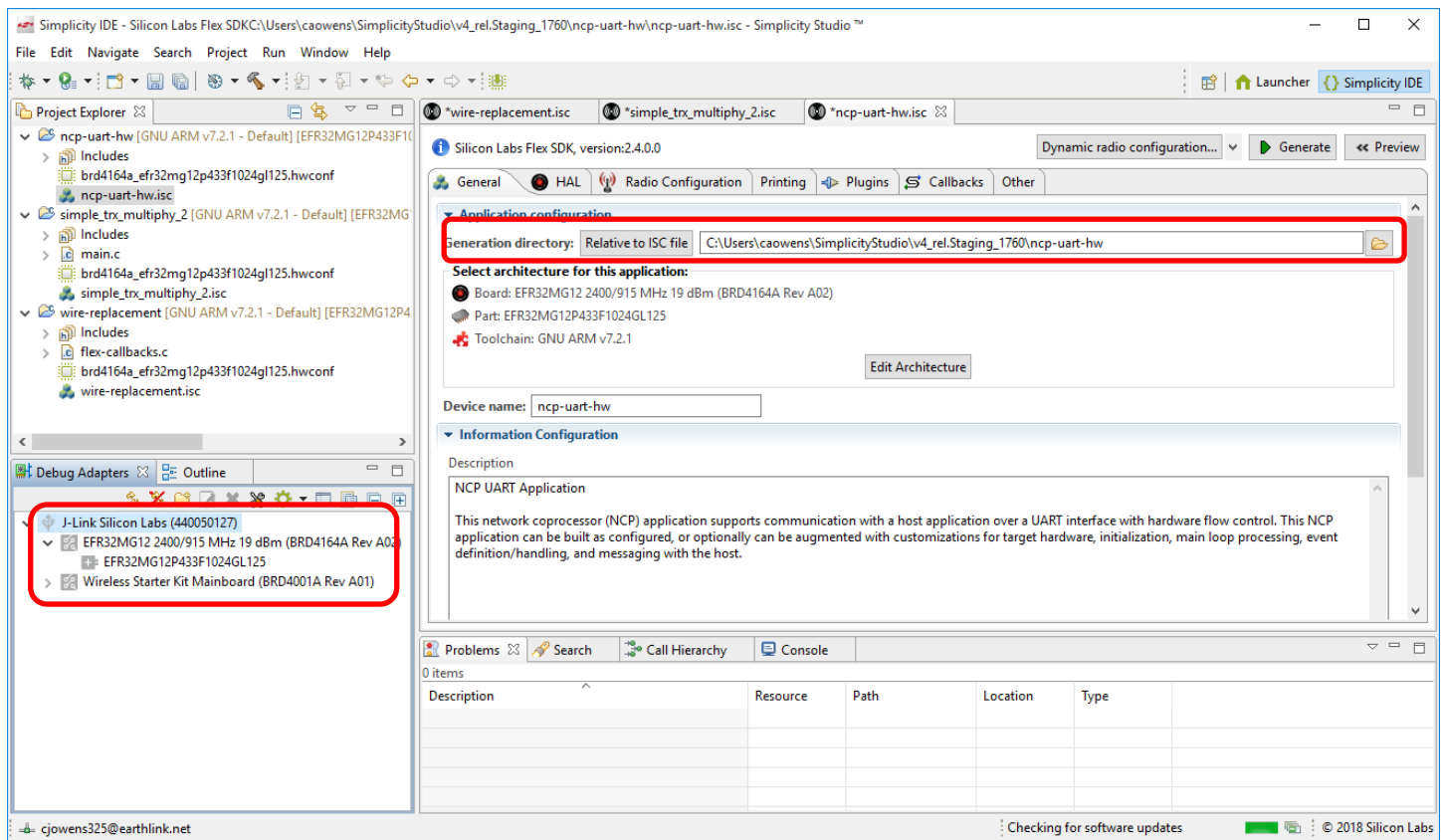


- Check settings in the Radio Configuration tab to make sure that they are correctly configured for the selected board.

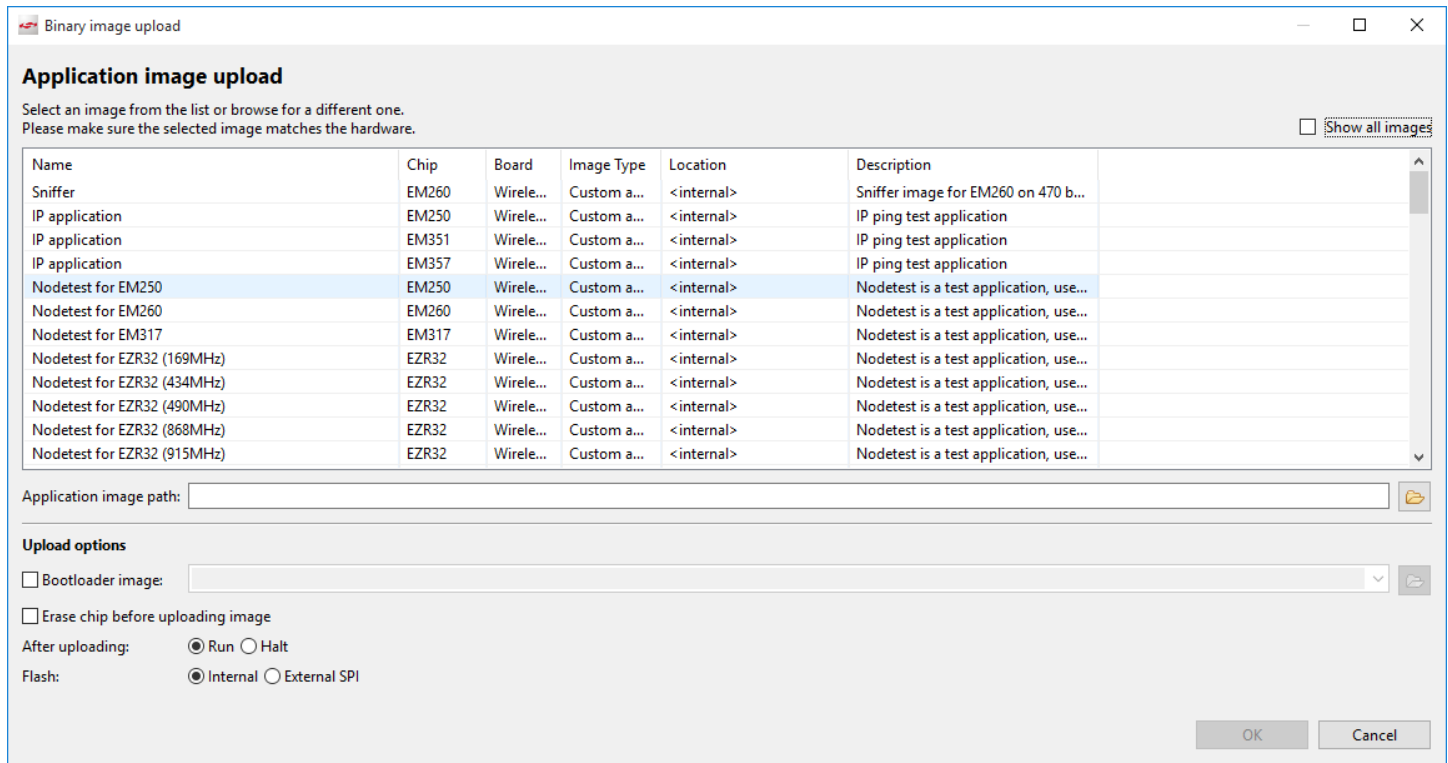


- Click **Generate** to generate project files, then click **OK**.

9. Click the target device in the Devices view and click build .
10. Both NCP examples require a standalone bootloader. To load the application and the bootloader images, first make sure your hardware is displayed in the Device perspective. Expand the radio board to show the part number, as you will need that along with the board number to find the correct bootloader file. Note that the folder in which the example was generated is displayed on the General tab.



- In the Devices view, right-click on the J-Link target device and select **Upload Application**. The Select Binary Image dialog is displayed.



- Browse to the folder with your compiled application and select the .gbl file (see section 1.2 The Gecko Bootloader for more information).

If you compiled the image with GCC, the files are in <folder on General tab>\GNU ARM v<version>.n.n - Default  
If you compiled the image with IAR EWARM, the files are in <folder on General tab>\IAR ARM - Default

- If you have not already loaded a bootloader, check **Bootloader image**, then browse to the folder containing a prebuilt bootloader image. Images are located in the Simplicity Studio platform bootloader folder under sample apps. In this case open the 'bootloader-uart-xmodem' folder, for example:

C:\SiliconLabs\SimplicityStudio\v4\developer\sdk\gecko\_sdk\_suite\<version>\platform\bootloader\sample-apps\bootloader-uart-xmodem).

Open the folder that corresponds to your board and part number and select the .s37 file, for example:

\efr32mg12p332f1024gl125-brd4162a\bootloader-uart-xmodem-combined.s37.

14. Check **Erase Chip**, to make sure that the main flash block is erased before your new image is uploaded. New users will typically always check this.
- The **After uploading** options are **Run** (begin executing the code immediately) and **Halt** (wait for an event, such as a debugger to connect or manual initiation of a boot sequence). During initial development you will typically leave this set to **Run**.
  - The Flash options determine the storage location, and are **Internal** and **External SPI**. Leave the option set to **Internal**.
- Your completed dialog should resemble the following:

**Binary image upload**

**Application image upload**

Select an image from the list or browse for a different one.  
Please make sure the selected image matches the hardware.

☐ Show all images

| Name                        | Chip  | Board     | Image Type  | Location   | Description                            |
|-----------------------------|-------|-----------|-------------|------------|----------------------------------------|
| Sniffer                     | EM260 | Wirele... | Custom a... | <internal> | Sniffer image for EM260 on 470 b...    |
| IP application              | EM250 | Wirele... | Custom a... | <internal> | IP ping test application               |
| IP application              | EM351 | Wirele... | Custom a... | <internal> | IP ping test application               |
| IP application              | EM357 | Wirele... | Custom a... | <internal> | IP ping test application               |
| Nodetest for EM250          | EM250 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EM260          | EM260 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EM317          | EM317 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EZR32 (169MHz) | EZR32 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EZR32 (434MHz) | EZR32 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EZR32 (490MHz) | EZR32 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EZR32 (868MHz) | EZR32 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Nodetest for EZR32 (915MHz) | EZR32 | Wirele... | Custom a... | <internal> | Nodetest is a test application, use... |
| Railtest for EFR32 xq1      | EFR32 | Wirele... | Custom a... | <internal> | RailTest test application              |

Application image path: C:\Users\caowens\SimlicityStudio\v4\_rel\Staging\_1760\ncp-uart-hw\GNU ARM v7.2.1 - Default\ncp-uart-hw.gbl

**Upload options**

☒ Bootloader image: C:\SiliconLabs\SimlicityStudio\v4\developer\sdk\gecko\_sdk\_suite\v2.4\platform\bootloader\sample-apps\bootloader-uart-xmodem\efr32mg12p332f1024gl125-brd4162a\bootloader-uart-xmodem-combined.s37

☒ Erase chip before uploading image

After uploading: ☒ Run ☐ Halt

Flash: ☒ Internal ☐ External SPI

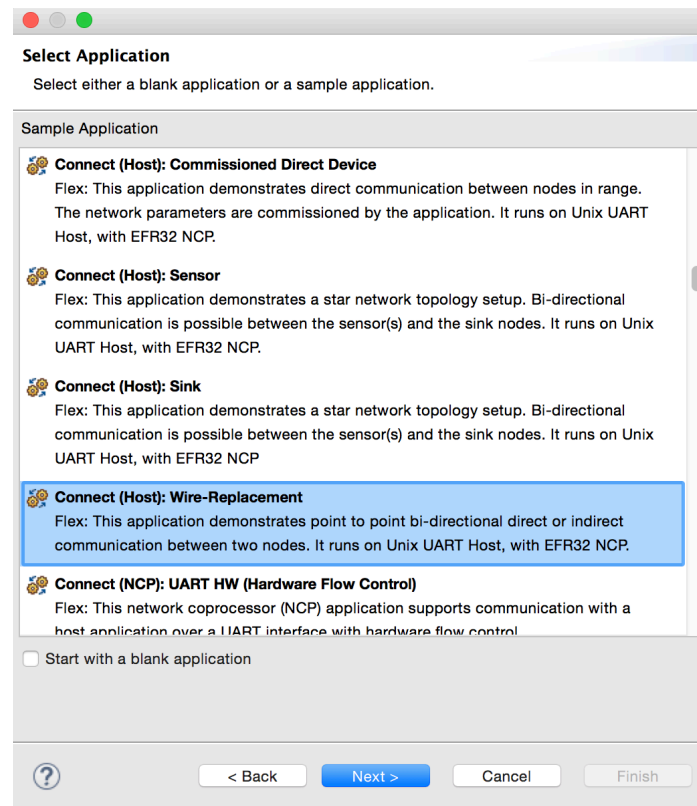
OK Cancel

15. Click **OK**.

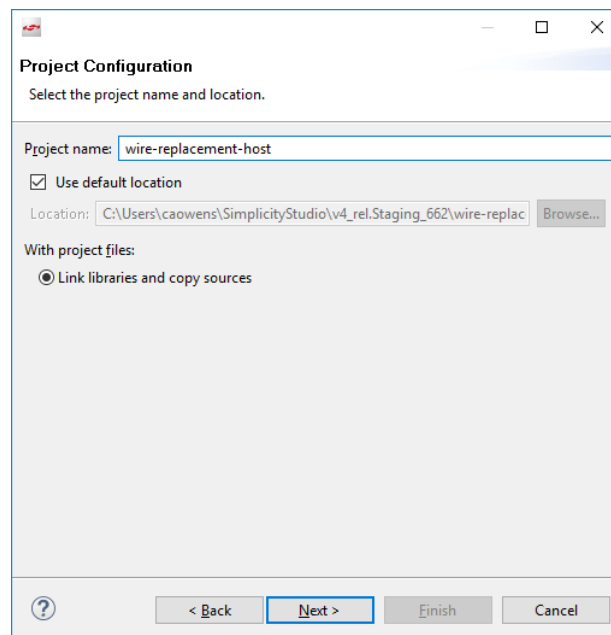
## 4.2.2 The Host Application

The process to build the host application is similar to the process to build the target NCP application.

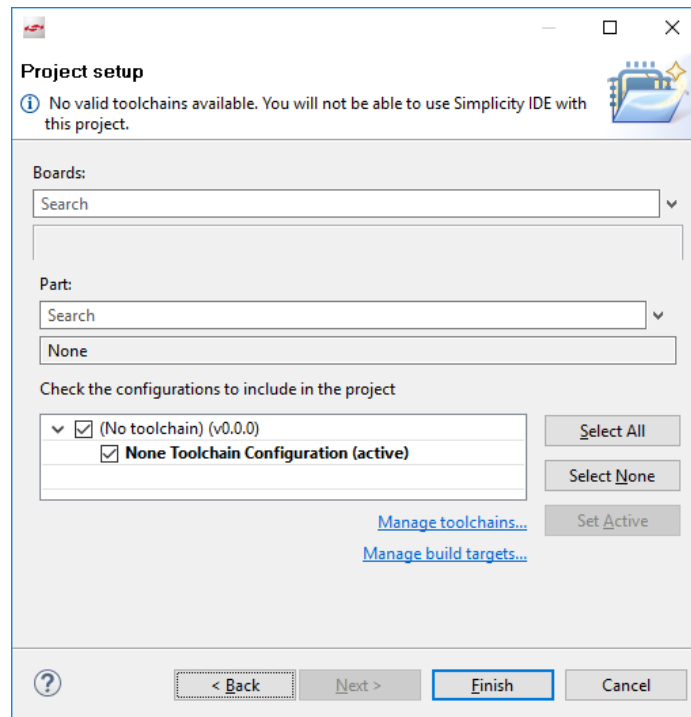
1. In AppBuilder, select **File > New > Project > Silicon Labs AppBuilder Project** to create a new application. Select a Host sample application and click **Next**.



2. (optional) Edit the project name and/or location, then click **Next**.



3. Verify that Boards and Part are not selected, then click **Finish** to start building your application.



4. Click **Generate**.
5. Compile the generated Host Makefile with GCC or Clang, by executing the following command in your generation directory:  

```
make -f wire-replacement-host.mak
```
6. Before executing the host application, identify your USB TTY driver device, which is connected to your NCP. The TTY driver device is usually `/dev/ttyUSB0`.
7. Run the application with the following command:

```
sudo build-wire-replacement-host/wire-replacement-host-unix-host-app -u /dev/ttyUSB0
```

The terminal acts as the platform through which you execute CLI commands.

#### 4.2.3 Testing your Setup

The bootstrap test verifies that the application can perform an initial handshake between the host and NCP. The text 'ASHv3 is up' will be displayed when ASHv3 successfully performs an initialization handshake, and then the application will quit.

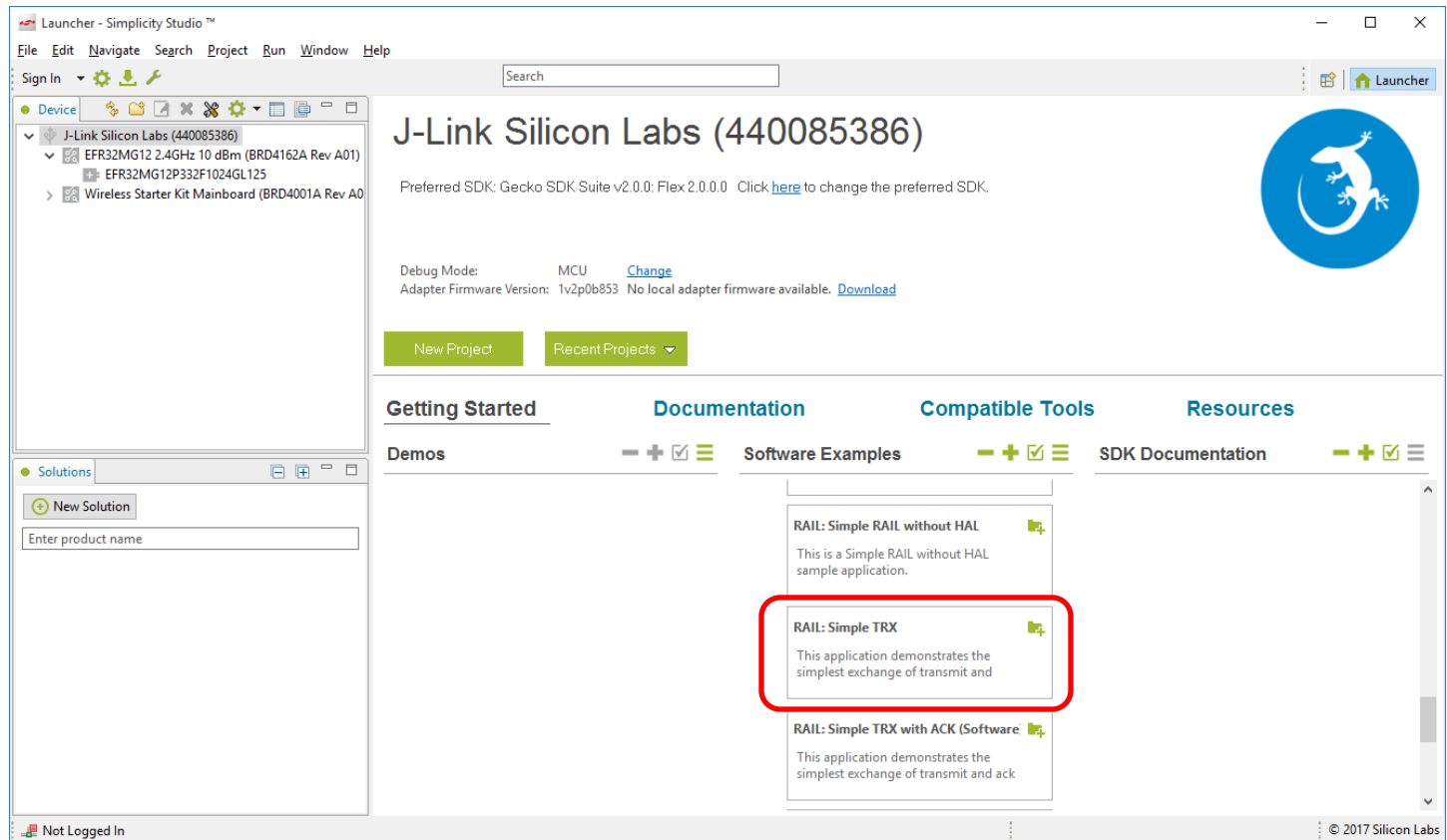
In order to run the test, execute the following command:

```
sudo build-wire-replacement-host/wire-replacement-host-unix-host-app -u /dev/ttyUSB0 --test-bootstrap
```

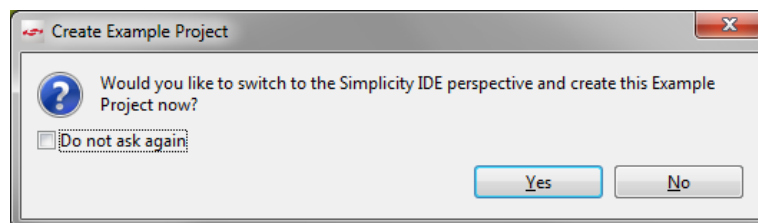
## 4.3 Working with RAIL Examples

### 4.3.1 Selecting a RAIL Example Application

1. In the Launcher perspective, expand the Silicon Labs Flex SDK list and click on an example application. RF engineers may wish to start with **RAIL: RAILTEST**. Firmware engineers may wish to start with the **RAIL: Simple TRX** example. This section uses the **RAIL: Simple TRX** example. In the Launcher perspective, click on that example.

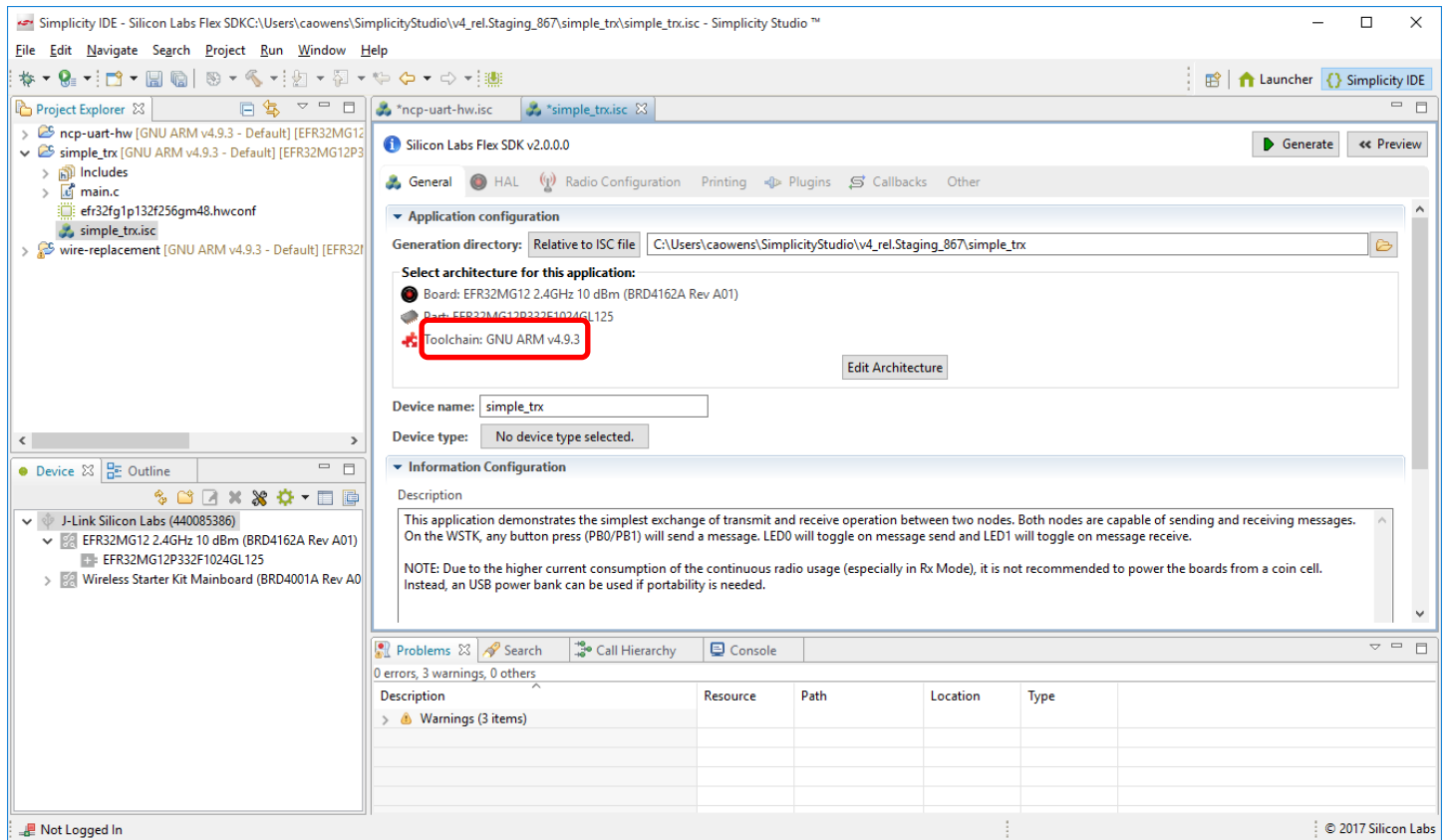


2. You are asked if you want to switch to the Simplicity IDE. Click **Yes**.



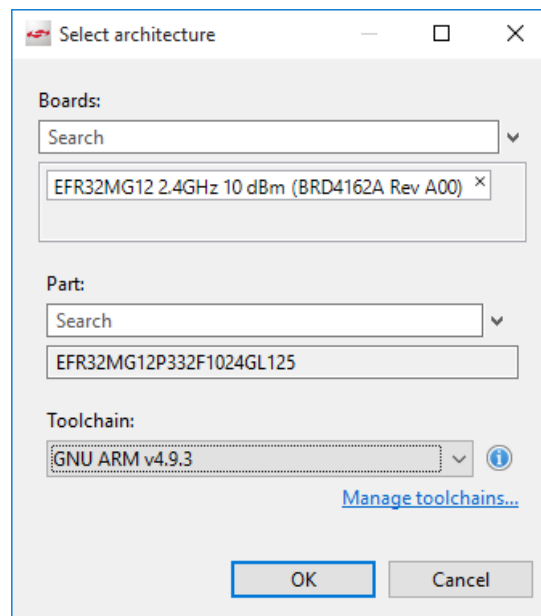


3. Simplicity IDE opens with the new project in AppBuilder view, and the focus on the General tab. Make sure the toolchain shown is the one you want to use. If you have both IAR and GCC installed, GCC is the default.

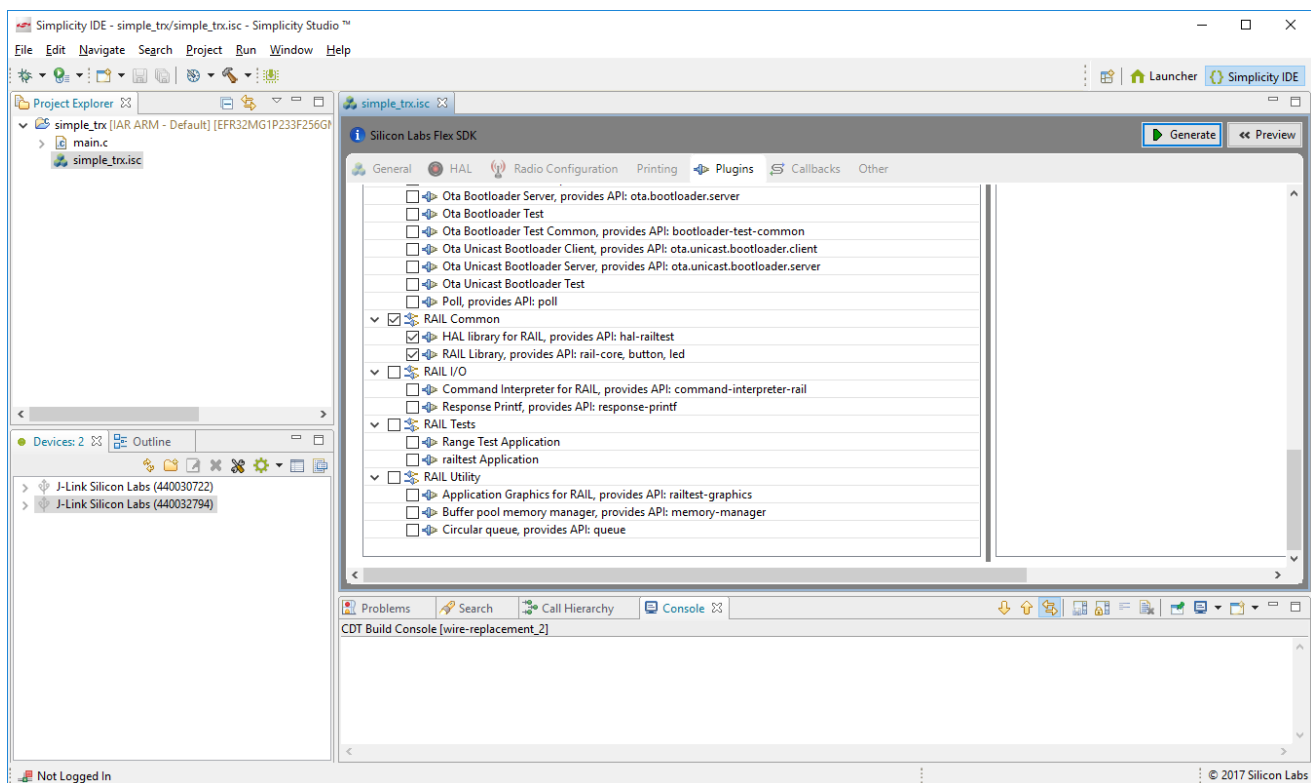


Note: You now have a Simplicity IDE button next to the Launcher button in the upper right.

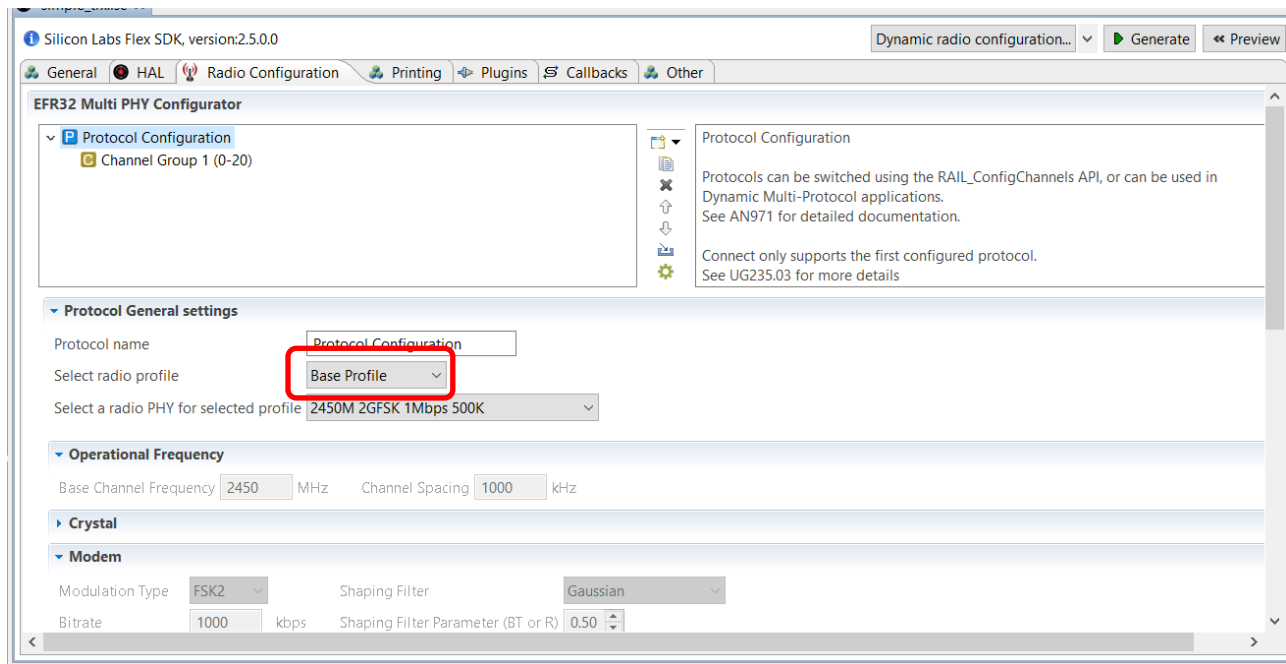
- To change the toolchain click **Edit Architecture**. In the resulting dialog, select the desired toolchain and click **OK**.



Compare the plugins tab for a RAIL application with the one shown for Connect. Only RAIL plugins are selected. Note that, in order to see information about the plugin, you either need to scroll up, or collapse the Connect plugins

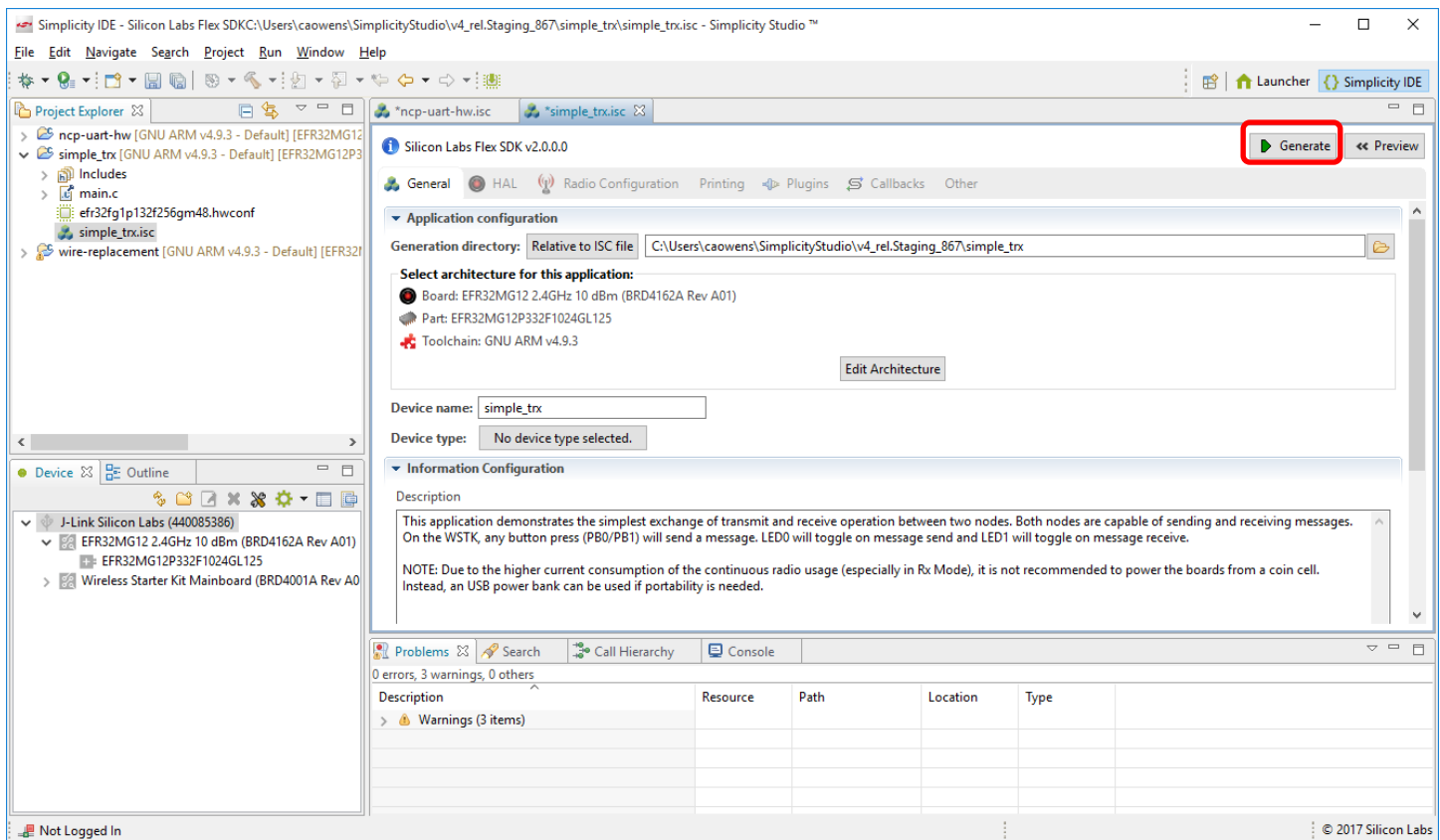


The RAIL application framework allows you to modify the PHY configuration for the application. Click the Radio Configuration tab, then click **Protocol Configuration** in the top left. You can then select a radio profile, and a pre-defined radio PHY of the selected profile. You can then customize the PHY by selecting the "Custom settings" in the PHY list. See AN971: *EFR32 Radio Configurator Guide* for more information on using the radio configurator.

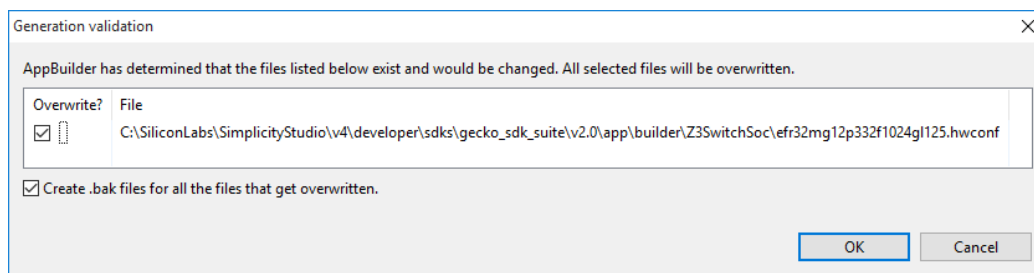


## 4.3.2 Generating the RAIL Application

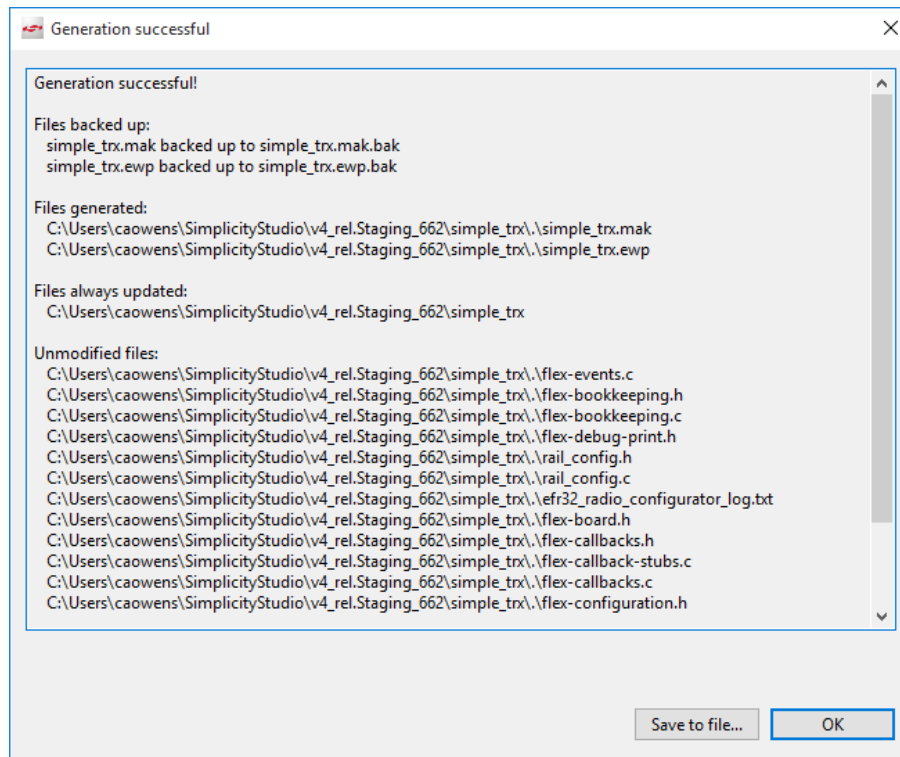
1. In the General tab, the Information Configuration block provides details about application setup and functionality (you may need to scroll down to see it). When you are ready to generate the sample project code, click **Generate**.



If you get the following overwrite warning, click **OK**.



- Once generation is complete, a dialog reporting results is displayed. Click **OK**.

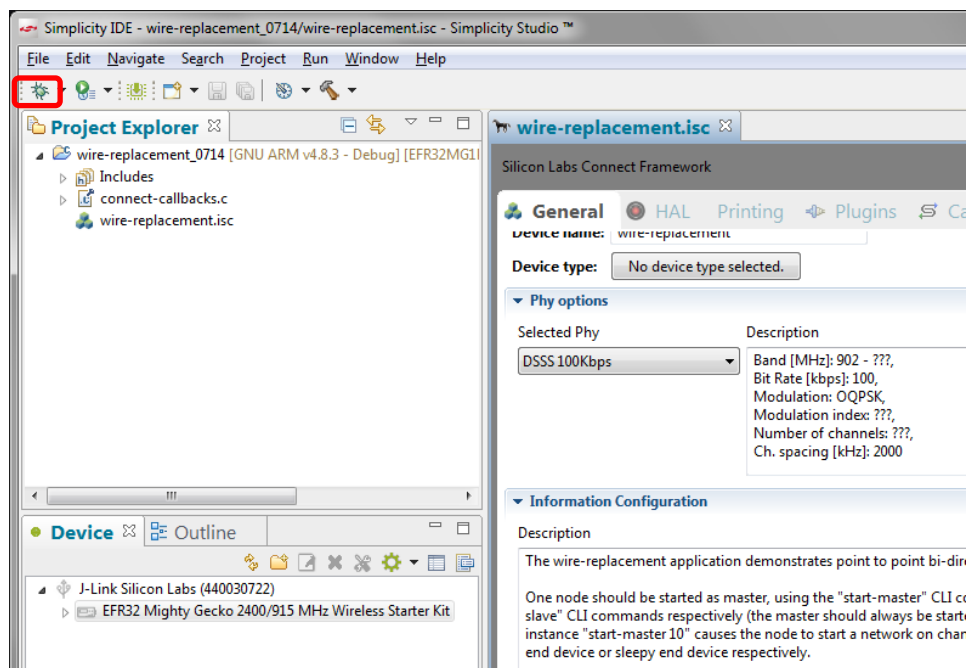


### 4.3.3 Compiling and Flashing the RAIL Example

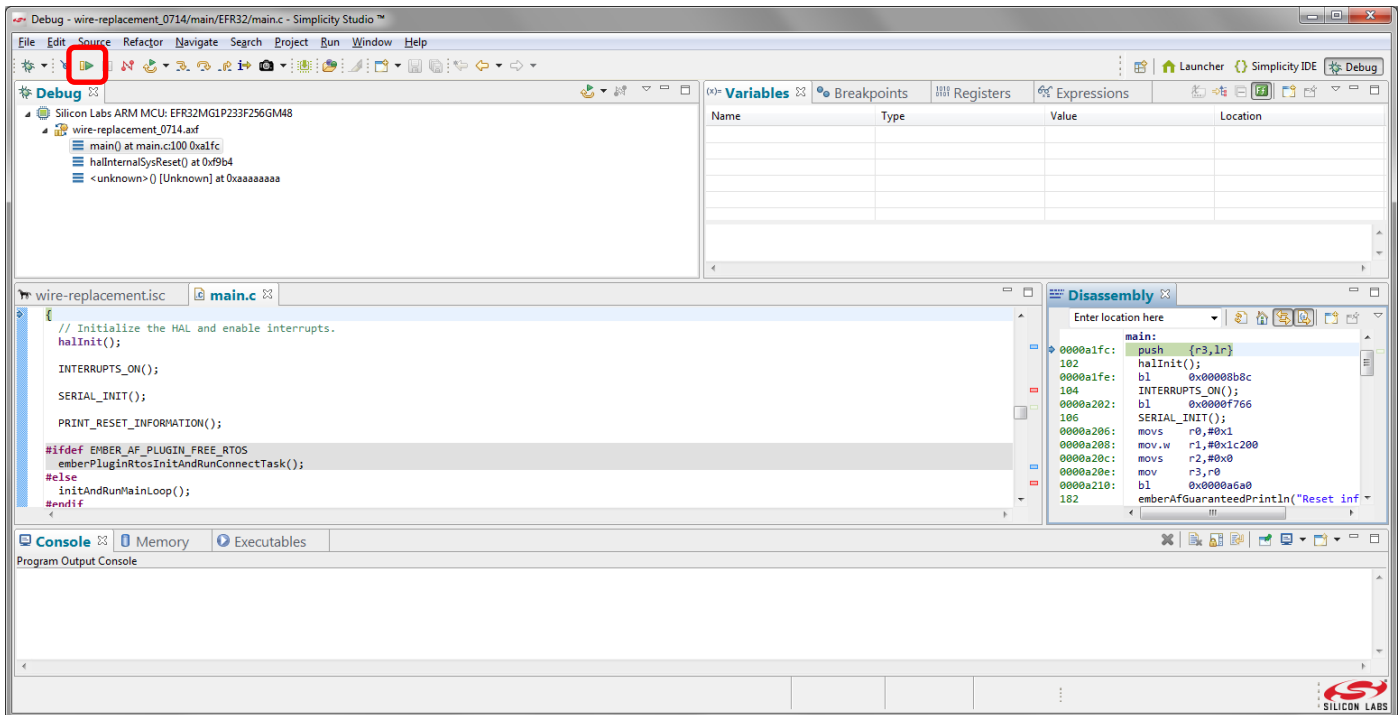
You can either compile and flash the application automatically, or manually compile it and then flash it.

#### Automatically Compile and Flash

- You can automatically compile and flash the application to your connected development hardware in the Simplicity IDE. After you click **OK** on the Generation Confirmation dialog, the Simplicity IDE returns. Click the **Debug** control.



- Progress is displayed in the lower left. Wait until flashing has completed and a Debug perspective is displayed. Click the **Resume** control to start the application running on the WSTK.



Next to the Resume control are Suspend, Disconnect, Reconnect, and stepping controls. Click the **Disconnect** control when you are ready to exit Debug mode.



## Manually Compile and Flash

After you generate your project files, instead of clicking Debug in the Simplicity IDE, click the **Build** control  in the top tool bar.

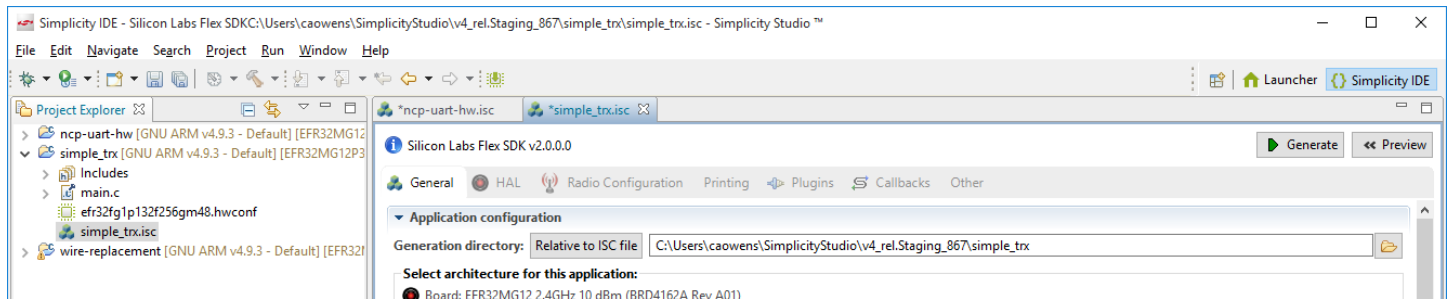
Your sample application will compile based on its build configuration. You can change the build configuration at any time by right clicking on the project and going to **Build Configurations > Set Active**.

You can also build your application directly in IAR-EWARM by opening IAR-EWARM and opening the generated project file inside IAR.

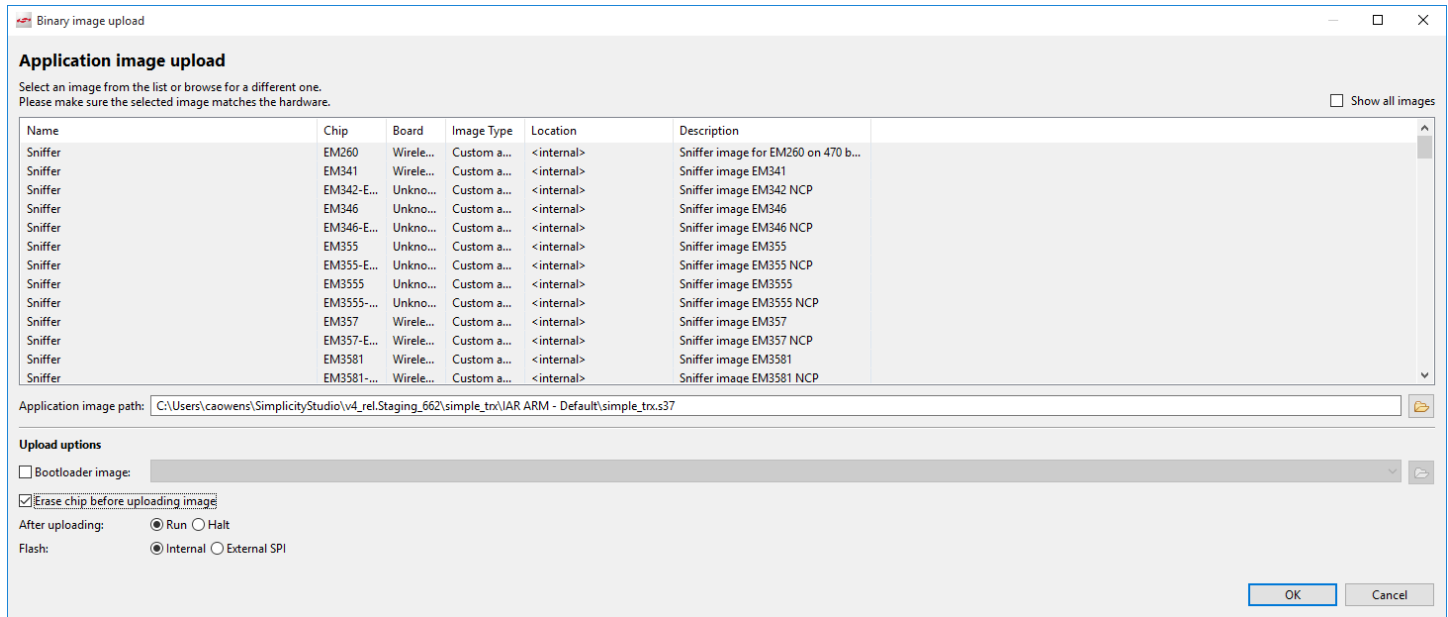
- Open IAR-EWARM.
- Select File > Open > Workspace and navigate to the location you selected for your sample application.
- Select the application .eww file and click **[Open]**.
- Select Project > Make or press F7. If the application builds without errors, you are ready to install the image on a device.

You can load the binary image through the Devices view in the Simplicity IDE perspective.

- Files are generated into the folder on the General tab.



- In the Devices view, right-click on the J-Link device and select **Upload Application**. The Select Binary Image dialog is displayed.



- Navigate to the .s37, .bin or .hex image you wish to upload. Files are located under the project folder on the General tab:  
If you compiled the image with GCC, the files are in <folder on General tab>\GNU ARM vn.n.n - Default  
If you compiled the image with IAR EWARM, the files are in <folder on General tab>\IAR ARM - Default
- Optionally, check **Erase Chip**, to make sure that any previous bootloader or other non-volatile data is erased before your new image is uploaded. New users will typically check this.
- The **After Load** options are **Run** (begin executing the code immediately) and **Reset** (wait for an event, such as a debugger to connect or manual initiation of a boot sequence). During initial development you will typically leave this set to **Run**.
- Click **OK**.

#### 4.3.4 Interacting with the RAIL Example

Interaction in the simple TRX example is through buttons on the WSTK.

Repeat the procedures in section 4.3.1 [Selecting a RAIL Example Application](#) through section 4.3.3 [Compiling and Flashing the RAIL Example](#) to compile and load the simple TRX example on a second device.

Press either button on either device to send a message. On transmission, LED0 on the transmitting device toggles on, and on reception, LED1 on the receiving device toggles on.

#### 4.4 Starting with a Blank Application

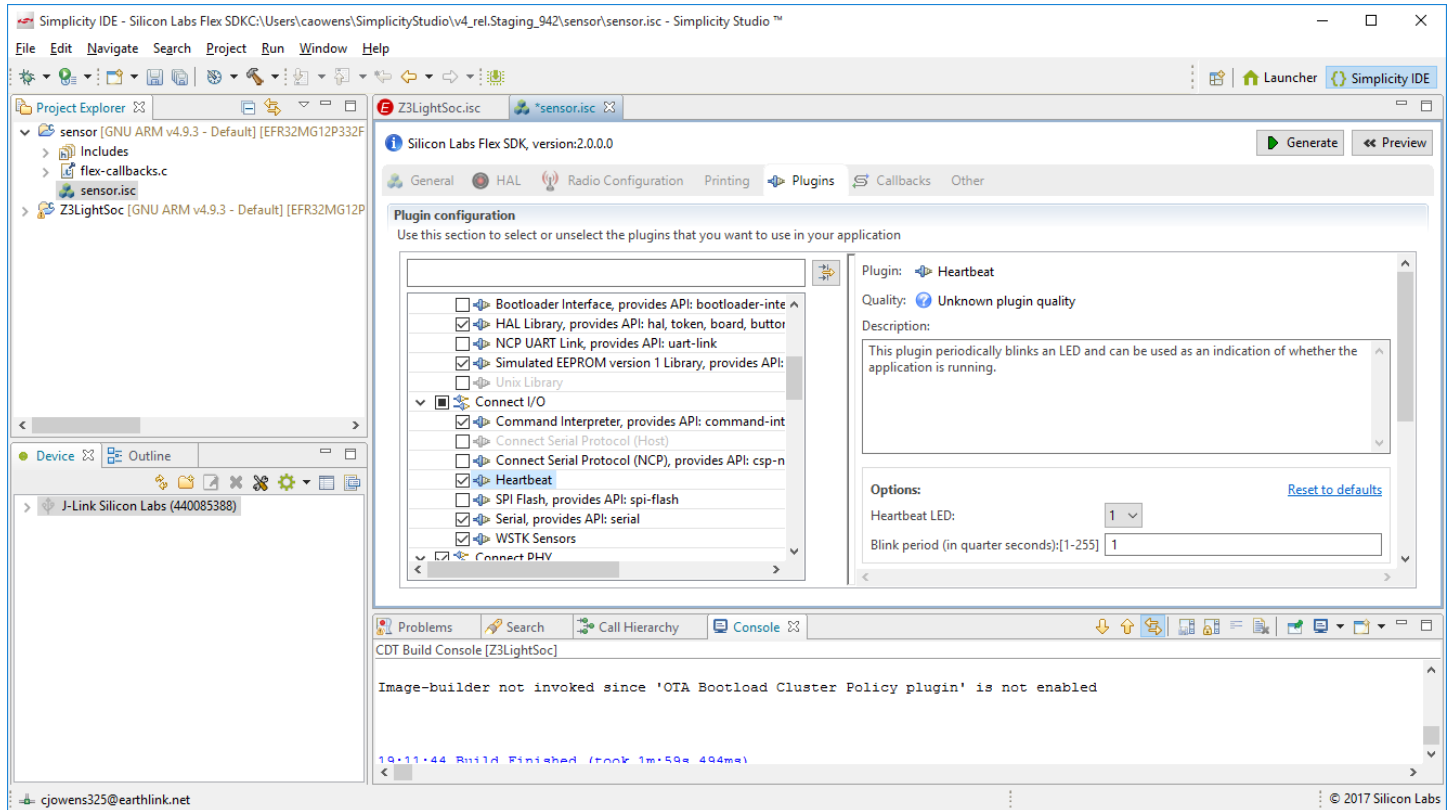
While Silicon Labs strongly recommends starting development from one of the example applications, if you want to start with a blank Connect application you can.

1. On the launcher page click **New Project**.
2. Click **Silicon Labs Flex SDK**, and click **Next**.
3. Check Start with a blank application, and click **Next**.
4. Name your application, and click **Next**.
5. Click **Finish**. The Simplicity IDE opens, but nothing is configured. The project will not build until it has been configured.

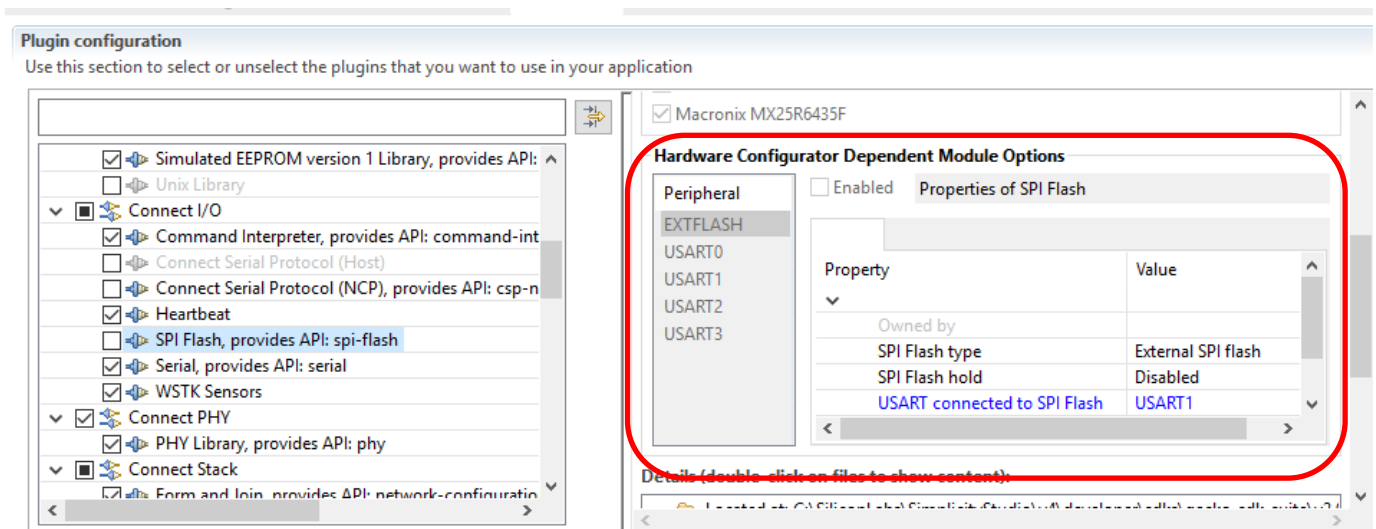


## 5 Next Steps

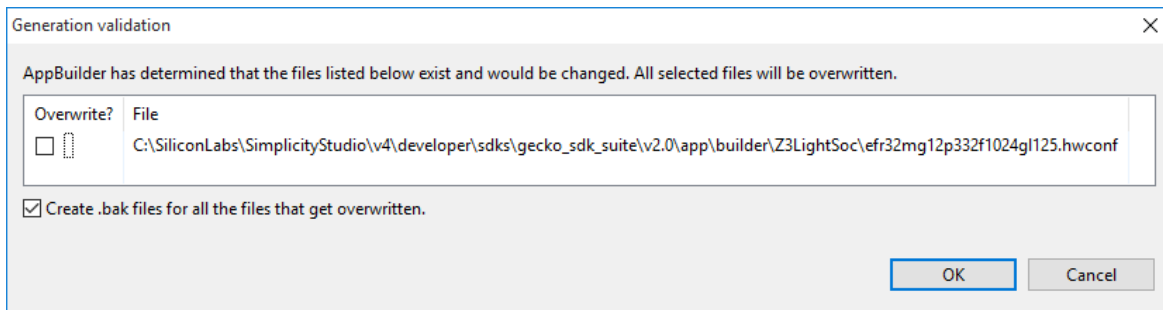
Explore configuring the example application to meet your needs. Much of the software configuration can be done through plugins. Select a plugin to see more information about it and its configuration options, if any.



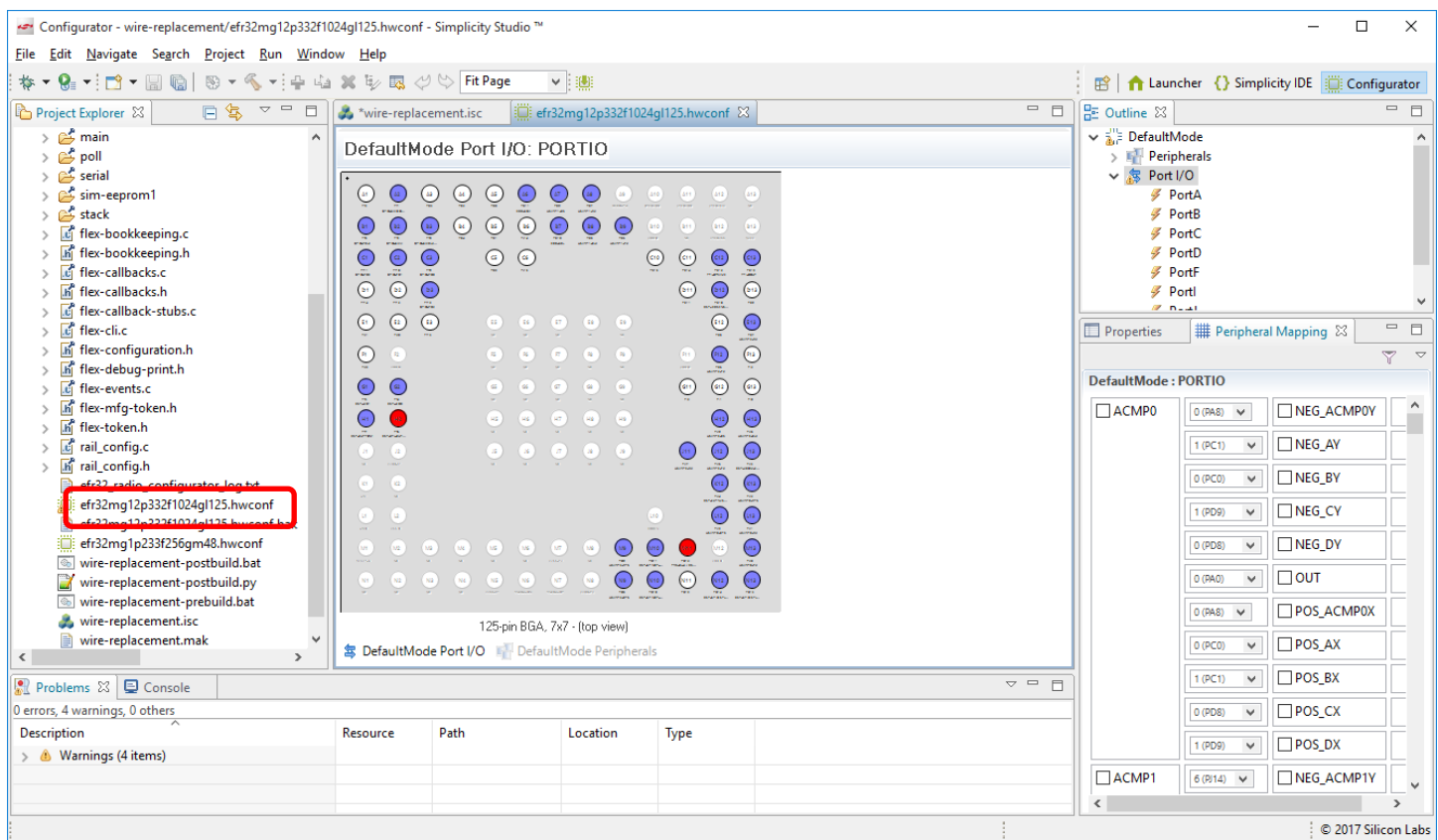
Simplicity Studio offers a Hardware Configurator tool that allows you to easily configure new peripherals or change the properties of existing ones. Hardware Configurator options are available on many of the HAL and I/O plugins.



**Note:** If you change hardware configuration options your changes are saved to a temporary file. In order to have the changes included in the project, you must check the Overwrite checkbox in the following dialog, displayed at the end of file generation.



You can also open the Hardware Configurator tool by double-clicking the .hwconf file that was generated along with your other project files, or by clicking **Open Hardware Configurator** on the HAL tab. Some configuration options are only available through the Hardware Configurator tool. Click **DefaultMode Peripherals** under the pin graphic to change to a peripherals view.



See *AN1115: Configuring Peripherals for 32-Bit Devices in Simplicity Studio* for more information about the Hardware Configurator both within the Simplicity IDE and as a separate tool.

## 6 Connect and RAIL API Documentation

The following is a collection of in-depth documents that cover different aspects of development with RAIL and Connect.

- [QSG138: Proprietary Flex SDK v2.x Quick Start Guide](#) (This document.)
- [Proprietary wireless \(Connect and RAIL\) training series](#). Collection of most up-to-date training resources.

The Silicon Labs Fundamentals series can be used as an introduction to developing wireless networking applications with the Silicon Labs EFR32 chips and development tools.

- [UG103.1: Wireless Networking Application Development Fundamentals](#)
- [UG103.4: HAL Fundamentals](#)
- [UG103.6: Bootloading Fundamentals](#)
- [UG103.7: Non-Volatile Data Storage Fundamentals](#)
- [UG103.12: Silicon Labs Connect Fundamentals](#)
- [UG103.13: RAIL Fundamentals](#)
- [UG103.16: Multiprotocol Fundamentals](#)

RAIL and Device Configuration Documentation

- RAIL API Reference Documentation <https://docs.silabs.com/rail/latest/>
- RAIL Test User Guide (available in Simplicity Studio)
- [AN1115: Configuring Peripherals for 32-Bit Devices in Simplicity Studio](#)
- [AN1119: Using RAIL for Wireless M-Bus Applications with EFR32](#)
- [AN1127: Power Amplifier Power Conversion Functions in RAIL 2.x](#)
- [AN971: EFR32 Radio Configurator Guide for RAIL in Simplicity Studio v4](#)

Non-Volatile Data Storage

- [AN1154: Using Tokens for Non-Volatile Data Storage](#)
- [AN961: Bringing Up Custom Devices for the EFR32MG and EFR32FG Families](#)
- [AN703: Using Simulated EEPROM Version 1 and Version 2 for the EM35x and EFR32 SoC Platforms](#)

Dynamic Multiprotocol

- [UG305: Dynamic Multiprotocol User's Guide](#)
- [AN1209: Dynamic Multiprotocol Development with Bluetooth and Connect](#)
- [AN1135: Using Third Generation Non-Volatile Memory \(NVM3\) Data Storage](#)

The *Connect 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.

- [UG235.01: Developing Code with Silicon Labs Connect v2.x](#)  
General overview of the code development process and supporting tools for Connect-based applications.
- [UG235.02: Using IEEE 802.15.4 with Silicon Labs Connect v2.x](#)  
Short introduction to the IEEE 802.15.4 features that are used in Connect.
- [UG235.03: Architecture of the Silicon Labs Connect Stack v2.x](#)  
Complete review of the Connect stack architecture, capabilities and features.
- [UG235.04: Customizing Applications with Silicon Labs Connect v2.x](#)  
Describes the plugins and callbacks implemented in the Silicon Labs Application Builder for Connect applications.
- [UG235.05: Using Micrium OS \(RTOS\) with Silicon Labs Connect v2.x](#)  
Describes the process of implementing a Connect-based application on top of the Micrium OS. (Replaces AN1153.)
- [UG235.06: Bootloading and OTA with Silicon Labs Connect v2.x](#)  
Details the bootloader options available for Connect-based applications.
- [UG235.07: Energy Saving with Silicon Labs Connect v2.x](#)  
Provides techniques to realize low-energy consumption in Connect-based applications.
- [UG235.08: Using the Command Line Interface with Silicon Labs Connect v2.x](#)  
Describes how to use the CLI with Connect and how to add a CLI command.

- [UG235.09: Using NCP with Silicon Labs Connect v2.x](#)

Discusses the implementation of a Network Coprocessor (NCP) application, based on the Connect stack.

#### Other Connect Documentation

- Connect Stack API Reference Documentation, <https://docs.silabs.com/connect-stack/latest>
- [AN844: Guide to Host/Network Co-Processor Communications Using Silicon Labs Thread and Connect](#)
- [AN902: Building Low Power Networks with the Silicon Labs Connect Stack v2.x](#)
- [UG266: Gecko Bootloader User's Guide](#)
- [AN1085: Using the Gecko Bootloader with Silicon Labs Connect](#)

Silicon Labs

# Simplicity Studio™4



## Simplicity Studio

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



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



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



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



**Support and Community**  
[community.silabs.com](http://community.silabs.com)

### Disclaimer

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

### Trademark Information

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



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

<http://www.silabs.com>