AN1301: Transitioning from Zigbee EmberZNet SDK 6.x to SDK 7.x

Zigbee EmberZNet Software Development Kit (SDK) version 7.x contains a number of changes compared to SDK 6.x. Many of these changes are due to an underlying framework redesign that results in an improved developer experience within Simplicity Studio 5. Projects are now built on a component architecture instead of AppBuilder plugins. Simplicity Studio 5 includes project configuration tools that provide an enhanced level of software component discoverability, configurability, and dependency management. These include a Zigbee Configurator and a Component Editor.

KEY POINTS

- Reviews updated file structure of SDK 7.
- Describes how to create and configure a project with Simplicity Studio 5 and Zigbee SDK 7.
- Compares Project Configuration functionality in AppBuilder to Project Configurator/Zigbee Cluster Configurator.
- Reviews main differences between Simplicity Studio v4 and v5.

1 Introduction

Gecko Software Development Kit suite (GSDK) version 3 (GSDK v3.0) introduced a new underlying Gecko Platform architecture based on components. Beginning with GSDK 4.0, the Zigbee EmberZNet SDK v. 7 now uses this underlying architecture. With Simplicity Studio 5 (SSv5) and GSDK 4, developers working with Zigbee EmberZNet will benefit from the following component-based project configuration features:

- Search and filter to find and discover software components that work with the target device
- Automatically pull in all component dependencies and initialization code
- Configurable software components including peripheral initializations, drivers, middleware, and stacks
- All configuration settings in C header files for usage outside of Simplicity Studio
- Configuration validation to alert developers to errors or issues
- Easily manage all project source via git or other SCM tools
- Manages migration to future component and SDK versions
- Simplifies transitions from Silicon Labs development kits to custom hardware

Other features of the SSv5/GSDK 4.x development environment include:

- Project source management options (link to SDK sources or copy all contents to user folder)
- Graphical pin configuration
- Iterative development (configure components, edit sources, compile, debug) using SSv5 configuration tools and third-party IDEs
- GNU makefile as a build option

Other changes are specific to the Zigbee SDK. Instead of configuring project functionality through Application Builder (AppBuilder) plugins, equivalent functionality is now available through Zigbee stack or platform components and configuration tools such as Project Configurator and Zigbee Cluster Configurator.

This document summarizes the differences between the Zigbee EmberZNet 7.0 in GSDK 4.0 and earlier AppBuilder-based versions. These differences include:

- A different underlying file structure.
- Changes in the tools used to create and configure projects.
- Differences between AppBuilder and the new component-based Project Configurator
- Comparison of the new Project Configurator components with AppBuilder plugins
- Other differences including:
  - Application Framework
  - GBL Files
  - Custom Plugins and Custom Components
  - Custom Events
  - Custom Tokens
  - Custom XML

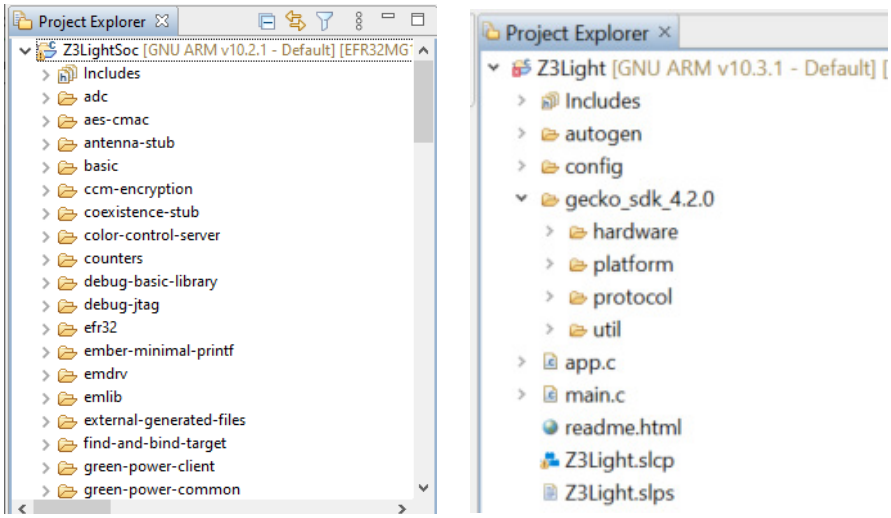
2 File Structure

SDK v7.0 is used with Simplicity Studio 5.3. Beginning with Simplicity Studio 5.3, the default installation location for the GSDK has changed. Previously it was installed under the Simplicity Studio installation. Now the default is:

- (Windows) C:\Users\<NAME>\SimplicityStudio\SDKs\gecko_sdk
- (MacOS): /Users/<NAME>/SimplicityStudio/SDKs/gecko_sdk

As before, projects are installed in a workspace or in the location specified when you create the project.

The generated file structure has been reorganized and simplified.



As for version 6.x, once you build a project the output, including the binary files, is in a compiler-specific directory, such as **GNU ARM v10.2.1 - Default**.

<projectname>.pintool contains configuration information similar to that in version 6.x <projectname>.hwconf file.

Includes:

The contents of the Includes directory are similar. In Zigbee 7.x the location of the include file depends on the choice made when creating the project (Link to Sources; Link SDK and copy project sources; Copy contents)

Autogen:

This directory contains the autogenerated project files, most of which were at the top level of the project directory in Zigbee 6.x. Never modify autogenerated files manually because manual changes are erased with every re-generation. For more details on the autogenerated files, see *UG491: Zigbee Application Framework Developer's Guide for SDK 7.x and Higher*.

Config:

Config > *zcl* > *zcl-config.zap* contains the Zigbee Cluster Configurator structure. Double-clicking this file is equivalent to opening the tool from Project Configurator's Configuration Tools tab. This directory also contains all of the configuration header files that, in Zigbee 6.x, were stored in individual directories at the top level of the project directory.

Gecko_sdk_<version>

Contains platform code and configuration files that are common across SDK projects.

.c files

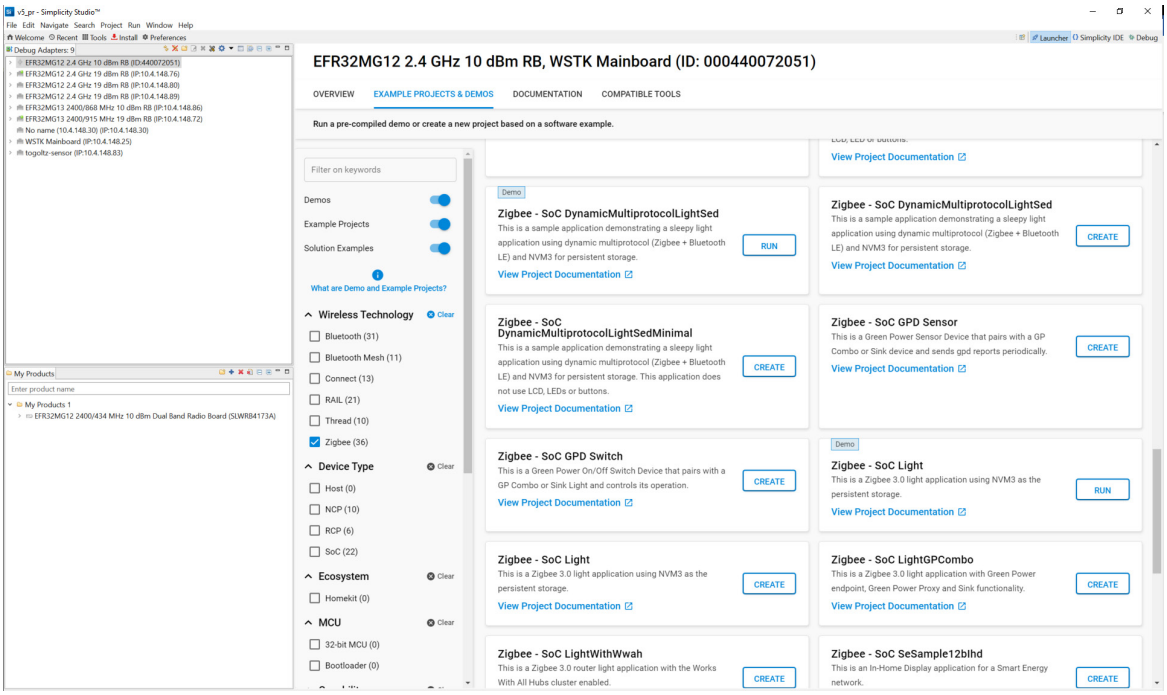
Main.c implements the `main()` function and the main loop. You can create any number of C files in the project's top-level directory. The only requirement is that an implementation of `main()` must be present in one of these files.

3 Creating and Configuring a Project

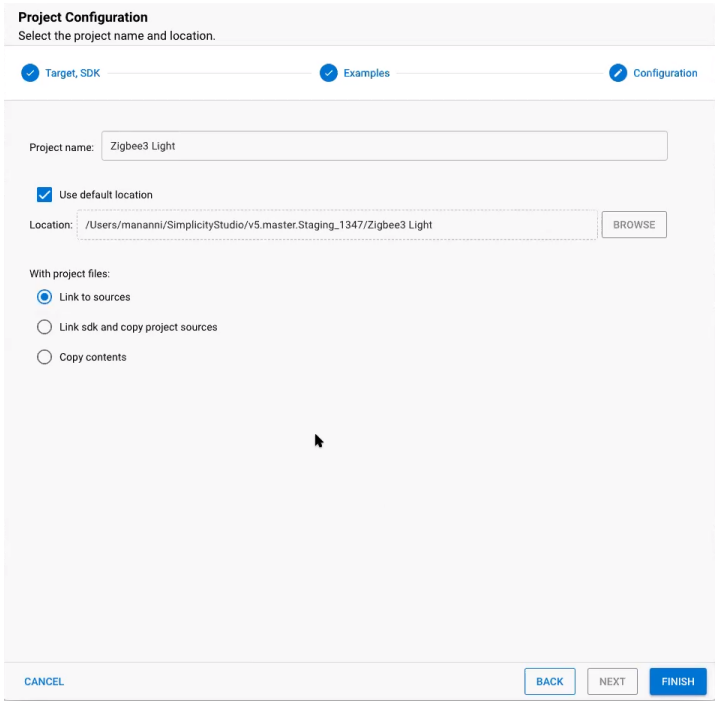
See the Simplicity Studio 5 User's Guide for details on creating a project. This section highlights the differences.

3.1 Creating a Project

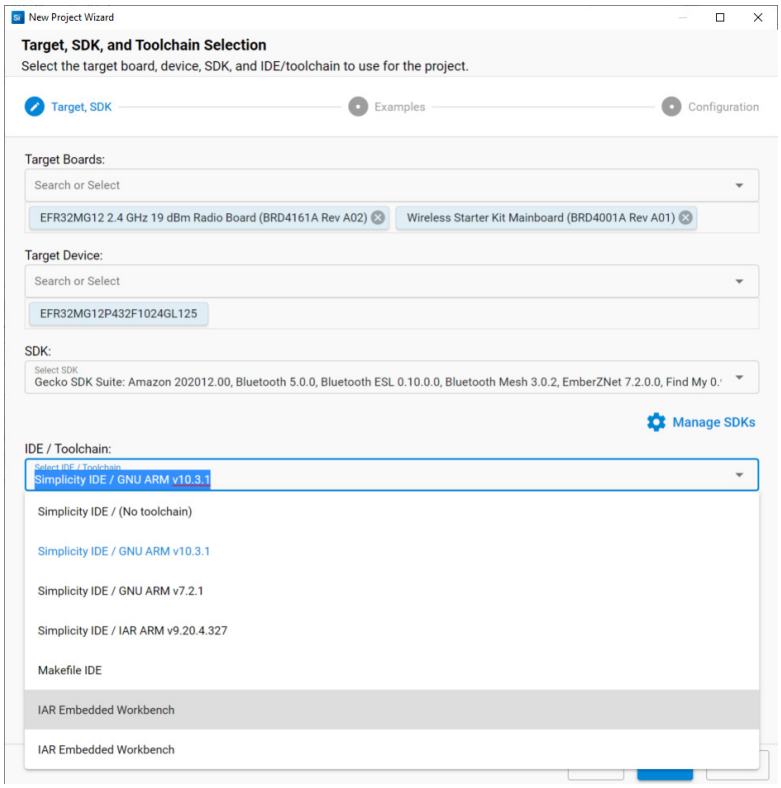
The project creation path is the same as for Zigbee SDK 6.x. On the **EXAMPLE PROJECTS & DEMOS** tab from the ssv5 Launcher page, filter on **Zigbee** projects. Click **Create** next to the target example; in this case, **Zigbee - SoC Light**.



Three new options allow you to specify how source files are referenced in the project. Note that the Project Location, which in AppBuilder is shown on the General tab, is defined here.



In Zigbee 7.x, the compiler to use is specified when you create a project and after that is difficult to change (see Simplicity Studio 5 User's Guide Tips & Tricks for a procedure to convert an existing project from GCC to IAR). If you do not want to use the default compiler (GCC unless otherwise specified), click **BACK**, and change on the first Project Configuration dialog.



3.2 Configuring a Project

Most project configuration can be done through tools in Simplicity Studio. These tools provide the functionality previously provided by AppBuilder.

- Project Configurator – the top-level project configuration tool through which you install and uninstall components and change other project parameters. It also provides access to other configuration tools.
- Component Editor – provides access to the configurable parameters of an individual component.
- Zigbee Cluster Configurator – manages the ZCL aspects of the project. The Zigbee Cluster Configurator is accessible through the Configuration Tools tab of the Project Configurator.
- Pin Tool – configures peripherals.
- Memory Editor – Graphical tool for editing memory layout of the applications in your workspace.

Once you create the project, the Project Configurator tabbed interface is presented, including:

- Overview
- Software Components
- Configuration Tools

You can always re-open the Project Configurator by double-clicking the *<projectname>.slcp* file.

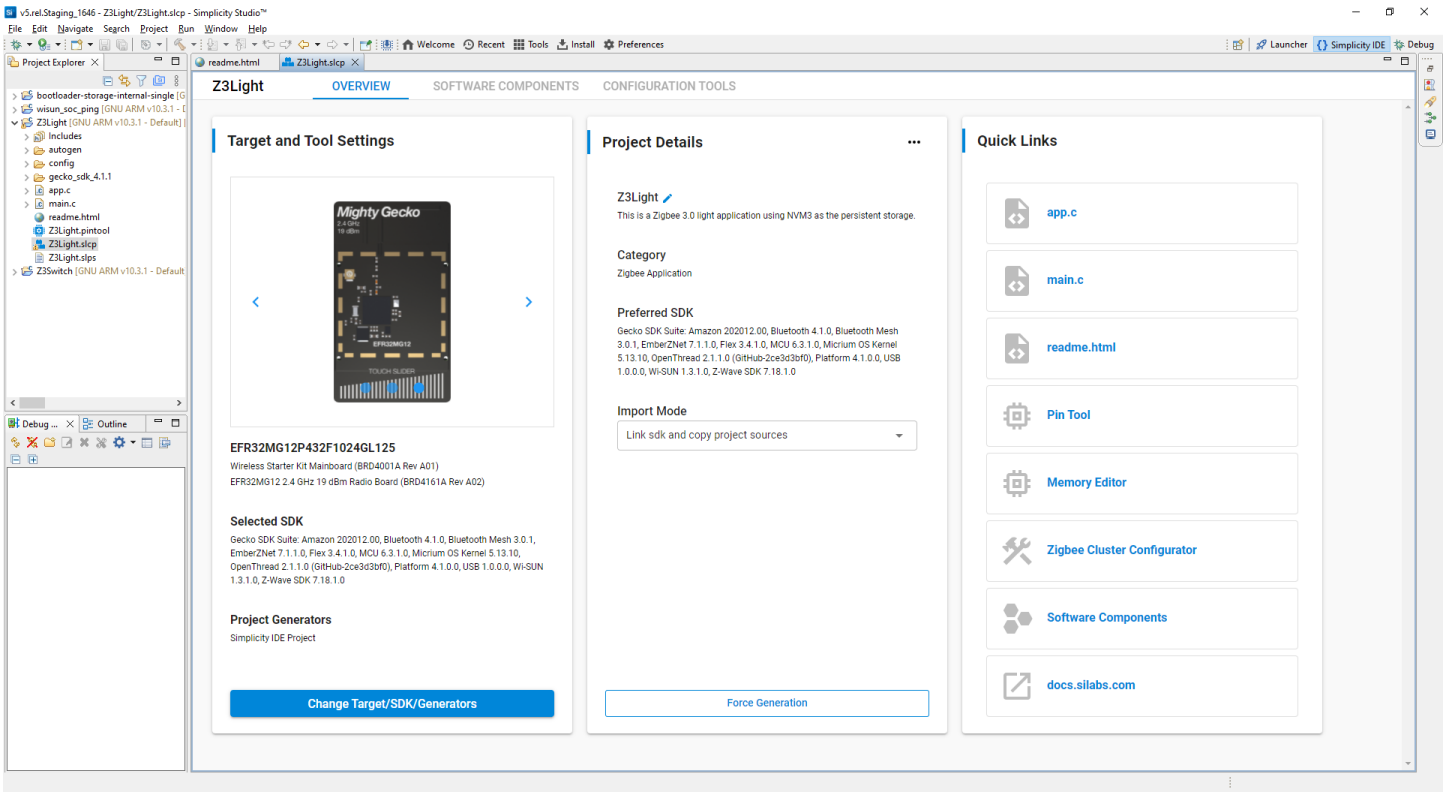
3.2.1 OVERVIEW Tab

The **OVERVIEW** tab, like AppBuilder's General tab, shows the target part information, and a description of the project.

Note that there is no **Generate** control. Changes made through Project Configurator are auto-saved and the project files are auto-generated. The Force Generation control on the Project Details card is provided for use only in the event of a system problem, where auto-generation fails.

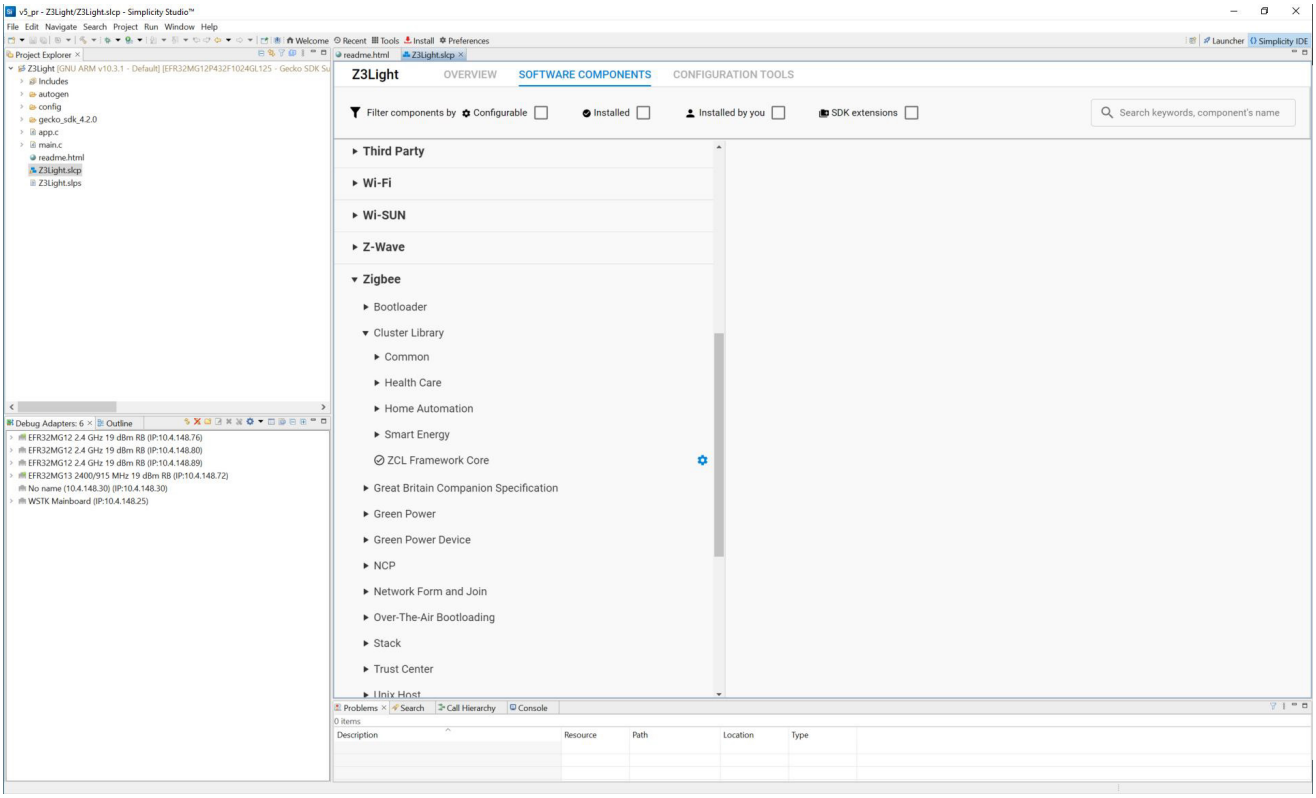
The Project Generator's interface, available through the **Change Target/SDK/Generators** control on the Target and Tool Settings card, controls generation of additional files for import into other IDEs. It does not change the compiler used to build the application image.

Note that the following figure shows the interface fully expanded. In general, you will need to scroll down to see the controls at the bottom of the cards.

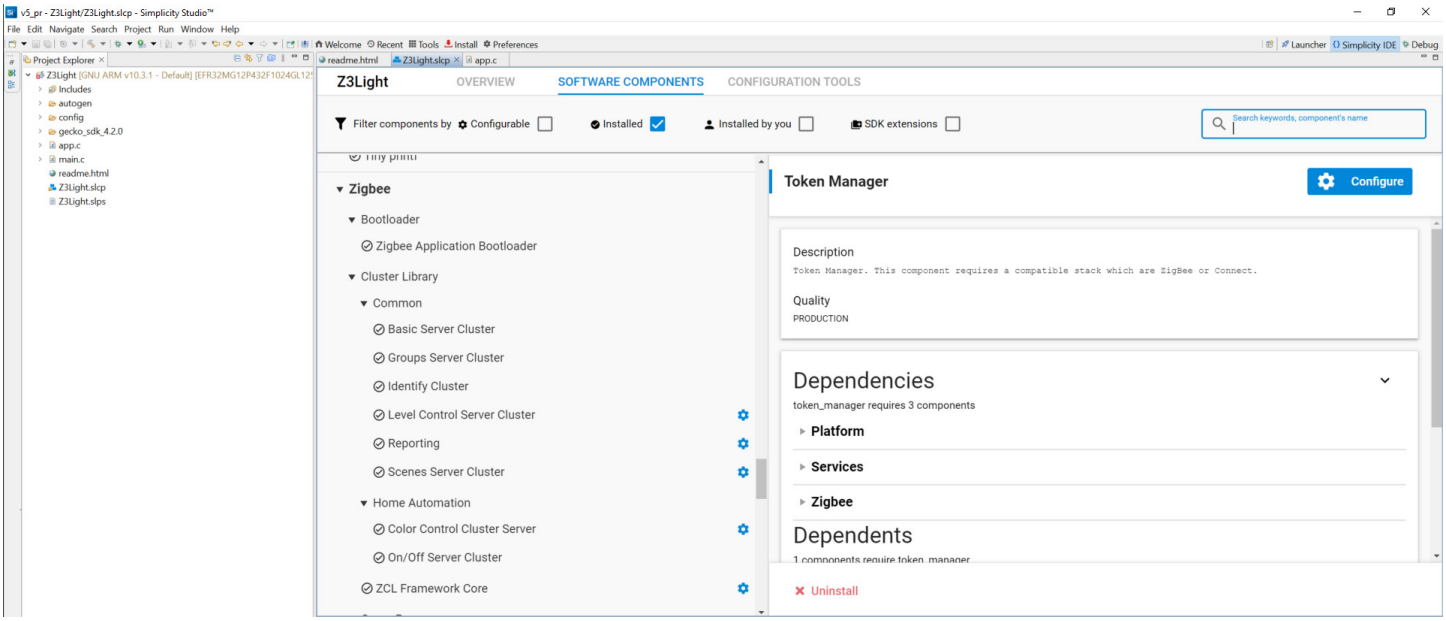


3.2.2 SOFTWARE COMPONENTS Tab

The **SOFTWARE COMPONENTS** tab shows the available components and those that are already installed in the example code.

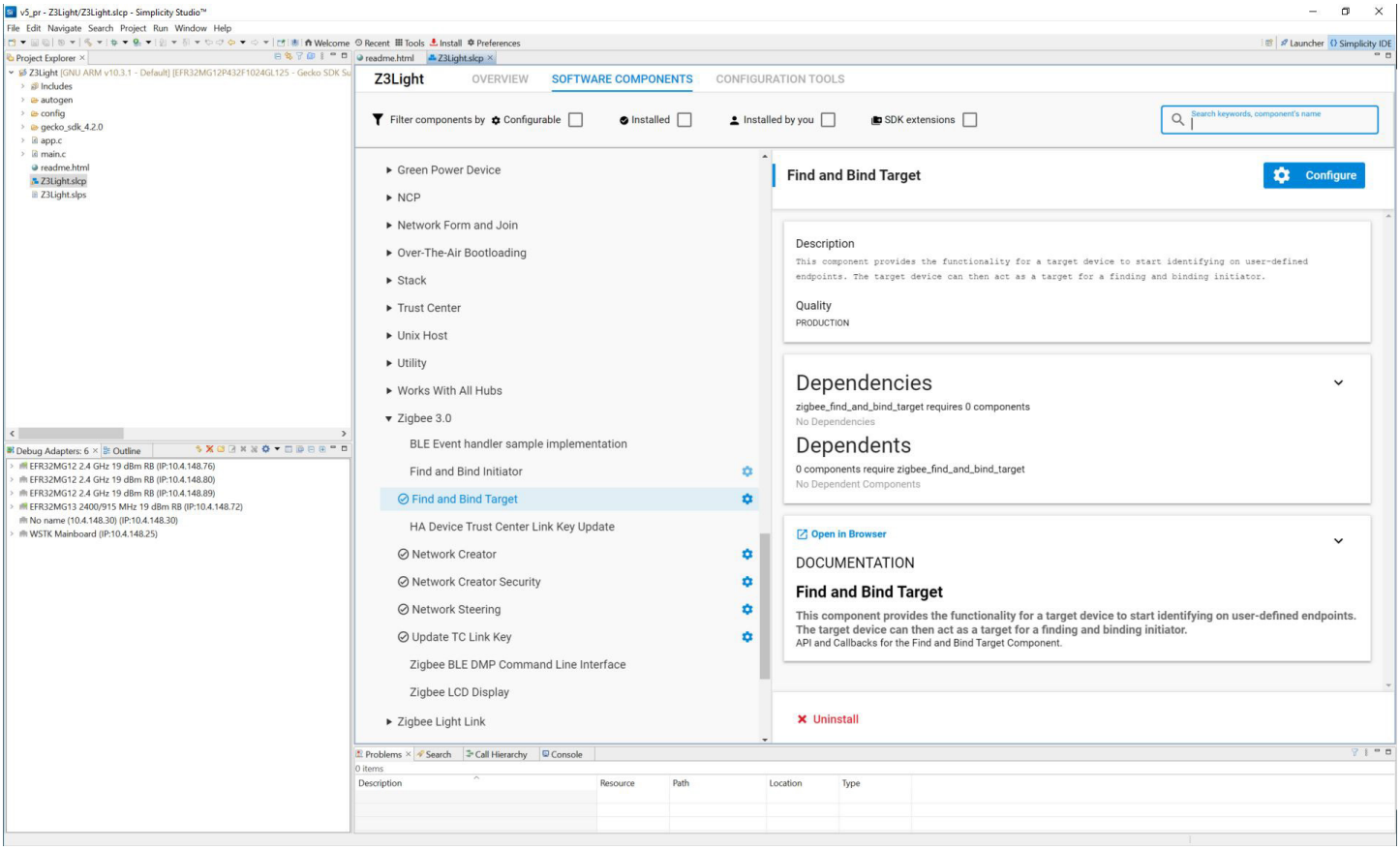


Component groupings are subject to change in SDK releases; as new components are added, the organization is updated. The most reliable way to find a specific component is to search for it. You can also filter the displayed components using the checkboxes at the top, such as Installed Components in the following figure.



Installed components, such as **Find and Bind Target** in the following figure, have a circled checkmark to the left. Configurable components have a gear icon to the right.

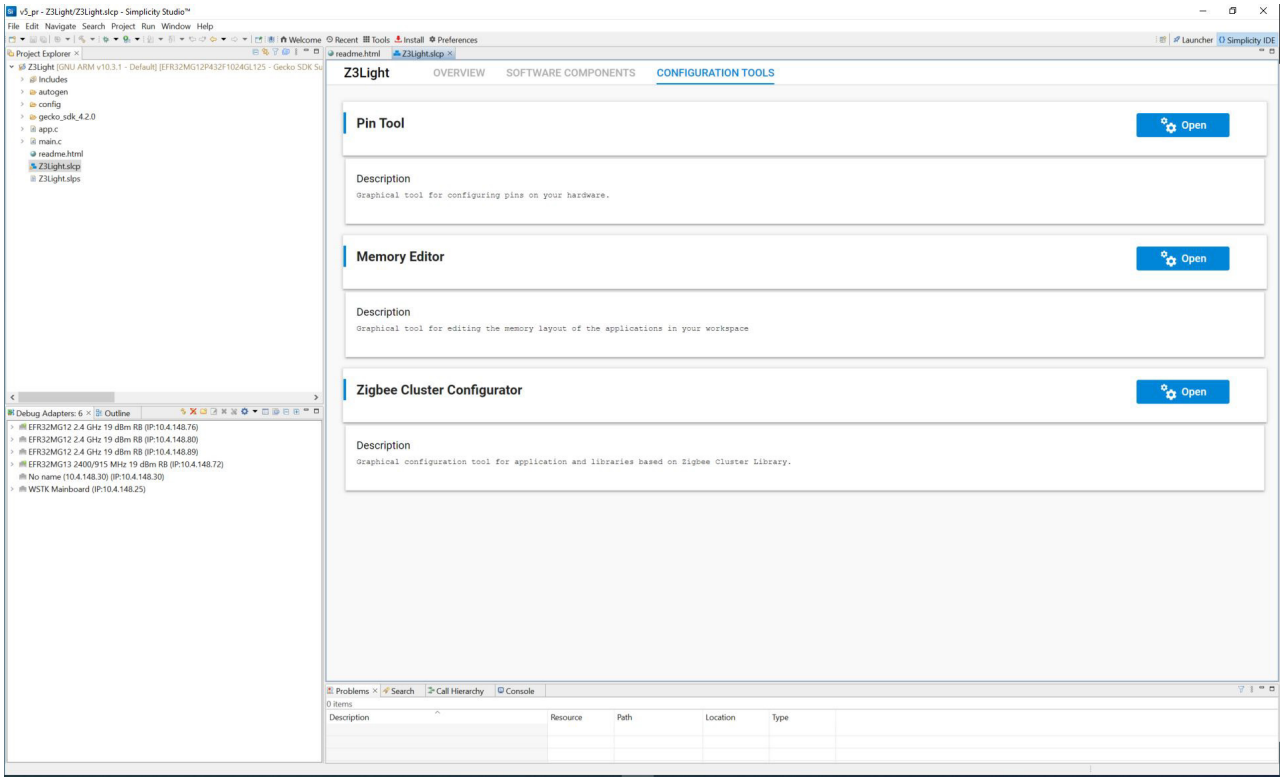
Select a component to see details about it, including its APIs and callbacks, if applicable. If it is configurable, a **Configure** control is shown in the upper right. Click that control or the gear icon next to the component name to open the Component Editor in a new tab.



A number of the Stack components correspond to the libraries that were included in AppBuilder projects. Because of the new component architecture, for the most part stub libraries are no longer required.

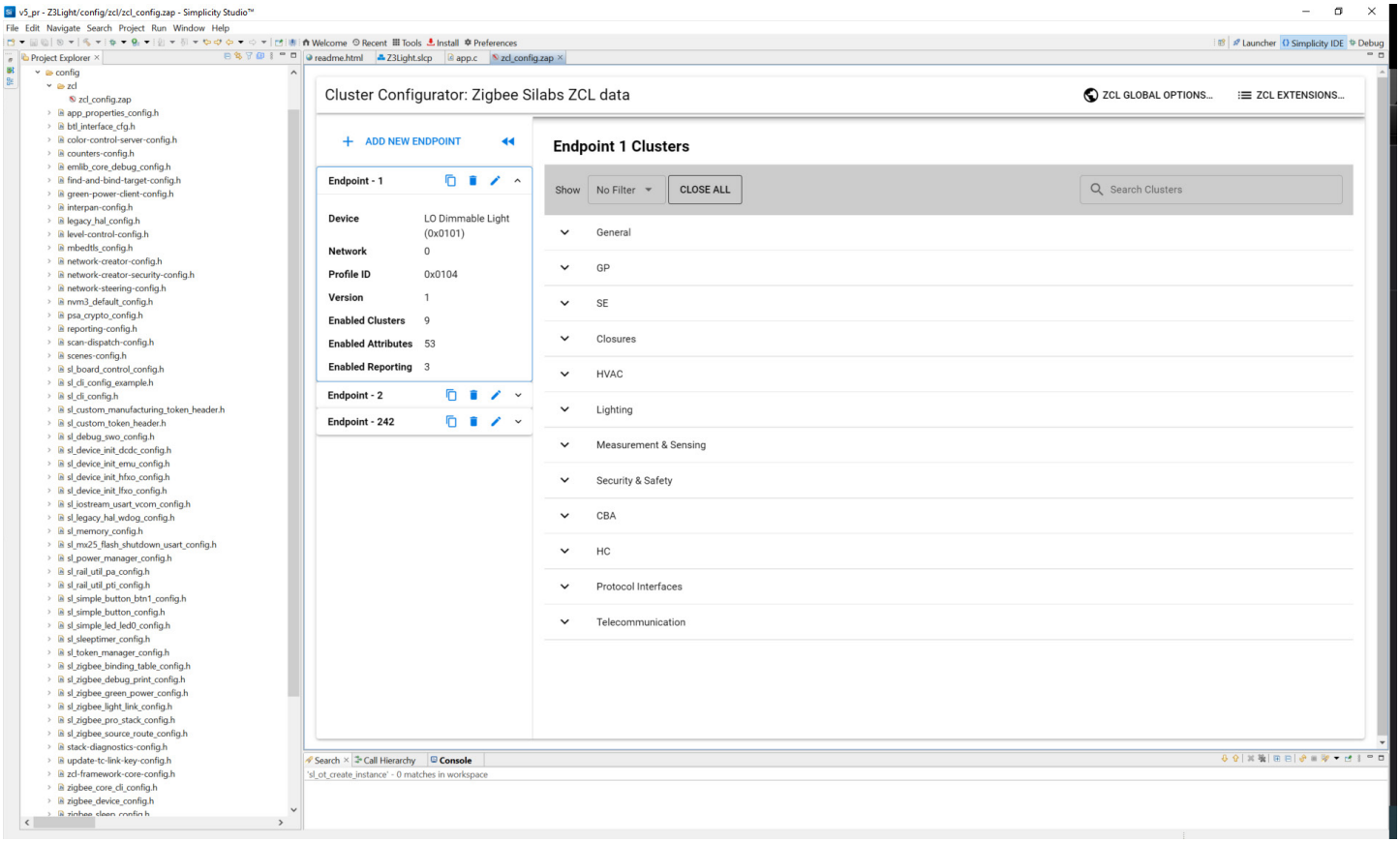
3.2.3 CONFIGURATION TOOLS Tab

The **CONFIGURATION TOOLS** tab is a quick way of opening useful tools for project development.



3.2.4 Zigbee Cluster Configurator

The Zigbee Cluster Configurator is an advanced configuration tool within Simplicity Studio that allows you to manage the Zigbee endpoints, clusters, and commands implemented by their device. The Zigbee Cluster Configurator works in concert with the Zigbee Application Framework to generate code for setting up the endpoints, clusters, attributes, and commands that make up a Zigbee application. Open the Zigbee Cluster Configurator through the Configuration Tools tab, or by double-clicking the *zcl_config.zap* file in the project config > zcl folder. A new tab opens.



See *AN1325: Zigbee Cluster Configurator User's Guide* for more information about using the Zigbee Cluster Configurator.

3.2.5 Additional Project Configuration

You can further configure the project by editing or adding files. The *main.c* file is automatically created along with the project. You can add any number of C files to the project by clicking **New** in the Project Explorer view. The files automatically become compilation units in the project and will be compiled and linked during builds.

4 Project Configuration Changes

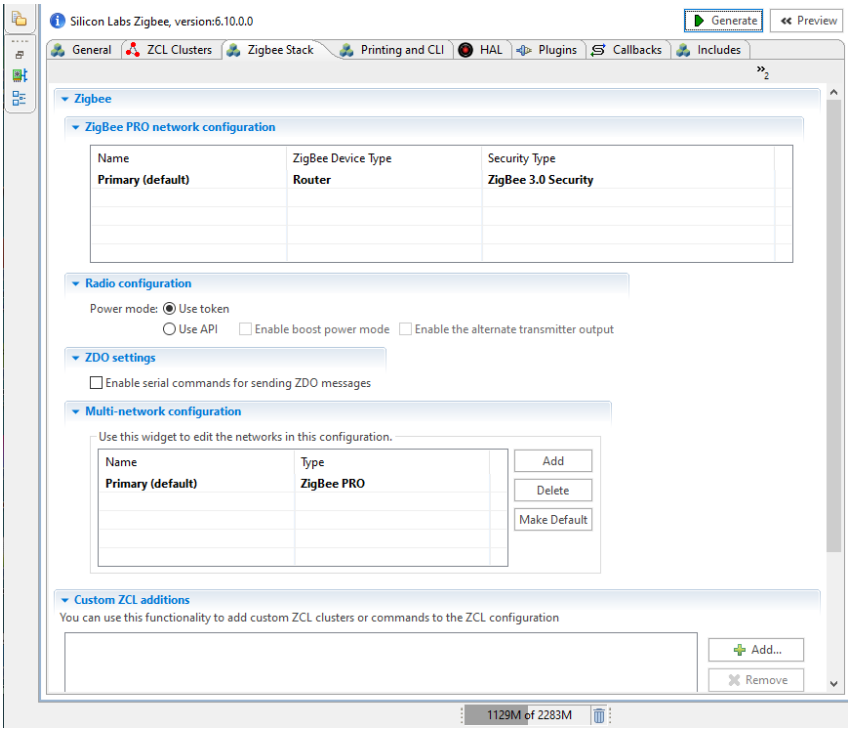
This chapter compares configuration functionality available through AppBuilder in Zigbee SDK 6.x with the equivalent functionality provided with the configuration tools in Zigbee SDK 7.x. The following sections review the AppBuilder tabs and their component equivalents. Functionality may have a direct correspondence, such the configuration formerly done through a plugin is now done through the equivalent component. In other cases, functionality may be through a custom code modification that developers made directly in project code.

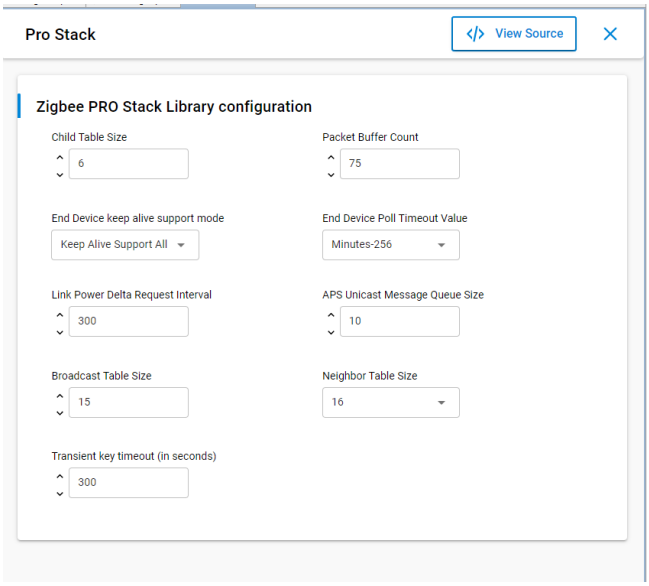
4.1 ZCL Clusters Tab

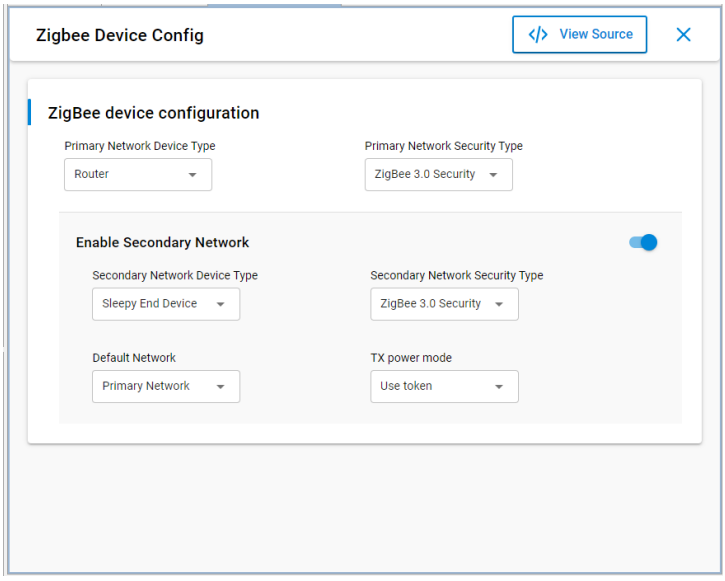
The functionality on this tab has been moved to a new configuration tool, the Zigbee Cluster Configurator, described in section 3.2.4 Zigbee Cluster Configurator.

4.2 Zigbee Stack Tab

In Zigbee 6.7.x and lower, the Stack tab was designed to configure networks, for example to add a network (but no more than two), change role, and change security settings. However, the functionality was extremely limited. Rather than reproducing this functional interface, the configuration options are included as components, **Zigbee Pro Stack Configuration** and **Zigbee Device Config**. Select a component and click **Configure** to open the Component Editor. Changes are auto-saved as you make them.





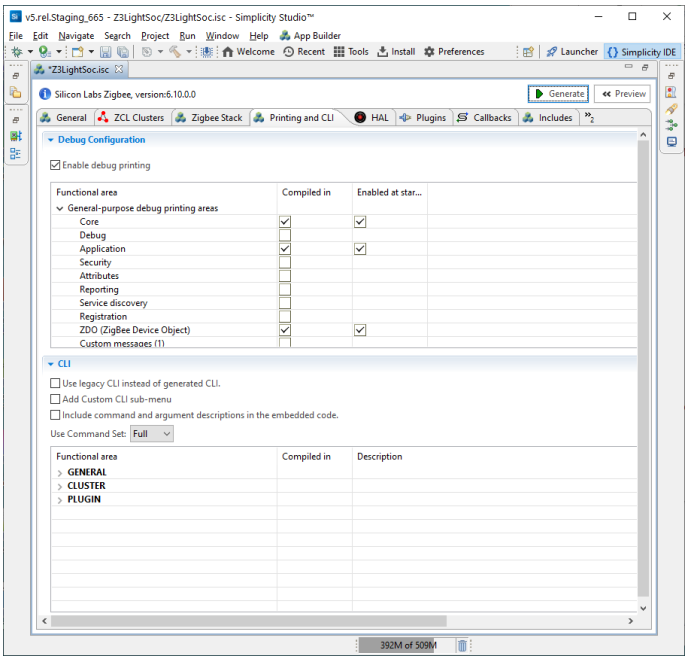


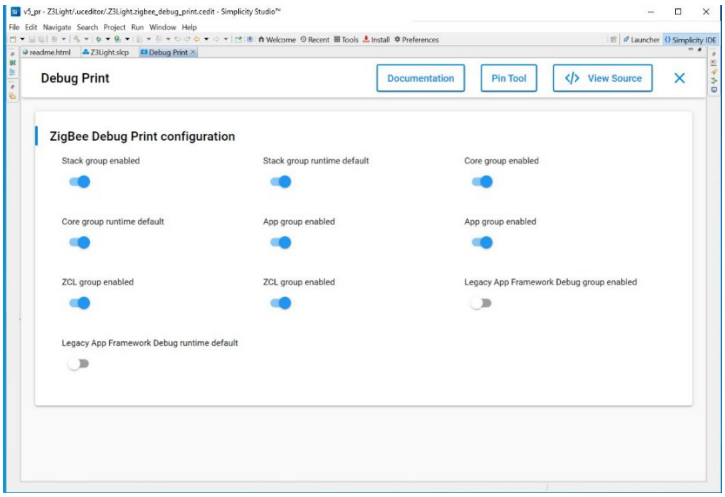
The Multi-network configuration has been replaced by the **Multi-Network** component.

Custom ZCL additions are handled through the Zigbee Cluster Configurator.

4.3 Printing and CLI tab

The AppBuilder implementation code includes a large number of APIs, some of which can be enabled at runtime. The equivalent component is Debug Print.



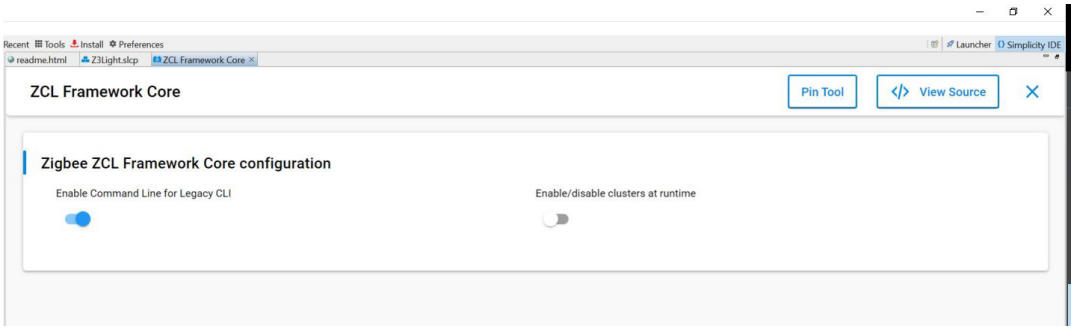


The Cluster API equivalent is the Core Group on the Debug Print component.

No exact parallel exists to the CLI section. The CLI commands are generated in a project based on the set of ZCL clusters and software components selected in the project. To use the CLI functionality in a Zigbee application install the following components:

- Zigbee CLI
- Zigbee ZCL CLI
- Zigbee core CLI

Any software component may carry with it some CLI commands. Installing a component in the project makes its CLI commands available to the application. For example, the **ZCL Framework Core** component allows you to enable or disable its CLI.



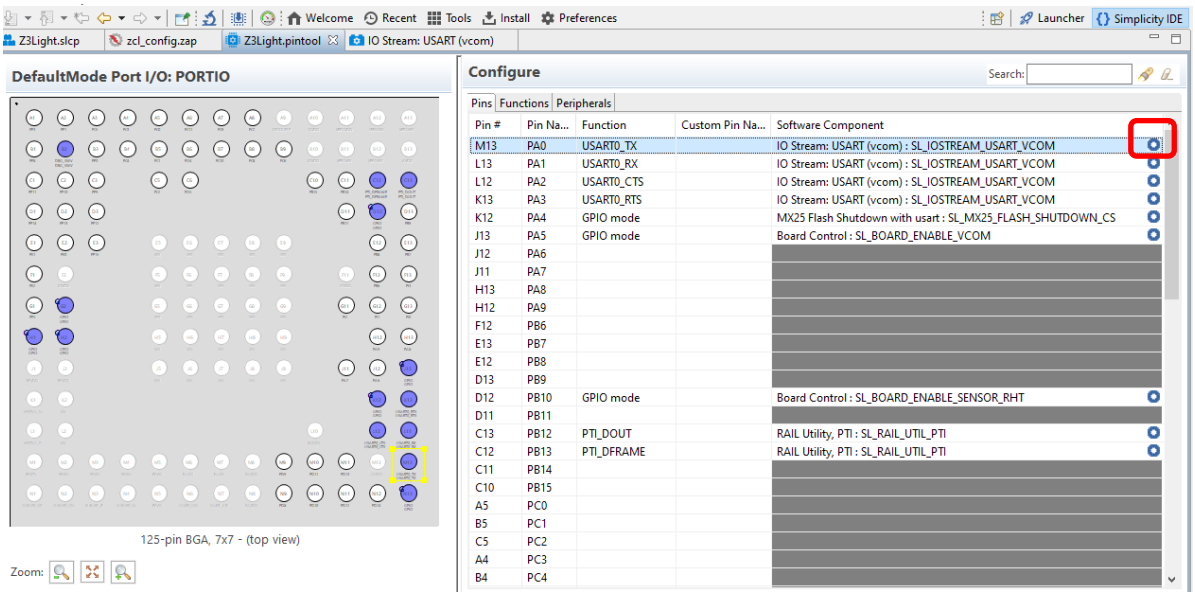
Cluster CLIs are handled through the Zigbee Cluster Configurator.

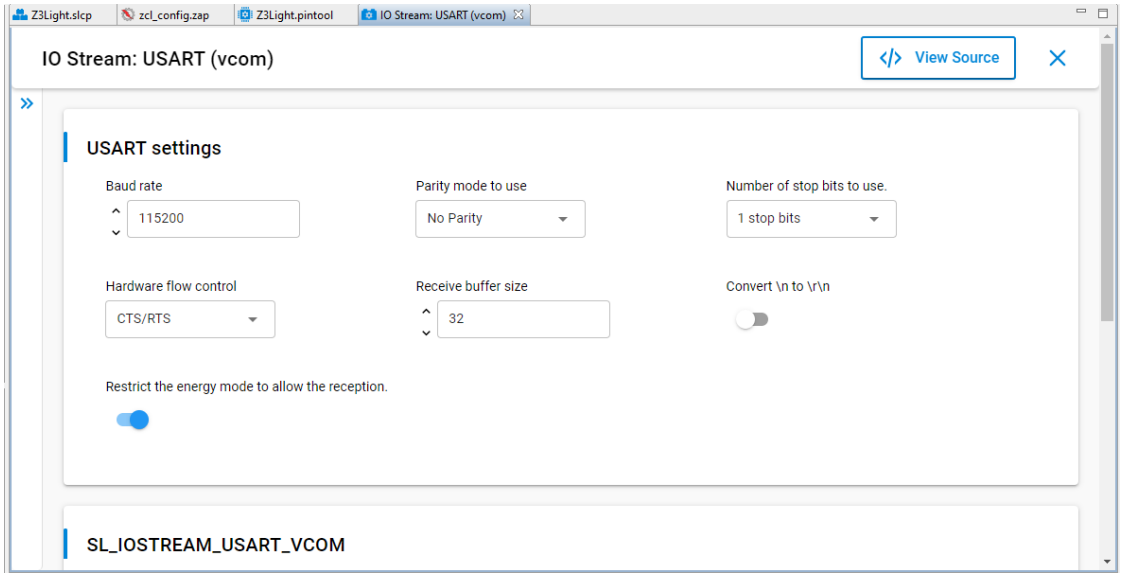
4.4 HAL Tab

Most of the content of this tab is not relevant to the new component architecture. Conceptually, the HAL (Hardware Abstraction Layer) is replaced by the underlying common Gecko platform. Platform configuration is also done through components.

The Hardware Configurator is replaced by the PIN tool and associated components. The interface varies based on the hardware platform being defined. As well as allowing you to configure pins and their peripheral assignments, you can also see any constraints imposed by components, and cross-link between the Pin Tool and the Project Configurator component view. Click the blue Configure icon next to a

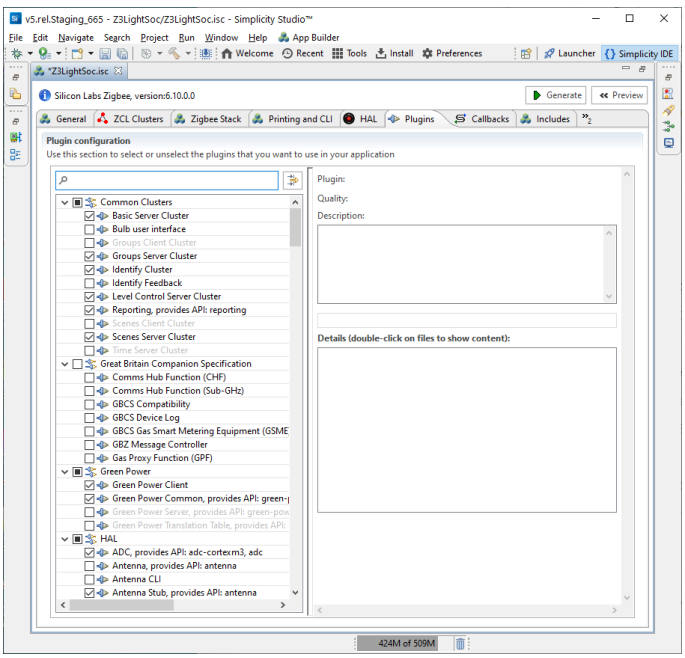
component name to open the associated component in the Component Editor. See the SSv5 User's Guide Pin Tool page and *AN1321: Configuring Peripherals for 32-Bit Devices with Zigbee 7.0 and Higher* for more information.

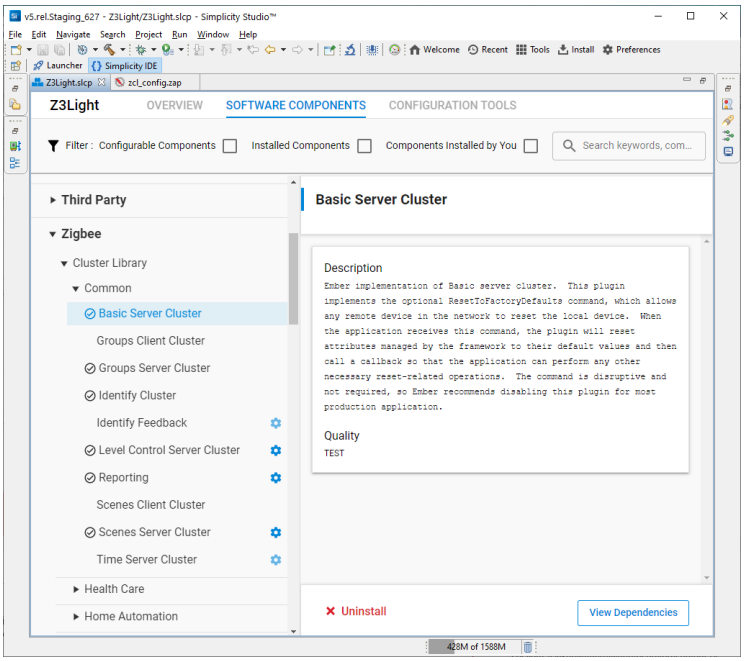




4.5 Plugins Tab

The functionality on this tab is entirely replaced by components and their configurations.





The file <GSDK location>\protocol\zigbee\esf-migration.yaml contains the individual component equivalent of individual plugins. This file can be used to determine which component should be installed when migrating a specific plugin.

Most plugins have a one-to-one equivalent to a component, but some plugins do not have an obvious equivalent component:

- dmp-ui-demo: Replaced by **UI Demo Functions** component
- dmp-ui-demo-stub: This stub is no longer required
- ember-minimal-printf: Replaced by **Legacy Printf** component
- idle-sleep: The functionality is now integrated in the **Power Manager** component
- lwip-interface: Replaced by **LwIP API** component.
- rtos-common: No longer required
- secure-ezsp-stub: This stub is no longer needed.
- simple-main: This plugin is no longer needed, as projects now provide access to the main() function for implementation.
- standard-printf-support: Replaced by **Legacy Printf** component

In general, group names remain the same.

Zigbee EmberZNet 6.x Plugin Group	Zigbee EmberZNet 7.0.0 Component Group
Common Clusters	Zigbee > Cluster Library > Common
Great Britain Companion Specification	Zigbee > Great Britain Companion Specification (Smart Energy is not fully support in Alpha1)
Green Power	Zigbee > Green Power Zigbee > Stack > Green Power
HAL	Not applicable
Health Care	Zigbee > Cluster Library > Health Care
Home Automation	Zigbee > Cluster Library > Home Automation
I/O	Platform > Utilities > Standard I/O
Kits	Platform > Board > Radio Board
Multiprotocol	Platform > Radio > Multiprotocol

Zigbee EmberZNet 6.x Plugin Group	Zigbee EmberZNet 7.0.0 Component Group
Network Form and Join	Zigbee > Network Form and Join
Password Protected CLI	CLI > CLI: Password protection
Printing	Zigbee > Utility > Debug Print
RAIL	Platform > Radio
Smart Energy	Zigbee > Cluster Library > Smart Energy
Stack Libraries	Zigbee > Stack
Standalone Bootloader	Zigbee > Utility > Standalone Bootloader
Testing	Zigbee > Utility > Stack Diagnostics Zigbee > Green Power > Green Power Test Device
Third Party	Third Party group
Trust Center	Zigbee > Trust Center
Utility	Zigbee > Utility
Zigbee 3.0	Zigbee > Zigbee 3.0
Zigbee Light Link	Zigbee > Zigbee Light Link
Zigbee OTA Bootloading	Zigbee > Over-The-Air Bootloading

4.6 Callbacks Tab

Callbacks are handled completely differently in the component architecture. In AppBuilder, for each callback, you could indicate if you want to use it or not. The callback would be put in either a callback stub or a callback file. One goal of the new structure is to eliminate the use of stubs.

In Zigbee SDK 7.x, to implement a callback add the corresponding C function code to one of the project's C files. For example, in Project Explorer view, right-click the project folder and click **New.** Add a callback implementation to this file and save it. The file automatically becomes a compilation unit in the project and the callback implementation will be compiled and linked during subsequent builds.

For every callback relevant to a project, the Zigbee Application Framework supplies a *weak* implementation that assures that the project builds successfully even if you do not explicitly implement the callback. If you do implement the callback, the C *weak functions* mechanism prevents duplicate symbol errors during linking. Depending on the callback type, the *weak* implementation may be supplied in one of the generated files or in one of the SDK files. Implementation details for the new callback framework are described in section 6 of *UG491: Zigbee Application Framework Developer's Guide for SDK 7.x and Higher*.

NOTE: Earlier SDK releases offered cluster-specific callback implementation for cluster actions, for example emberAfGroupsCluster-ServerInitCallback, emberAfOnOffClusterServerPreAttributeChangedCallback, and so on. These callback implementations should be moved to or dispatched from a strong implementation of the corresponding global callback:

- emberAfClusterInitCallback
- emberAfPostAttributeChangeCallback (including manufacturer-specific attributes)
- emberAfPreAttributeChangeCallback
- emberAfMessageSentCallback
- emberAfDefaultResponseCallback

4.7 Includes Tab

The "Additional Macros" functionality has changed. Includes are no longer together in a single header file. Instead, every component has its own header. Macros related to a specific component can be defined in the component's SLCC file.

The Token configuration functionality has been replaced by the Token Manager component. See section 5.5 Custom Tokens for more details.

Event configuration functionality is no longer in the Simplicity Studio interface, but now is done in the Zigbee stack event system, by initializing the events in code. The Z3Light_callbacks.c file provides a good example. Open the file either through the Project Explorer view for a created project, or in <Install>/protocol/zigbee/app/framework/scenarios/z3/Z3Light. The API used for events is in *protocol/zigbee/app/framework/common/zigbee_app_framework_event.h*. For more information, see section 5.4 Custom Events.

4.8 Other Options Tab

Select PHY support: Provided through Zigbee > Stack > PHY support components

Use stack protection with IAR: Install the component Platform > Utilities > IAR Stack Protection.

Plugin Service - Mbedtls security services: Provided through Platform > Security components:

5 Other Differences

5.1 Application Framework

See *UG491: Zigbee Application Framework Developer's Guide for SDK 7.x and Higher* for a complete description of working with the Application Framework.

5.2 GBL Files

Beginning with Gecko SDK Suite 3.2, Gecko Bootloader format .gbl files are no longer automatically generated when you build a project. To make a .gbl file from an .s37 file, use Simplicity Commander (see UG162: Simplicity Commander Reference Guide, the GBL Commands section).

5.3 Custom Events

Events and event handlers are no longer defined through the project configuration process, but rather are defined in the implementation.

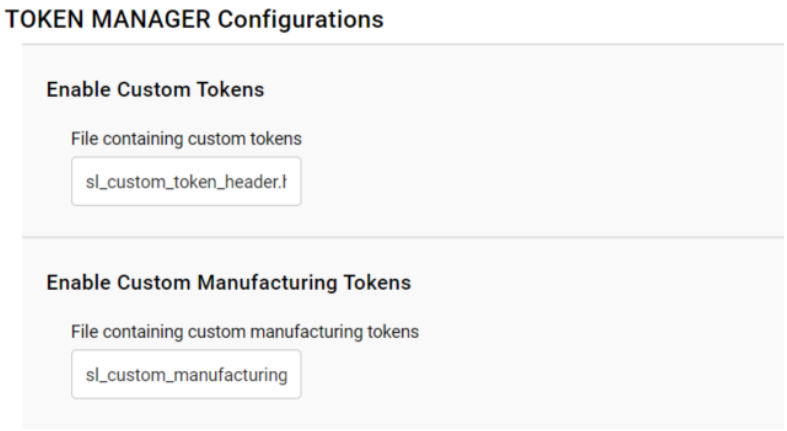
To add an Event, define a static sl_zigbee_event_t variable and an event handler. Call the `sl_zigbee_event_init()` function before the event can be set to active. The APIs have been updated in 7.x. See the following table for the new APIs for some of the common event functions in 7.x.

Zigbee EmberZNet 6.x	Zigbee EmberZNet 7.x Equivalent
emberEventControlSetInactive	sl_zigbee_event_set_active
emberEventControlSetActive	sl_zigbee_event_set_inactive
emberEventControlSetDelayMS	sl_zigbee_event_set_delay_ms

The full API used for events can be found in the file *protocol/zigbee/app/framework/common/zigbee_app_framework_event.h*.

5.4 Custom Tokens

Custom tokens are now handled through the Token Manager component. Once the component is installed, use the Component Editor to enable custom tokens and custom manufacturing tokens as desired. Specify the token header in **File containing custom tokens** for user tokens, and in **File containing custom manufacturing tokens** for manufacturing tokens. Once the files are created, they function the same as token definition files in EmberZNet 6.x. See *AN1154: Using Tokens for Non-Volatile Data Storage* for more details.



5.5 Custom XML

An interface for adding custom XML is provided in the Zigbee Cluster Configurator. See *AN1325: Zigbee Cluster Configurator User's Guide* for more information.

5.6 Custom CLI Commands

An API is provided to add custom CLI commands to an application. The function `sl_cli_command_add_command_group()` will add an sl_cli_command_group_t of commands to the application at runtime. See the Command Line Interface API for more information (https://docs.silabs.com/gecko-platform/latest/service/api/group-cli).

5.7 Custom SDK Extensions

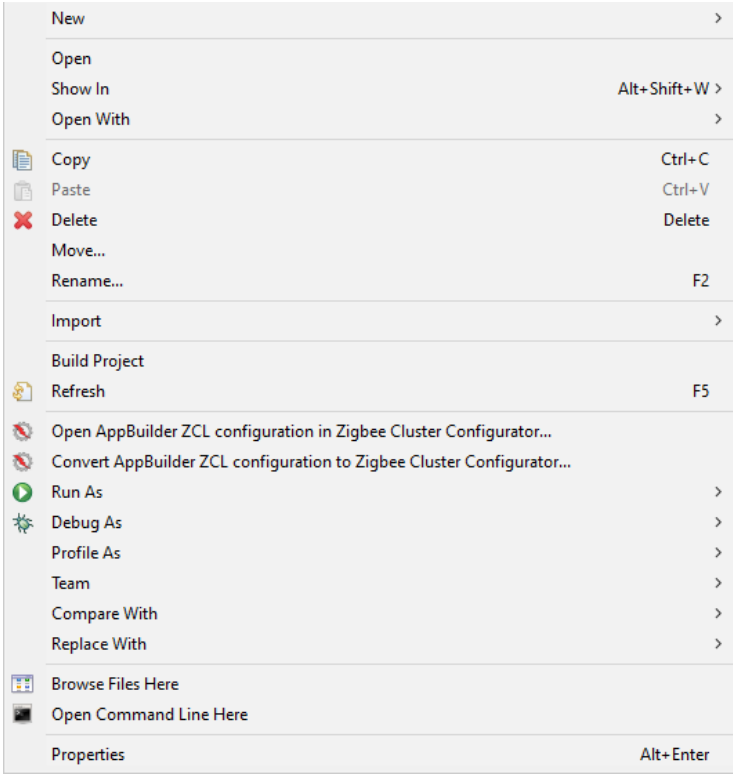
SDK extensions are a collection of components and other items that can be used to control access to certain functions or contain customer-created custom components maintained separately from the Silicon Labs SDK. More information can be found: https://docs.silabs.com/simplicity-studio-5-users-guide/latest/ss-5-users-guide-getting-started/install-sdk-extensions.

6 Project Migration

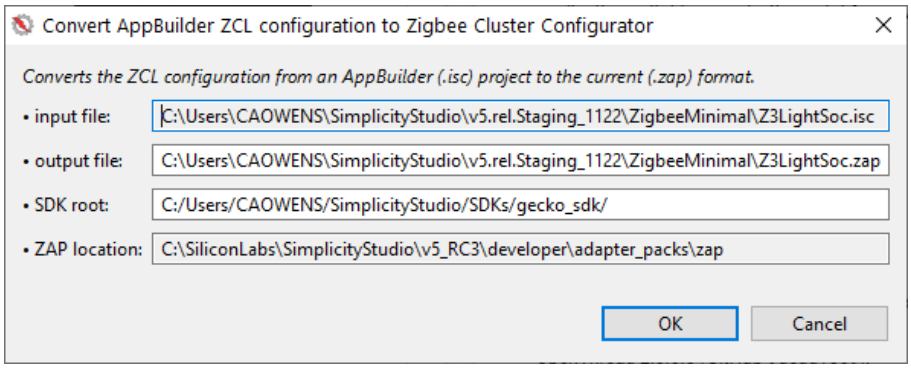
This section explains how to migrate a project originally created in Zigbee EmberZNet 6.x in the AppBuilder Framework to Zigbee EmberZNet 7.x. Projects using Application Framework V2 should be upgraded to the AppBuilder Framework before proceeding.

6.1 Migrating SoC Applications

1. Create a new **ZigbeeMinimal** project in the new SDK from the SSv5 Launcher view. If your project uses DMP, consider using one of the DMP sample apps instead (see section 6.5 Migrating DMP Applications for more info).
2. A tool has been provided to convert the ZCL configuration of the existing AppBuilder project into the ZAP format. Note that this will not carry over any custom ZCL extensions. Those must be manually imported to the new project as described in section 5.5 Custom XML. If the ISC file is not in your workspace, copy it into the folder for the new ZigbeeMinimal project. Right-click the ISC file and select **Convert AppBuilder ZCL configuration to Zigbee Cluster Configurator …**



Verify and change the information as needed.

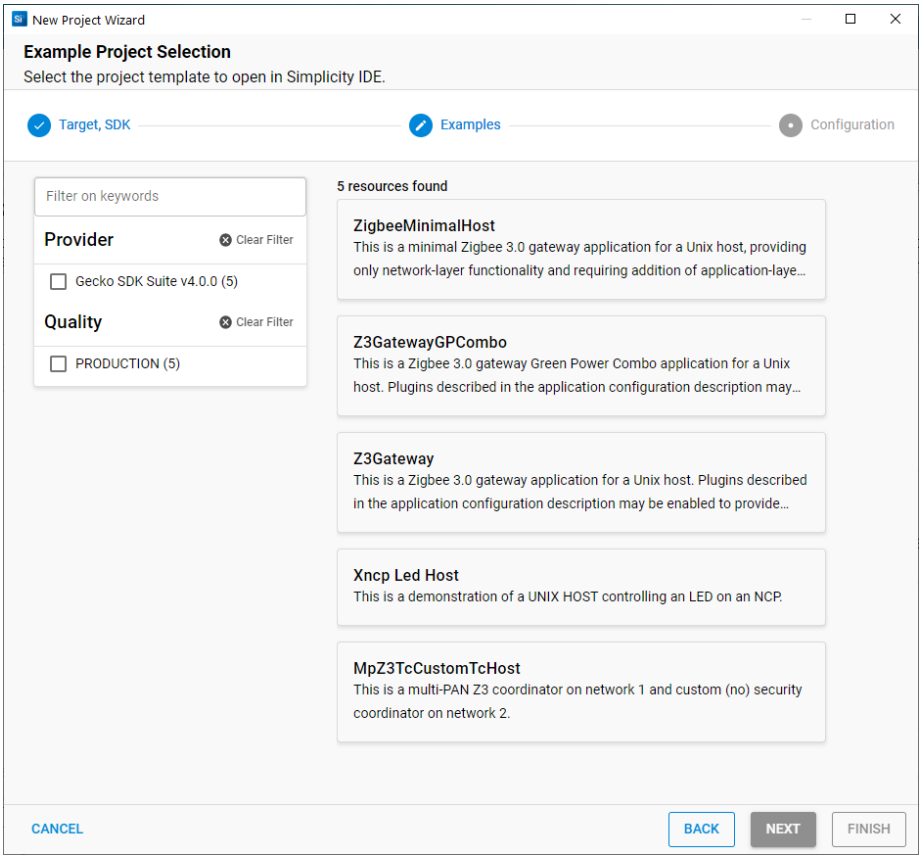


Click **OK** and a .zap file will be created in the project root directory of the project containing the .ISC. Add the created file to the config/zcl folder of the ZigbeeMinimal project.

3. Familiarize yourself with the functionality available through the Component Library. Install the components that correspond to functions in your original project. Edit any configuration parameters using the Component Editor. Some plugins may not have equivalent components. See section 4.5 Plugins Tab for more information.
4. As described in section 5.4 Custom Tokens, the Token Manager component should be used to specify the location of the custom token header.
5. As described in section 5.3 Custom Events, events and events handlers are no longer defined in the project configuration. Therefore you must manually declare the events in your code.
6. The main() function is now available in the main.c file, so the initialization code should be placed in it. The strongly defined callbacks that were in your callbacks file must now be implemented in the file app.c.
7. Review the Zigbee EmberZNet API changes in the SDK release notes and update your application code accordingly.

6.2 Migrating Host Applications

Host applications should be migrated using a similar procedure to that in section 6.1 Migrating SoC Applications. The main difference is that the starting application should be a Z3Gateway. Note that, beginning with Zigbee EmberZNet 7.0, to find the Z3Gateway sample application from the New Project Wizard or from the My Products view, set the Target Device to **Linux 32 bit** or **Linux 64 bit**, depending on your architecture.



6.3 Migrating NCP Applications

NCP application should be migrated using a similar procedure to that in section 6.1 Migrating SoC Applications. The main difference is that the starting application should be an NCP-UART-HW or NCP-SPI, depending on the configuration used. ZCL cluster configuration does not apply to an NCP project, and as such no .zap file needs to be generated.

6.4 Migrating Custom Plugins to Components

Components are defined in .slcc files, which contain the configuration of the component, paths to source and header files, and so on. The app framework components are defined in the folder *protocol/zigbee/app/framework/component*. The slcc files for the stack components are defined in the *protocol/zigbee/component* folder.

The actual component implementation is defined in a folder named for the component in the SDK, at the path **protocol/zigbee/app/framework/plugin**. The component folder contains a folder named "config", which will store a header file containing the configurable elements. These configurable elements in the header are the same configurable elements that you see in the Component Editor.

In order to convert a custom plugin into a component, the plugin.properties file must be manually separated into two separate files, one .slcc file containing the component definition, and a C header file containing the component configuration. This component should be imported into the project as a Custom SDK Extension by using a .slce file. The implementation should not change except for adding the include of the component configuration file as well as updating any APIs that were changed with the change to SDK 7.x.

The configuration file uses a standard CMSIS format, a reference document can be found at https://docs.silabs.com/cmsis/5.3.0/pack/configWizard

If your component is also dependent on a custom ZCL cluster, you'll need to update the **cluster-to-component-dependencies.json** files to add the dependency.

Silicon Labs Configurator (SLC) is a metadata specification for Silicon Labs SDKs. It also describes methods of creating and configuring embedded software projects for Silicon Labs IoT devices using this metadata. The *Silicon Labs Configurator (SLC) Specification* (https://siliconlabs.github.io/slc-specification/) is a helpful reference when defining software components. SLC also supports defining SDK extensions, or collections of components that can extend Silicon Labs SDK functionality with your custom functions. The SLC-CLI is a command-line tool that allows you to work with projects, components, and extensions outside of Simplicity Studio. See *UG520: Software Project Generation and Configuration with SLC-CLI* for more information

6.5 Migrating DMP Applications

6.5.1 Architectural Changes

Before the migration to Project Configurator, a lot of DMP configuration and functionality was handled by AppBuilder via Zigbee plugins dedicated to handling non-Zigbee requirements, for example:

- Other protocol stacks (for example, configuration of Bluetooth LE parameters in the BLE plugin)
- Platform settings (configuration of RTOS parameters in the RTOS-common / micrium plugins)

With the move to Project Configurator, all aspects of project configuration are handled by the Project Configurator, or one of the Advanced Configurator extensions. This means that several plugins that were maintained to handle non-Zigbee functionality have been superseded by configuration components within their respective protocol. As a result, these superseded plugins have been deprecated and are not ported to software components. Moving these components out of the Zigbee application domain helps to enforce a separation of concerns, and results in a more modular code base and configurations that are easier to transition from project to project.

Other changes include changes to how the Zigbee and Bluetooth pieces of the application are implemented in project code. Previously the Bluetooth protocol was "hosted" within the Zigbee application, meaning the Zigbee application oversaw instantiating Bluetooth tasks and implementing a handful of Bluetooth callbacks. Consistent with the concept of separation of concerns, Bluetooth events are handled separately, and the application and RTOS kernel are responsible for initializing both Zigbee and Bluetooth tasks.

6.5.2 Suggested Approach

As a result of the DMP-specific changes mentioned above, automatic migration of AppBuilder ISC files to SLCP files is not feasible. This means that DMP projects must be manually migrated to the Project Configurator. The recommended approach for proceeding with this migration is to start a new Project Configurator project, with one of the existing DMP sample-applications.

For best results, the same BLE API and component versions should be used in the migration. From that point, application-specific functionality can be ported using the guidance in this document. Keep in mind, where previously all application callbacks were implemented in a single file, they are now broken up by protocol domain, where Zigbee callbacks and functionality are added directly to the main *app.c* file, and the Bluetooth functionality is added to *sl_ble_event_handler.c*. Note that not all applications can seamlessly migrate to the new architecture. Some experimentation may be required during the early stages of migration.

6.5.3 Other Concerns

While many parts of this migration may be achieved by copying and pasting code from one application into another, there are a few areas where special attention must be paid.

- Bluetooth callbacks should be integrated into the main BLE event handler mechanism in *sl_ble_event_handler.c.*
- Use caution in situations where data and control flow may cross over between the protocol domains, as they may lead to thread-safety concerns.
- Migrating to a DMP application may require modifying platform values, such as stack size. Configure the RTOS appropriately via CMSIS options.
- Make sure all BLE APIs are up to date with the latest versions.

Simplicity Studio

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

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com