



AN1160: Project Collaboration with Simplicity Studio

This document presents “best practices” procedures for sharing and collaborating on Simplicity Studio projects among a team of developers or developers and Support.

KEY POINTS

- Collaborating on Simplicity Studio projects through source control.
- Importing received Studio projects.
- Exporting and sharing Studio projects

1. Introduction

This document describes how to accomplish the following tasks:

- Construct a project so that it can be shared with team members.
- Set up source control repositories (repos) for the Simplicity Studio files and for customer projects to be used with Simplicity Studio.
- Share a project with a colleague or a Silicon Labs representative.

Refer to the following sections based on your role and objectives.

Section 2. [Source Code Control](#):

- I want to set up project collaboration with source control for my team.
- I want to construct a new Studio project from an example to use with source control.
- I am part of a team collaborating on Studio projects in source control.

Section 3. [Import Wizard](#):

- I have received a project from a coworker or Silicon Labs representative.
- I want to import a project from another Studio installation.

Section 4. [Export Wizard](#):

- I want to share a project with a coworker or Silicon Labs representative.
- I want to export a project to be added to another Studio installation.

2. Source Code Control

This chapter describes setting up collaboration on Studio projects with source control. If you simply want to import a project you have received or to send a project, see section [3. Import Wizard](#) or [4. Export Wizard](#), respectively.

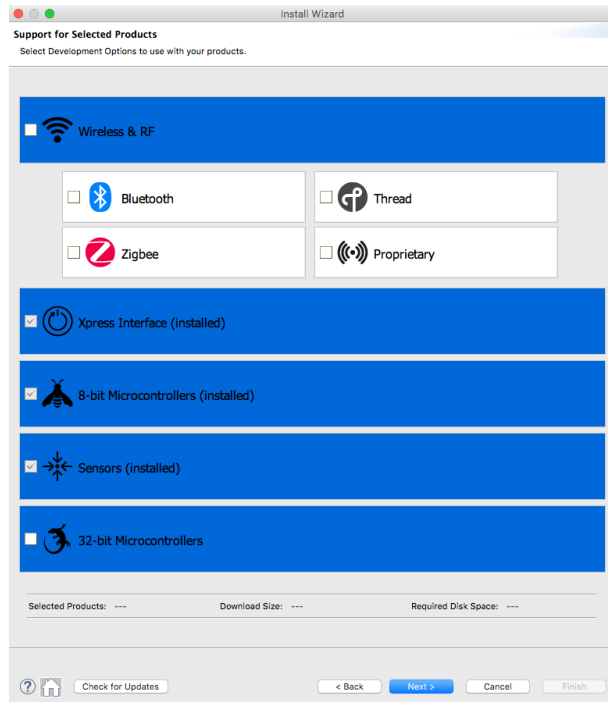
2.1 What to Check In

Simplicity Studio projects are usually dependent on content from a specific Software Development Kit (SDK). To ensure that developers are using the same version of the SDK, it is recommended to check the entire SDK into Source Control. The SDK and Studio projects can be checked into different repos. See the next three sections for instructions on how to check in the SDK and projects.

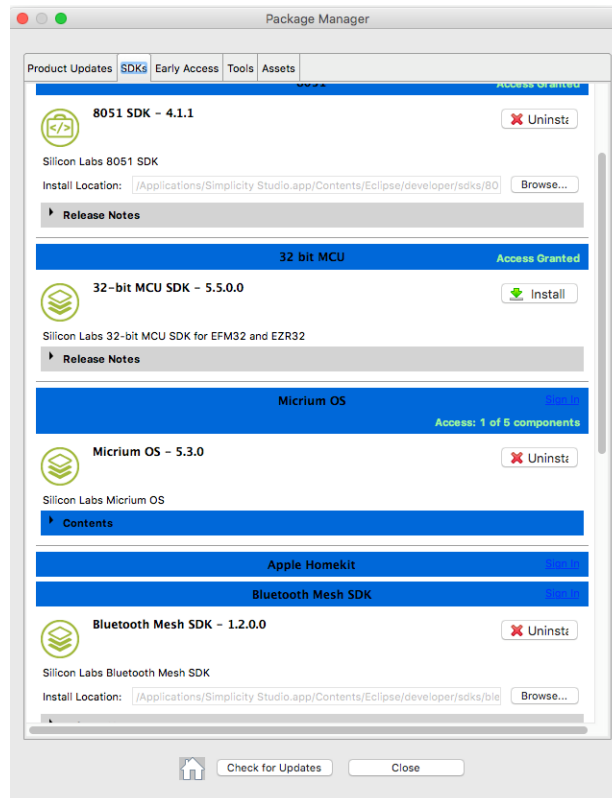
2.1.1 Setting up the Studio Files for Source Control

This section describes how to 1) install an SDK with a product group configuration and 2) how to install only one component of the SDK.

To install a full SDK, open the **Help** menu and select **Update Software**. On the Installation Manager dialog, click **[Install by Product Group]**. Check the product groups you wish to install. This is the recommended procedure as it ensures that all the necessary content for product-specific development is downloaded.



Users who do not wish to install an entire product group can install a single SDK component. Open the **Help** menu and select **Update Software**. On the Installation Manager dialog, click **[Package Manager]** and then click the SDK tab. Click **[Install]** next to the desired component.



In order to maintain the same environment for the whole team, the SDK must be made external to Studio. After installation, copy the entire directory from Studio to the desired repo location. The Studio directories are as follows, where STUDIOLOCATION is your local source Studio file installation location, and SDK_TYPE is **8051** for 8-bit products, **gecko_sdk_suite** for 32-bit and Wireless products, and **blemesh** for the Bluetooth mesh product:

Mac:

```
{STUDIOLOCATION}/Contents/Eclipse/developer/sdks/{SDK_TYPE}/
```

Windows:

```
{STUDIOLOCATION}/v4/developer/sdks/{SDK_TYPE}/
```

Linux:

```
{STUDIOLOCATION}/developer/sdks/{SDK_TYPE}/
```

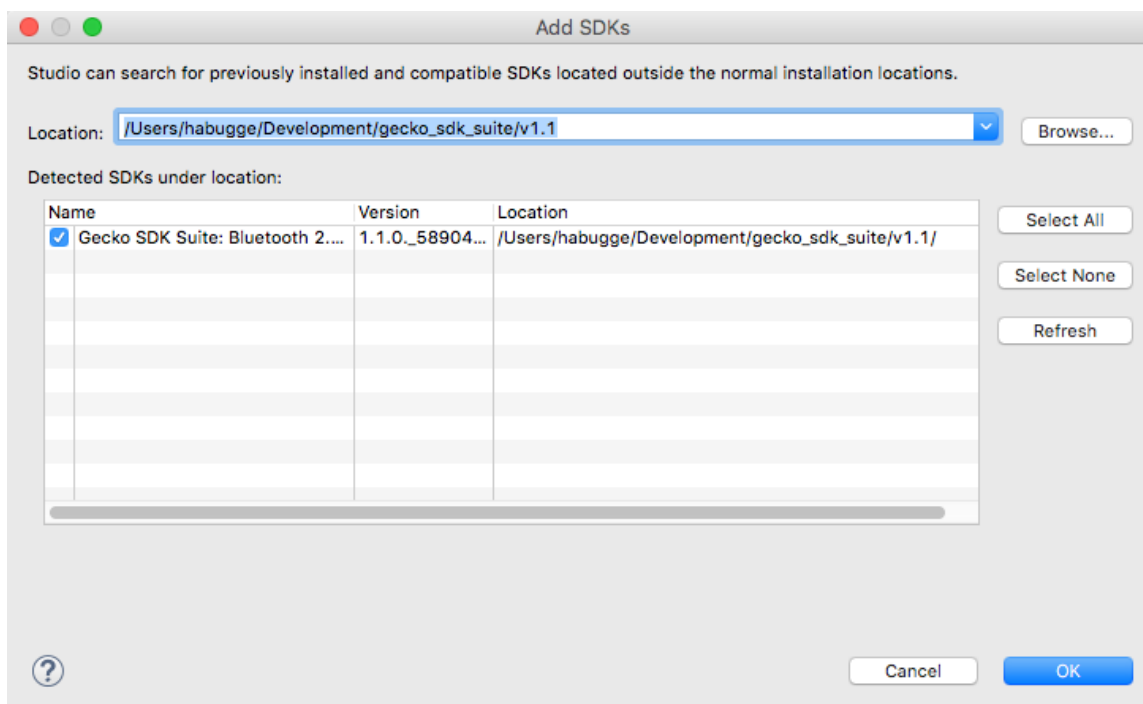
After the directory is copied to the desired location on disk, check the entire directory into source control.

2.1.2 Adding the External SDK to Studio

At this point you have an SDK repo on your file system. The next step is to add it as an external SDK to your Simplicity Studio installation through Simplicity Studio's Preferences dialog. Open the Preferences dialog, select **Simplicity Studio > SDKs** and click **[Add]**. Browse and open the SDK directory:

```
{SDK_REPO_LOCATION}/{SDK_NAME}/{VERSION}/
```

The SDK should show under **Detected SDKs under location**. Click **[OK]**.

**2.1.3 Setting up a Project for Source Control**

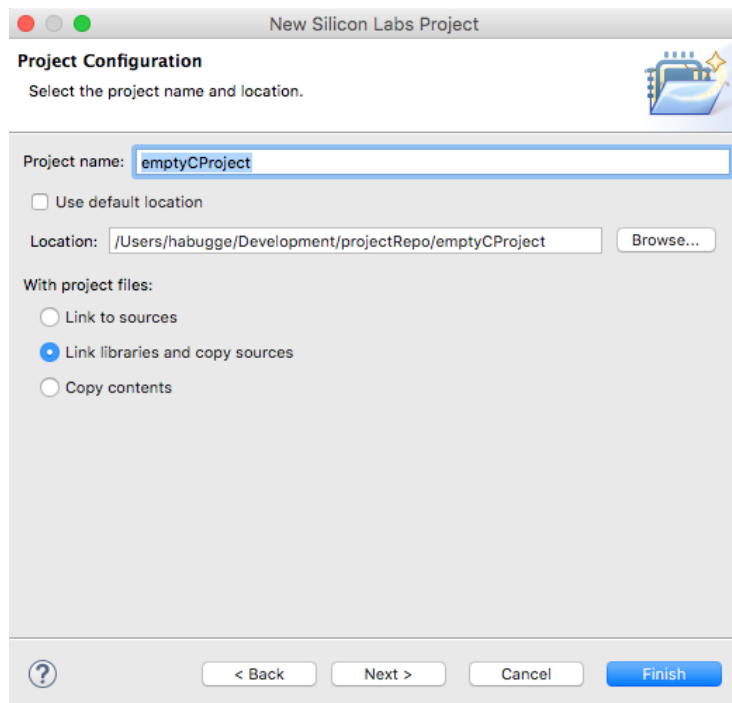
On the **File** menu, select **New > Project** and create a new empty project or a project from an example. You will modify parameters on the Project Configuration and Project Setup pages. Additional decisions on how projects will be generated must be made if you are working on Wireless projects, as described in section [2.1.3.4 App Builder](#).

2.1.3.1 Project Configuration

On the Project Configuration Page, uncheck **Use Default Location** and click [**Browse**]. Navigate to the desired project repo location.

We recommend leaving the **Link libraries and copy sources** option selected. This option only copies the sources that we expect the user to edit. Linking the SDK/Libraries makes it convenient to upgrade the SDK version at a later point, as the files will reference the new SDK. A potential problem with linking is if developers are collaborating on the same project with a different SDK version, and the files in the SDK are different. As long as the SDK is maintained in source control, that is not a problem.

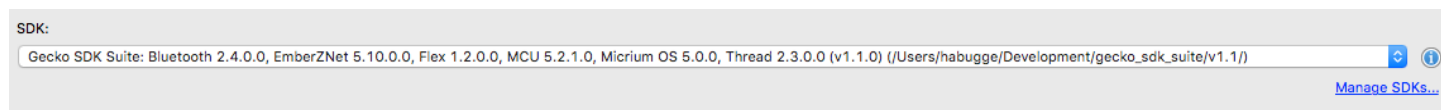
Bluetooth projects are an exception; the contents will always be copied.



2.1.3.2 Project Setup

The project setup page might appear before or after the Project Configuration (it varies depending on project type).

Add boards and part. Make sure the correct SDK is selected in the SDK section.



2.1.3.3 Repo Setup

Create a .gitignore file in the root repo folder, and make sure that build artifacts, system files, and so on are excluded.

An example .gitignore file is:

```
*/*- Debug
*/*- Release
.DS_Store
```

Push the project to source control.

2.1.3.4 App Builder

Wireless projects require different decisions than EFM8 or EFM32 based projects since they include an .isc file that is acted on by the **AppBuilder** executable. The person setting up configuration management of Simplicity Studio-based projects has to decide if:

- The .isc file and support files that are not provided by the Silicon Labs SDK will be pushed into the repository and each engineer that clones the project will open the .isc file with AppBuilder and run the **Generate** function.
- The output of the **Generate** function will be pushed into the repository and engineers cloning the project will not run **Generate** but just build the project from the files cloned from the repository. (The .isc file should still be checked into the repository in this case, but mainly for reference or if the SDK configuration of the project has to be changed in the future).

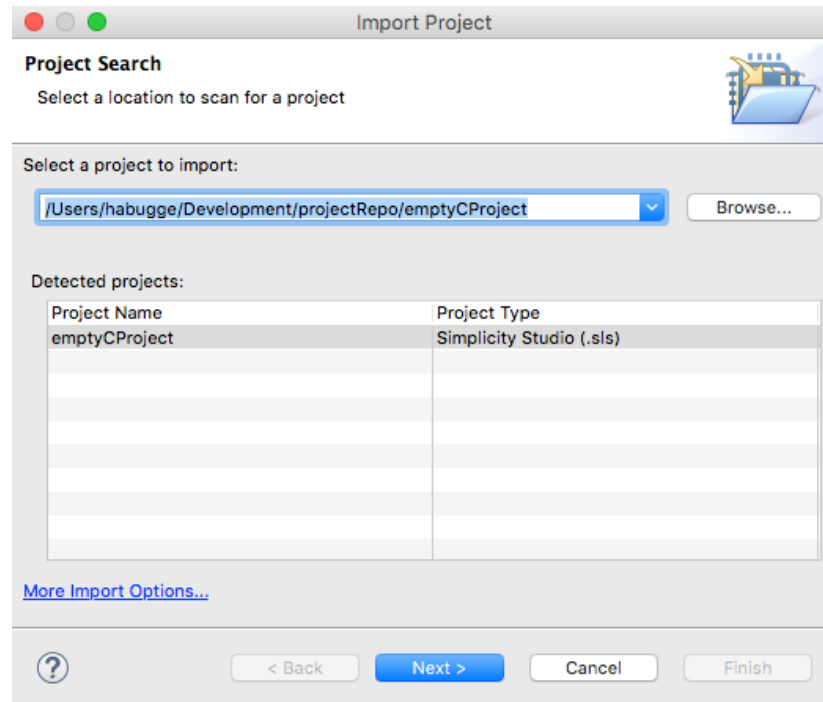
Some factors to consider when making this decision:

- Will the project be built on different operating systems? If the answer is yes, then the .isc file should be used and the engineers will need to run the AppBuilder **Generate** function after cloning the project. This is because of the differences in how the paths are handled by the different operating systems. The **Generate** function will adjust the paths based on the OS.
- What is the configuration of the development team? If there are one or two central architects that decide the radio configuration, wireless services and other features configured through AppBuilder with other engineers contributing the company-specific features of the project, then it is probably better for the architects to make the configuration changes in the .isc file and then **Generate** and push the output of the generation into the repo.
- How stable is the project? If the project is still undergoing constant changes as far as radio configuration, services, clusters, and so on that can be made by any engineer on the project, then it is better to push the .isc file and support files and have each engineer run **Generate** again after pulling a fresh update from the repo.
- Bluetooth projects mainly use AppBuilder for generating the GATT source files that describe the supported Bluetooth profile of services and characteristic attributes as well as configuring the project paths. Additional SDK source files are not pulled into the project when Appbuilder **Generate** is run. Therefore the decision on whether each engineer will run **Generate** after cloning the project or not should be based on A) Will the project be built on different operating systems and B) How many engineers are involved in defining and modifying the Bluetooth profile.

2.2 Cloning and Importing the Project

Clone the project to the preferred location on your local file system. Once the project is cloned, import the project.

1. On the **File** menu, select **Import** and click **[Browse]**. Browse to the root directory of the project, and Studio will automatically detect the project. There might be several detected projects in the **Detected projects** list. Make sure to pick the entry with Project Type **Simplicity Studio (.sls)**. Click **[Next]**.

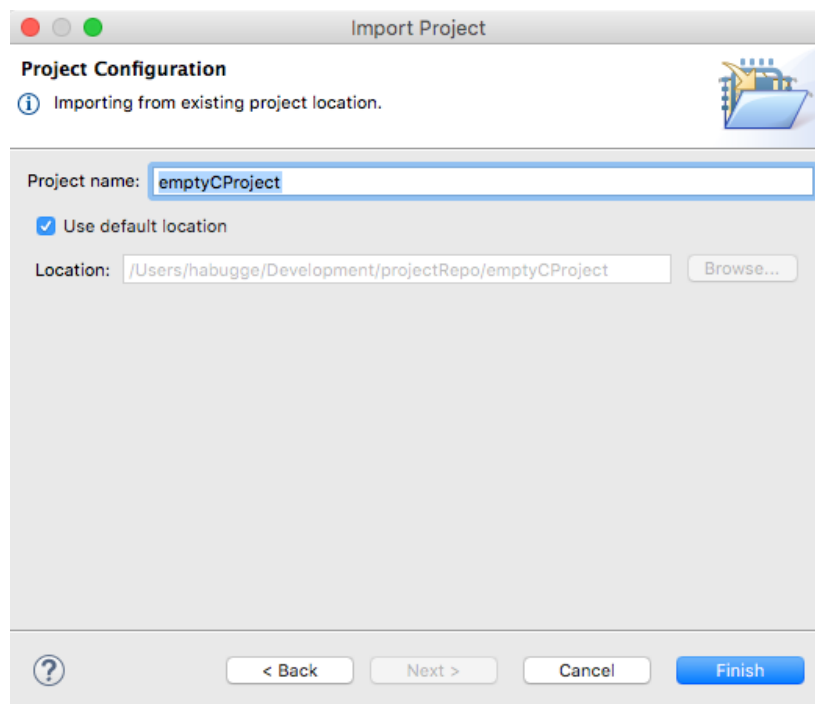


2. Make sure the SDK location is the location of the SDK added externally in the SDK section. Click **[Next]**.

SDK:


Gecko SDK Suite: Bluetooth 2.4.0.0, EmberZNet 5.10.0.0, Flex 1.2.0.0, MCU 5.2.1.0, Micrium OS 5.0.0, Thread 2.3.0.0 (v1.1.0) (/Users/habugge/Development/gecko_sdk_suite/v1.1/)

3. The default location for opening an existing project directory will be the project itself. The Information message “Importing from existing project location” indicates that the project will be imported and referenced from the existing location. If the project location is changed, the project will be copied to a different location, and no longer reference the repo. Therefore, leave the default location unchanged. Click **[Finish]**. After Import, the location can be verified by right-clicking the project .isc file and clicking **Properties**.



If you are working with MCU8 and MCU32 bit projects, go ahead and build the project. For AppBuilder/wireless projects, the rules set up in the configuration management of Simplicity Studio should be followed (see section 2.1.3.4 [App Builder](#)). These rules define if the projects should be generated or not. If the project needs to be generated, open the .isc file and click **[Generate]**.



Then build the project using the  tool.

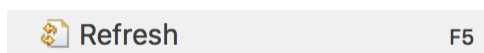
2.3 Checking in Project Changes

2.3.1 Checking in App Builder Projects

For App Builder projects the rules set up in the configuration management of Simplicity Studio should be followed (see section 2.1.3.4 [App Builder](#)). Those rules define what changes should be checked in and not.

2.4 Pulling Project Changes

Close Simplicity Studio, and pull the project changes. Reopen Studio, and the changes should be detected automatically, if not, refresh the project by right-clicking the .isc file and clicking **Refresh**.

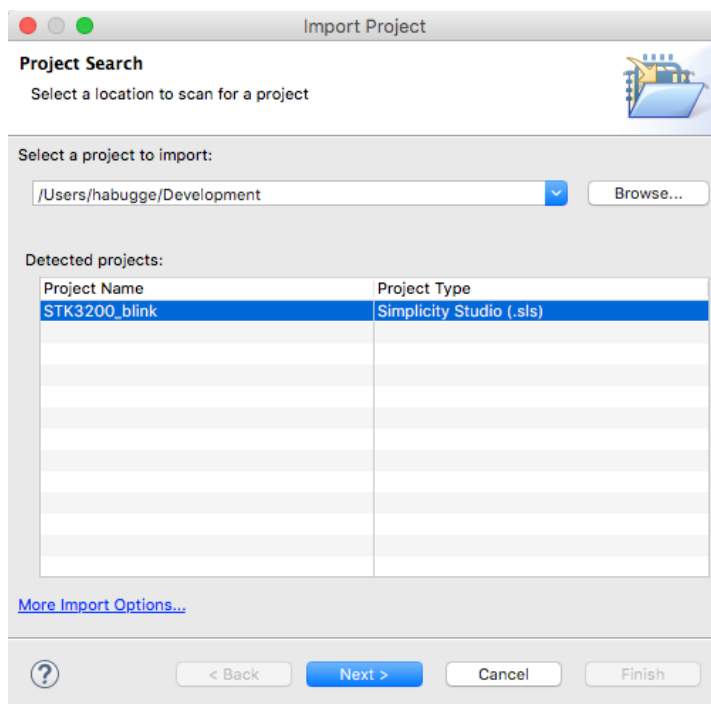


3. Import Wizard

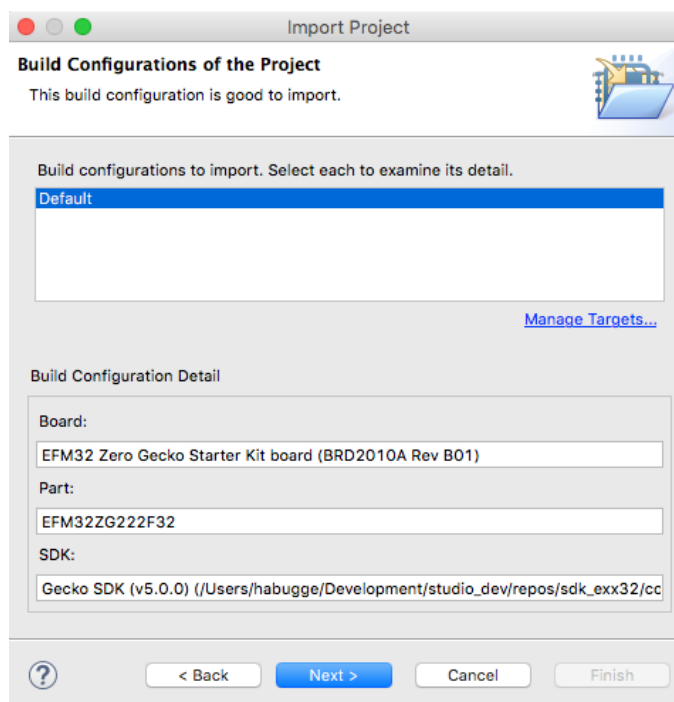
The Simplicity Studio Import Wizard supports import of the following file formats into Simplicity IDE: .cproject, .ewp, .eww, .isc, .sls, .slsproj, .uvproj, .uvprojx, and .wsp. The example demonstrates import of .sls, but the procedure is similar for the other file formats.

3.1 Import Procedure

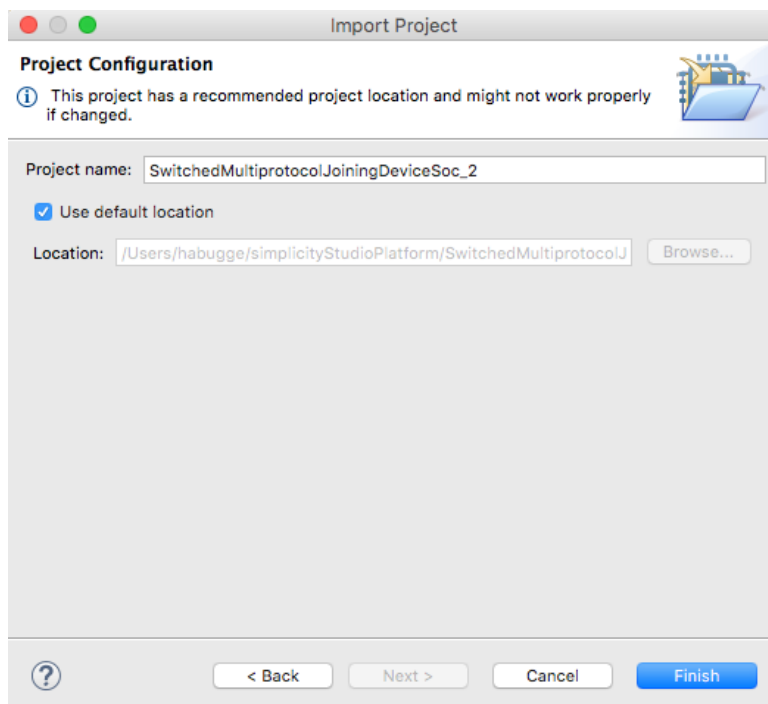
1. On the **File** menu, select **Import** and click [**Browse**]. Browse and select the directory where the project intended for import is located. A list of all the valid projects the import wizard finds in the selected directory is shown under **Detected projects**. Select the wanted project from the list, as the importer might detect several project types. Click [**Next**].



2. The build configuration dialog is displayed. If the build configuration is not resolved you can resolve it on the next page. Click [**Next**].

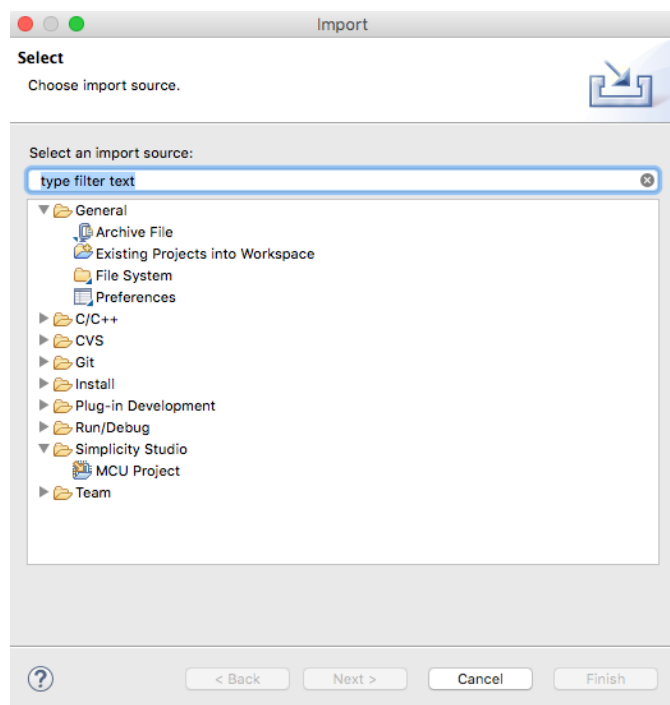


3. The Simplicity Studio workspace is the default location for projects. You can choose a different location. Note that some projects have a recommended location. The recommended location is the default location. You can change the location at your own risk but it is not recommended as the project might not work properly. Note: Changing the name of AppBuilder projects is not recommended. Click **[Finish]**.



3.2 More Import Options

To access to the Eclipse native Importer, on the **File** menu, select **Import** and click the **More import options** link. The following dialog is displayed. This functionality is native to Eclipse; we recommend using the Studio Importer.

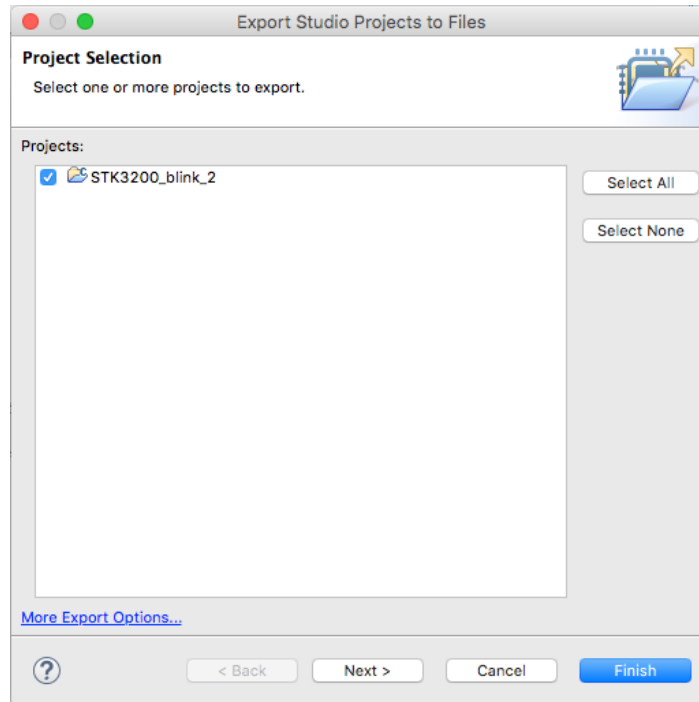


4. Export Wizard

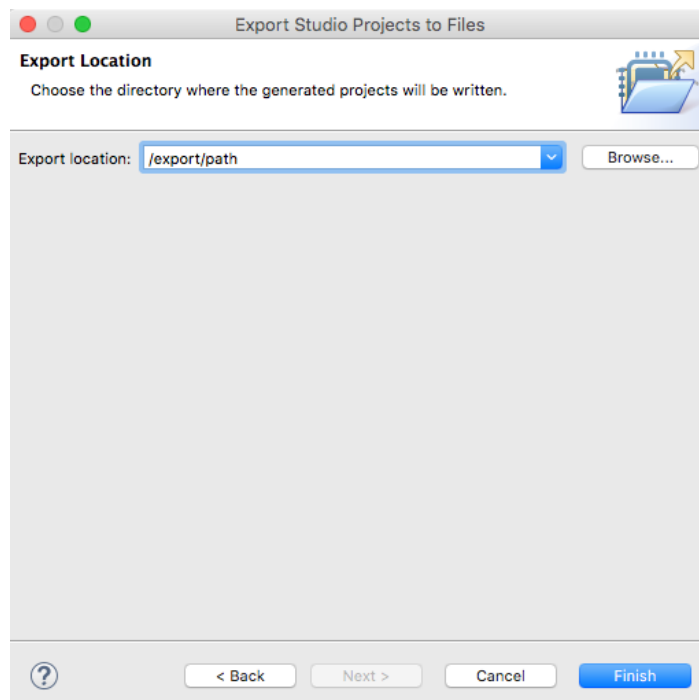
The Simplicity Studio IDE Export Wizard exports project(s) to .sls format. .sls is an archive file containing the source and library files of the project. This makes it easy to export and send an .sls project to a Silicon Labs representative or a coworker. The project references include files from the SDK, and the person receiving the project should have a compatible SDK installed.

4.1 Export Procedure

1. On the **File** menu, select **Export**. If there are multiple projects in your environment, you can choose to export several of them at the same time. Check the project(s) you want to export in the project selection list. Each of them will be exported to a separate .sls file. If you click **[Finish]** on this dialog, the files will be exported to the workspace directory (after a clean workspace/install), or the last directory exported to. To specify the export location, click **[Next]**.

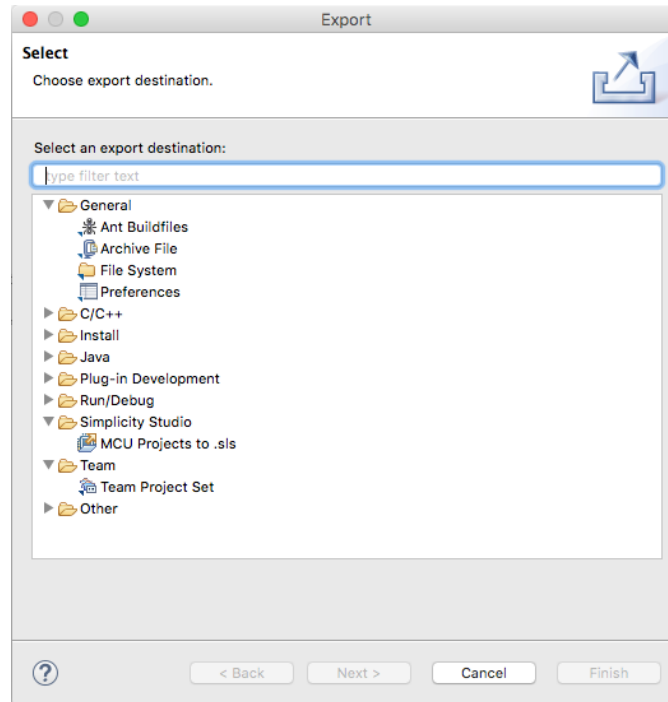


2. Click **[Browse]** and select the target directory for the export. Click **[Finish]**.



4.2 More Export Options

To access to the Eclipse native Exporter, on the **File** menu, select **Export** and click the **More export options** link. The following dialog is displayed. This functionality is native to Eclipse; we recommend using the Studio Exporter.



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



SW/HW
www.silabs.com/simplicity



Quality
www.silabs.com/quality



Support and Community
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>