Ember AppBuilder User's Guide

Version 3.3 Build 1985

Table of Contents

Table of Contents

Table of Contents

1 Introduction

Document Number: 120-4035-000

The Simplicity Studio AppBuilder is a graphical tool used to create configuration and build files for the Ember Application Framework. The configuration files created by Simplicity Studio AppBuilder indicate which components and configuration you would like for your compiled binary image. By using Simplicity Studio AppBuilder along with the Ember application framework, you can quickly create a ZigBee-compliant application that includes all of the required functionality for ZigBee compliance testing.

Simplicity Studio AppBuilder is component of Ember Desktop and provides a graphical user interface for creating a ZigBee application. It allows you to choose the clusters you want for your application, and to generate the configuration files required to build your application.

Simplicity Studio AppBuilder is intimately linked with the Ember application framework. Simplicity Studio AppBuilder documentation does not include documentation for the application framework itself. Documentation for the application framework is included in the stack release for the framework. Application framework documentation includes a deeper explanation of the application framework architecture, the callback interface, and the application framework API.

1.1 The Goals of Simplicity Studio AppBuilder

Simplicity Studio AppBuilder is intended to meet these goals:

- Allows Silicon Labs to ship ZigBee-compliant sample applications with EmberZNet PRO for the EM250, EM260, and EM35x platforms.
- Enables rapid development and decreases your time-to-market by providing reference implementations of standard application subsystems.
- Helps you configure stack and HAL settings for your ZigBee applications.
- Provides clean separation between your application code and the application framework's implementation of the ZigBee Cluster Library. This allows you to focus on application-specific items, without worrying about ZigBee compliance issues.

2 How To Use Simplicity Studio AppBuilder

2.1 What is Simplicity Studio AppBuilder?

Simplicity Studio AppBuilder is a configuration file generator. It must be used in conjunction with the Ember application framework, which is shipped as part of the EmberZNet PRO stack. All of the code that will be compiled into your binary image is included in the stack distribution. Simplicity Studio AppBuilder creates configuration and build files that tell the Ember application framework which portions of the code to include in, and which to exclude from, your compiled binary image.

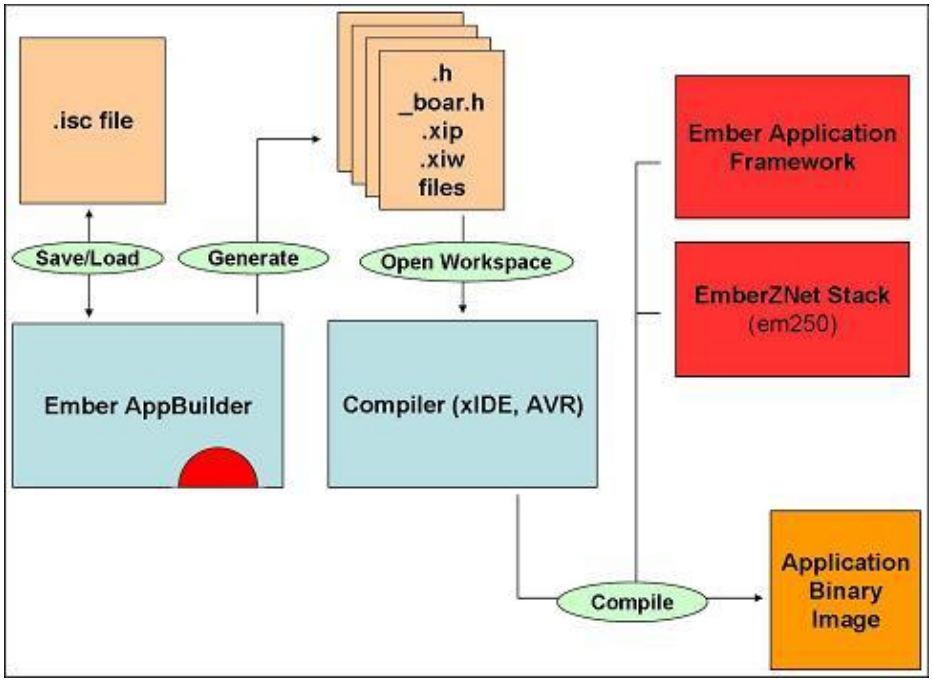
2.1.1 Simplicity Studio AppBuilder is NOT a source-code generator

With the exception of a few header (.h) files, Simplicity Studio AppBuilder does not generate the C source code for your application. All of the source code that ultimately will be included in your binary image is provided in the Ember application framework included with the EmberZNet PRO stack distribution.

2.2 How does Simplicity Studio AppBuilder work?

Simplicity Studio AppBuilder generates configuration files that you use to pick and choose the appropriate code for your application from the source code in the Ember application framework. The Ember Application Framework contains all of the source code you need to create any type of ZigBee-compliant device.

After generating the configuration files, you output them into the installation directory of your EmberZNet PRO stack. Then all you need to do is add your own application-specific code and compile the ZigBee-compliant binary image for your device. The Simplicity Studio AppBuilder generates project files for your compiler toolchain. You have the option of either compiling your application inside Simplicity Studio or loading the generated project file into the compiler of your choice and building there.



2.3 How do you create a ZigBee-compliant application?

Use Simplicity Studio AppBuilder for the first two of these steps in creating your application.

2.3.1 Step 1: Create your application configuration using Simplicity Studio AppBuilder

Open Simplicity Studio AppBuilder and create a new configuration for your application. You create your application by specifying options in these tabs:

- **General Application Configuration**
- **ZCL General Configuration**
- **ZCL Cluster Configuration**
- **Stack configuration**
- **RF4CE configuration**
- **Printing and Command Line Interface (CLI)**
- **HAL configuration**
- **Plugins**
- **Callback configuration**
- **Includes**

Each Simplicity Studio AppBuilder configuration file represents a single ZigBee device, and you use the file to generate the header and build files for a single binary image.

2.3.2 Step 2: Generate the header and build files for your application using Simplicity Studio AppBuilder

After you have chosen all of the appropriate options for your desired device, click the **Generate** button to create the header and build files for that device. This action creates files and places them in the directory you specify.

2.3.3 Step 3: Add your own application-specific code

Once you have specified the components to include in your application, you can add device-specific logic to your application.

2.3.4 Step 4: Compile your application

When you have completed steps 1-3, you are ready to compile your binary image. Compile the binary image for your ZigBee application.

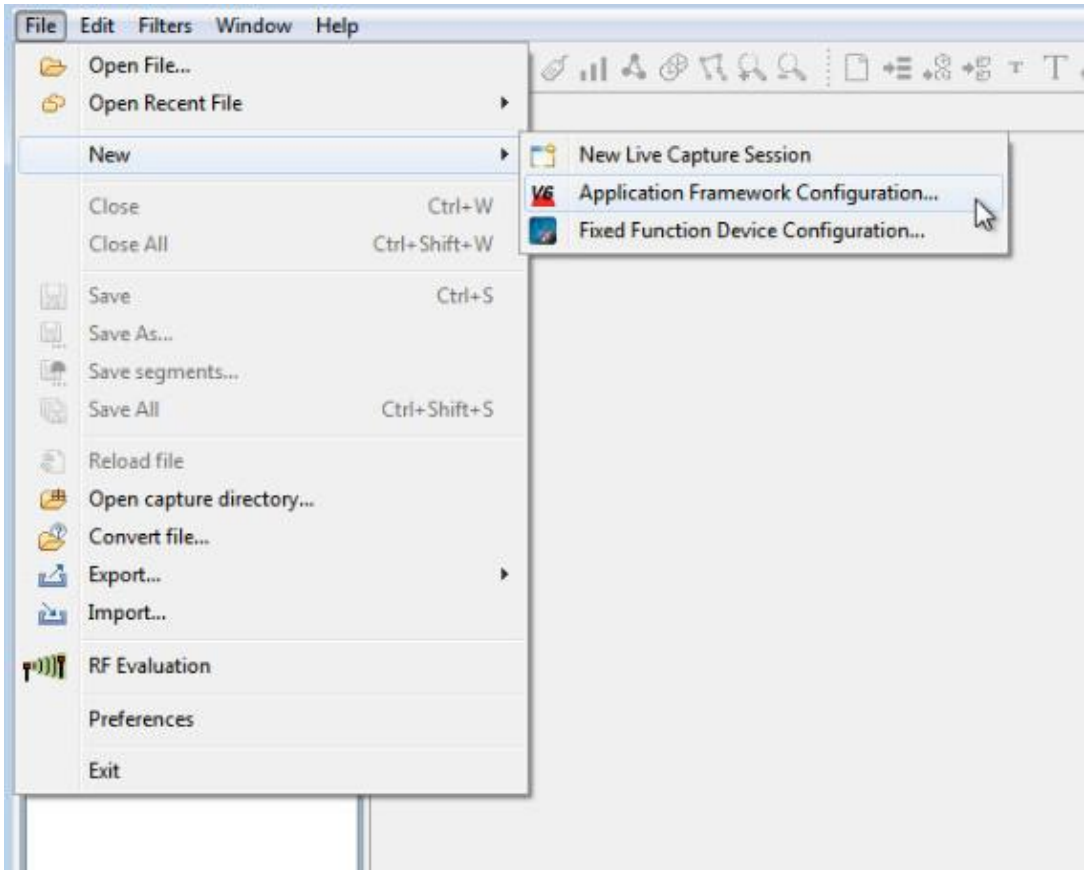
2.3.5 Step 5: Load your application onto your device

After you compile your binary image, it is ready to load on your device. You can use Ember Desktop or one of the Ember command-line loaders to load your image.

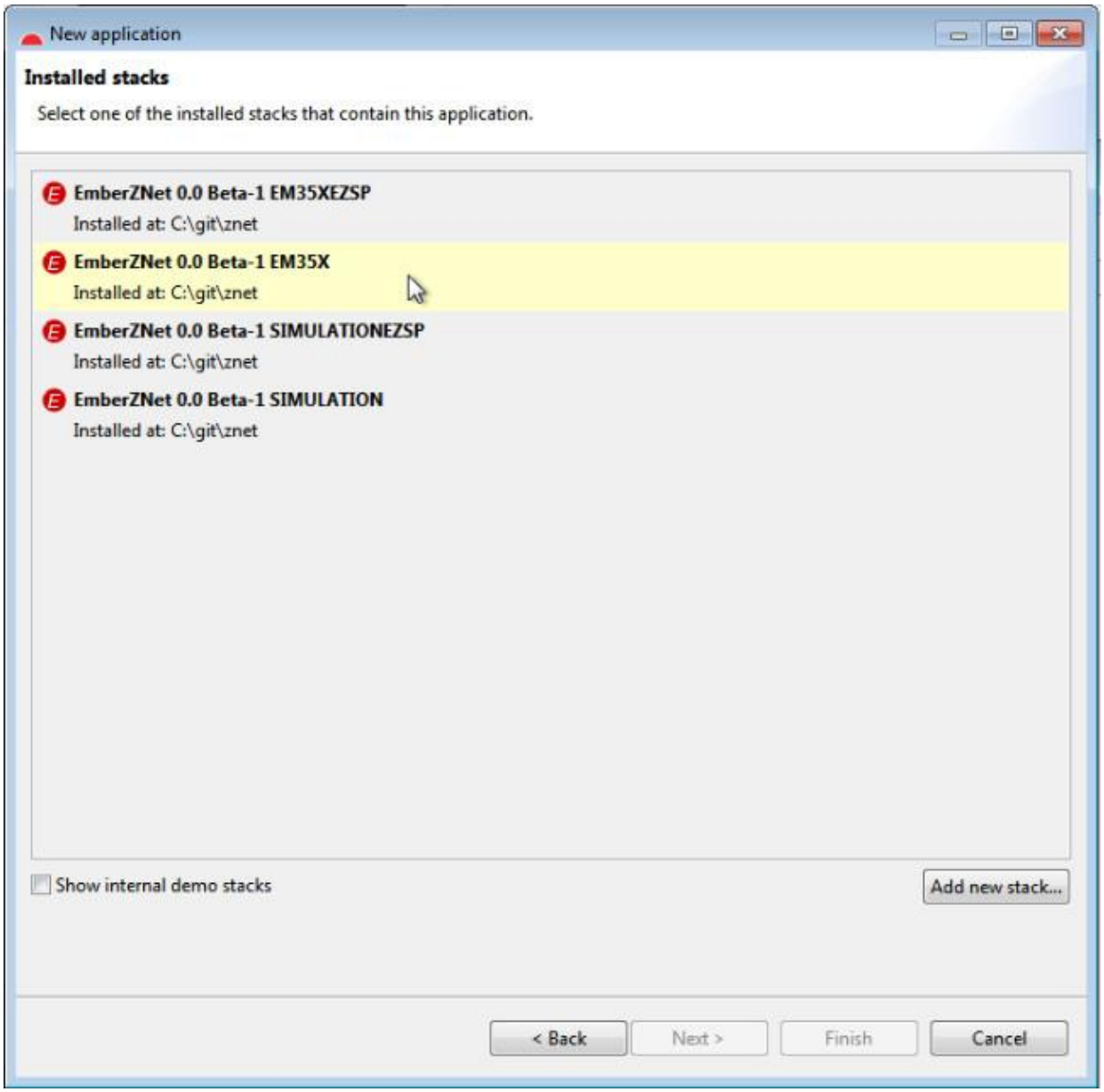
3 Creating A New Device Configuration

Each ZigBee device corresponds to a single device configuration and is represented in a single Device Configuration Editor.

You can create a new device configuration at any time within Ember Desktop by choosing **File | New...**



Simplicity Studio AppBuilder supports several different stack configurations. When a new device configuration is created, Simplicity Studio AppBuilder asks which stack you would like to use. For more information on this see Simplicity Studio AppBuilder preferences.



3.1 Saving a device configuration

You can save a device configuration into Simplicity Studio AppBuilder's native .isc file format. To save your device configuration as an .isc file, choose **File | Save...**

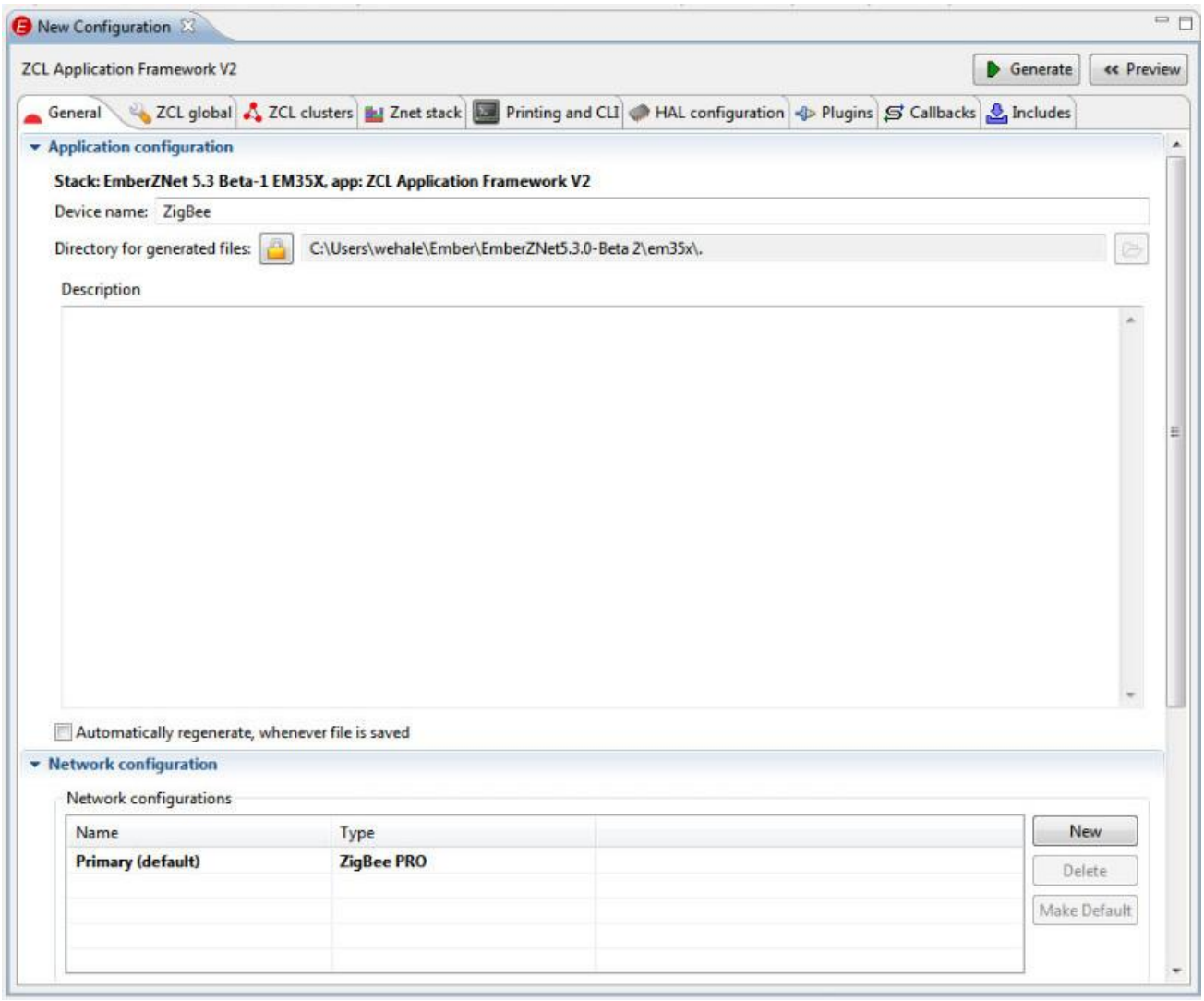
To open an existing device configuration, choose **File | Open File...**

3.2 Auto-generate upon save

Simplicity Studio AppBuilder can automatically generate configuration files when you save your configuration to a .isc file. To use this option, select the **Auto-generate upon save** checkbox in the bottom right hand corner of the Device Configuration Editor.

4 General Application Configuration Tab

The General Application Configuration tab allows you to configure global values for your application like the Device Name and Generation Directory.



4.1 Device Name

The device name is used to identify your application and will become the prefix for any generated files as well as the folder name for your generated files.

4.1.1 Directory for Generated Files

By default when you click the "Generate" button, all files are generated into a sub directory under the stack installation location. The directory for generated files shows the root of the generation location. The files are generated into the (root)/app/builder/(application name) directory.

You have the option of changing the default generation location however this is not recommended as moving the generation location away from your stack installation directory may result in compile errors.

4.1.2 Description

The application description is saved into the project .isc file and is for your use. It does not have any effect on the application or its generation.

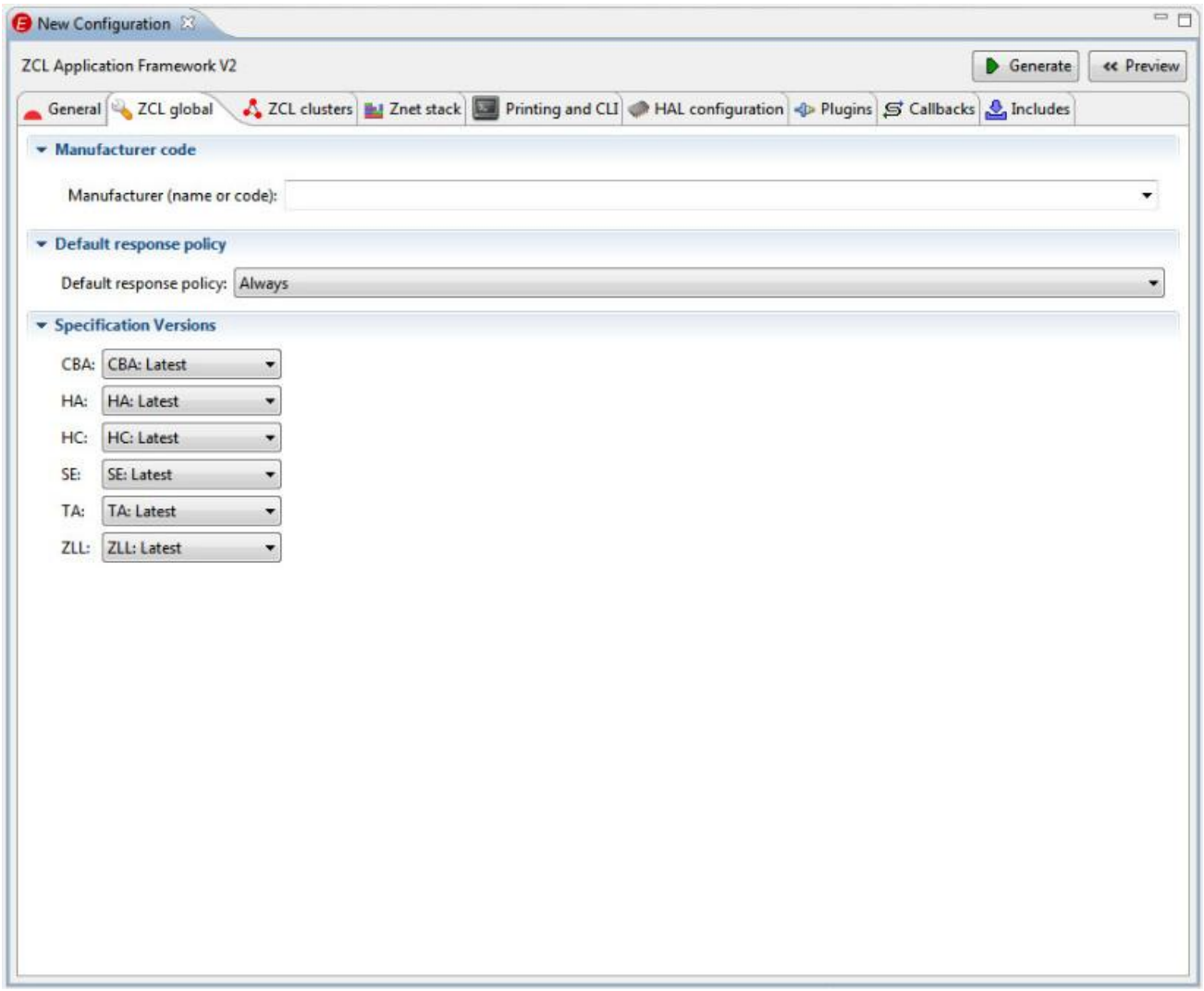
5 Network Configuration

The Network Configuration area is where you configure the number of networks that will be used by your application. An application may only be a coordinator or router (always on) on a single network so if you have more than one network configured you must make sure that only one of the networks configured is an "always on" type network.

To add a network simply click on the "New" button. The new network will show up in the table and you can then give the network a name for your own reference and a type. While the network is created and named on the General Application tab, it is configured on relevant stack tab. For instance if you create two ZigBee Pro networks you must configure them on the Znet Stack tab in order to ensure that only one of them is an "always on" type network.

6 ZCL Global Configuration Tab

The ZCL Global Configuration Tab is where you configure all of the ZigBee device level configuration elements for your device.



7 Manufacturer Code

The manufacturer code is used to indentify your device and is stored in the application header file.

8 Default Response Policy

The default response policy defines how your application will request default responses to outgoing messages.

8.1 Always

The device will always expect a default response for all outgoing messages.

8.1.1 Conditional

The device will request a default response for all messages unless in is instructed not to do so by the profile xml description.

8.1.2 Never

The device will never ask for a default response to any messages that it sends. Thus the "enable default response" bit will be set to "false" for all transmitted messages.

9 Specification Versions

The specification versions area allows you to configure the specific specfication version that your application will be targeted at. Any capabilities that are not supported in the specfication version chosen will be removed from your application and application configuration options.

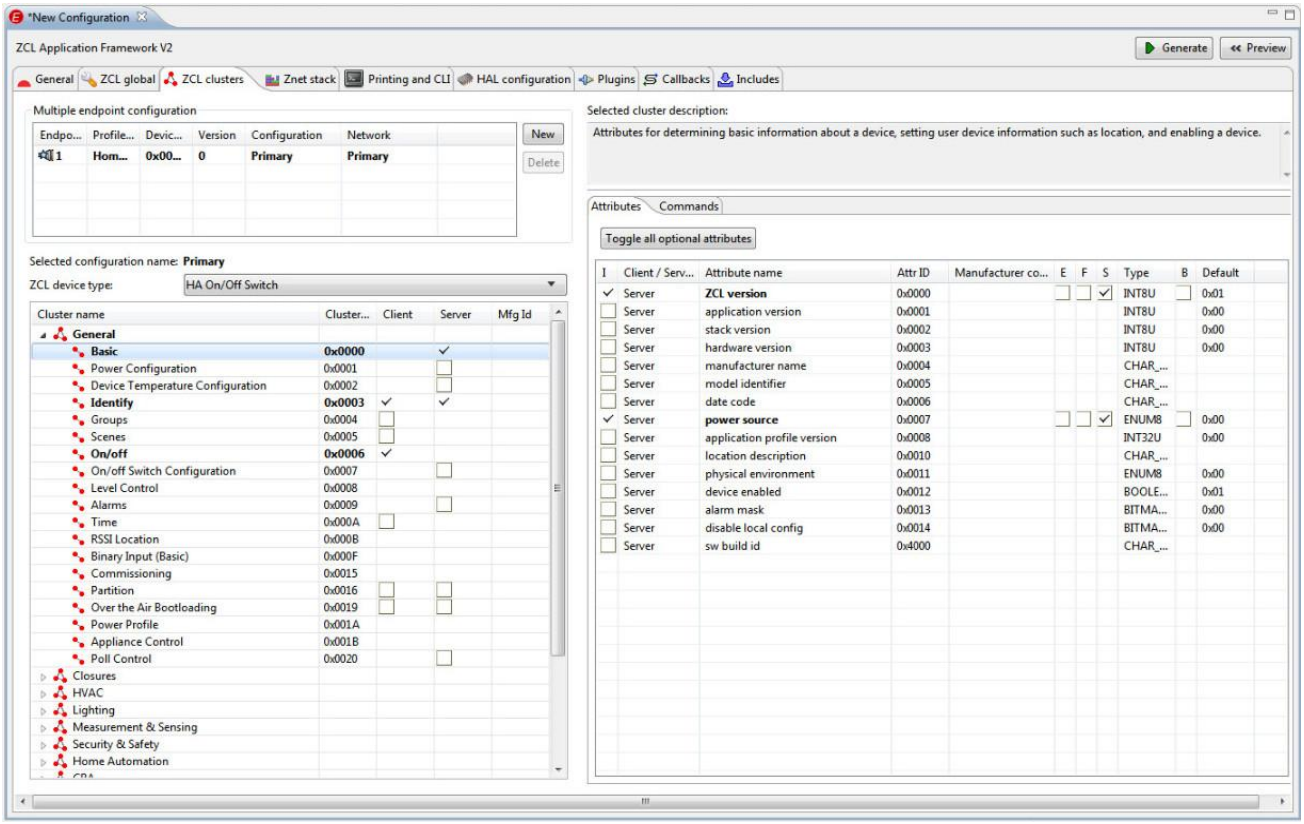
10 ZCL Cluster Configuration Tab

Each Device Configuration Editor contains a single **ZCL cluster configuration** tab. This tab allows you to choose the specific combination of clusters to include in your ZigBee device.

For more information on ZigBee Clusters and the ZigBee Cluster Library, see document 120-3029-000, Application Development Fundamentals.

10.1 Multiple Endpoint Configuration Table

The Multiple Endpoint Configuration Table allows you to configure the endpoints on your device.



By default, a new device configuration includes only a single endpoint. If you wish to include multiple endpoints in addition to the "Primary" endpoint, you must add them to the Multiple endpoint configuration listing by clicking **New**.

10.1.1 Endpoint Number

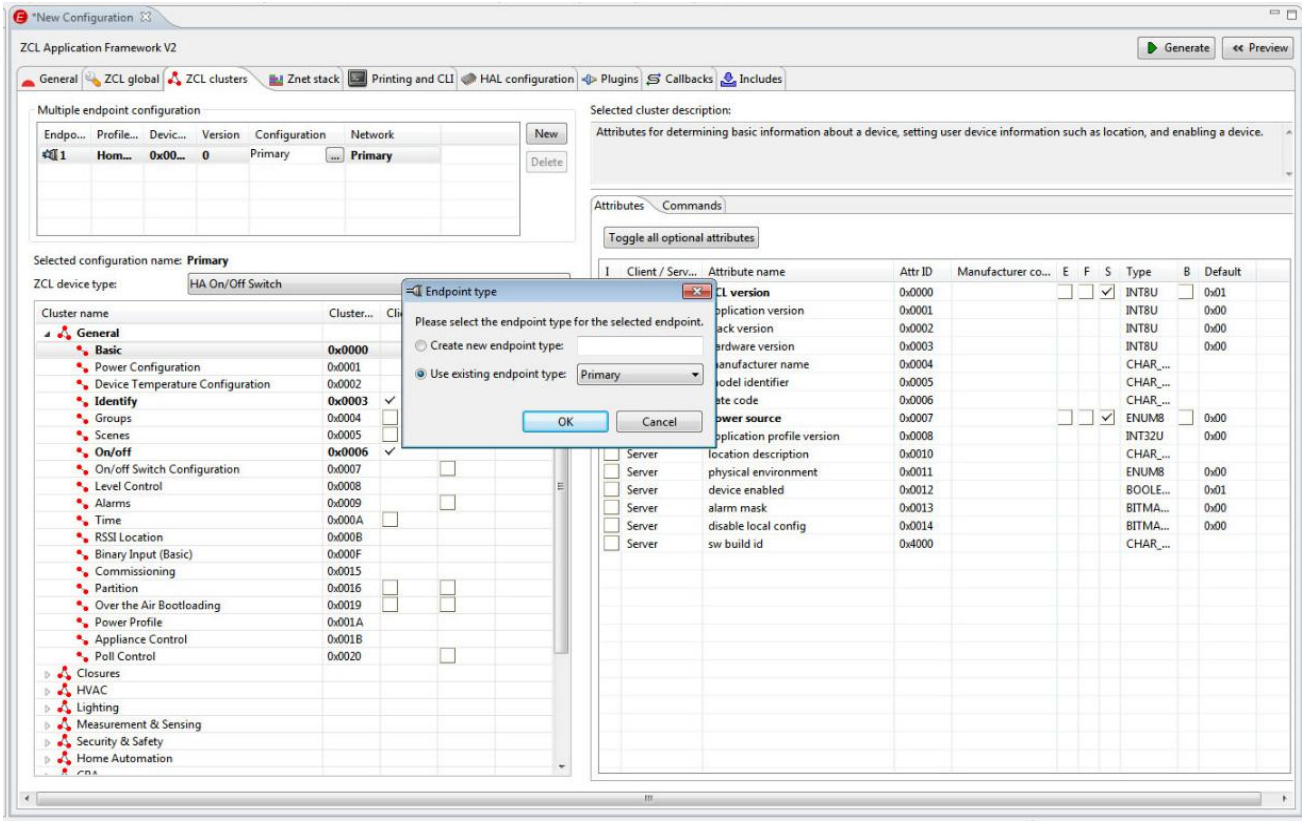
You may change the endpoint on which your device will be hosted by clicking on the number in the "Endpoint" column. When you click on the Endpoint number, the number field becomes editable and you may change the endpoint to any of the valid ZigBee endpoints available for your use.

10.1.2 Endpoint Type

Each endpoint implements a specific endpoint type. The endpoint type is not a ZigBee construct. It is a name created for Simplicity Studio AppBuilder so that multiple endpoints that implement the same application can share the metadata associated with that application. This conserves the use of flash within the device.

By default, each new endpoint will implement the endpoint type named "Primary."

You may change the endpoint type by selecting the endpoint row and then clicking on the "Configuration" column. A button appears to the right of the Configuration name. Click the button to launch the dialog for endpoint type.



Once you have created a new endpoint type, the application for that type can be implemented on as many endpoints as you like without using any more memory on your device for metadata storage. Each endpoint still needs to have its own attribute storage to contain the endpoint-specific values. Simplicity Studio AppBuilder and the application framework take care of this for you.

10.1.3 Network

As of EmberZNet PRO 4.7, some platforms support the ability to operate on multiple networks simultaneously. EmberZnet PRO 4.7 limits this multi-network functionality to two networks but it will be extended in the future. On these platforms, each endpoint belongs to a specific network. Multiple endpoints can belong to the same network, but each endpoint belongs to only one network.

By default, the first endpoint will belong to the network named "Primary." When new endpoints are created, they will belong to the same network as the first endpoint.

On platforms that support this feature, you may change the network for each endpoint by choosing the network that you wish to use for a specific endpoint.

New networks can be created and configured in the Network Configuration area of the General Tab. Networks are configured on the stack tab depending on the type of network created. For instance if you create a ZigBee Pro network, it can be configured on the Znet Stack Tab

10.2 Generated device type pulldown

The **Generated device type** pulldown allows you to select from a set of pre-defined ZCL Device Types. When you select a specific ZCL Device Type, Simplicity Studio AppBuilder populates the cluster table with

the clusters appropriate to that device type as defined by the ZigBee Cluster Library specification.

In addition to these pre-defined device types, you also have the option to create a custom device type. With this option, you can choose from any of the available ZigBee clusters included within the Ember Application Framework.

10.3 Cluster Table

The Cluster Table displays which clusters will be included as part of each endpoint type. In the case of a custom device implemented on any endpoint, you have the option to choose the clusters to include and their client/server role.

10.4 Attribute Table

The Attribute Table is situated to the right of the Cluster Table. When you click a specific cluster, its attributes display in the Attribute Table. You may configure the device's behavior by configuring the attributes in this table.

I	Attribute name	Attr ID	E	F	S	Type	B	Defa...
✓	**ZCL version**	0x00...	☐	☐	☑	INT8U	☐	0x01
☐	application version	0x00...				INT8U		0x00
☐	stack version	0x00...				INT8U		0x00
☐	hardware version	0x00...				INT8U		0x00
☐	manufacturer name	0x00...				CHA...		
☐	model identifier	0x00...				CHA...		
☐	date code	0x00...				CHA...		
✓	**power source**	0x00...	☐	☐	☑	ENU...	☐	0x00
☐	location description	0x00...				CHA...		
☐	physical environment	0x00...				ENU...		0x00
☐	device enabled	0x00...				BOO...		0x01
☐	alarm mask	0x00...				BITM...		0x00
☐	disable local config	0x00...				BITM...		0x00
☐	sw build id	0x40...				CHA...		

10.5 External Attribute storage

By default, all attributes are stored in a RAM buffer provided by the application framework. Storing some attributes may not make sense, however, since they are either already stored somewhere else in the system or are not stored at all but read from some external piece of hardware. In this case you may indicate that the attribute is externally stored. When the application framework needs to interact with the attribute it will use the external attribute callbacks such as emberAfExternalAttributeWriteCallback or the emberAfExternalAttributeReadCallback. By implementing these callbacks and marking the attribute as externally stored, the attribute storage is removed from the framework and placed into the domain of the application. The metadata used to represent the attribute on the network is still stored inside the framework so that the framework can respond to ZDO requests and so on.

10.6 Persistent attribute storage

As noted above, by default, all attributes are stored in RAM. As a result, attribute data does not survive a reboot of the device. If you want to maintain state for an attribute across reboots, the attribute must be stored in persistent memory on the device. In order to store a specific attribute in persistent memory, select the checkbox in the F column.

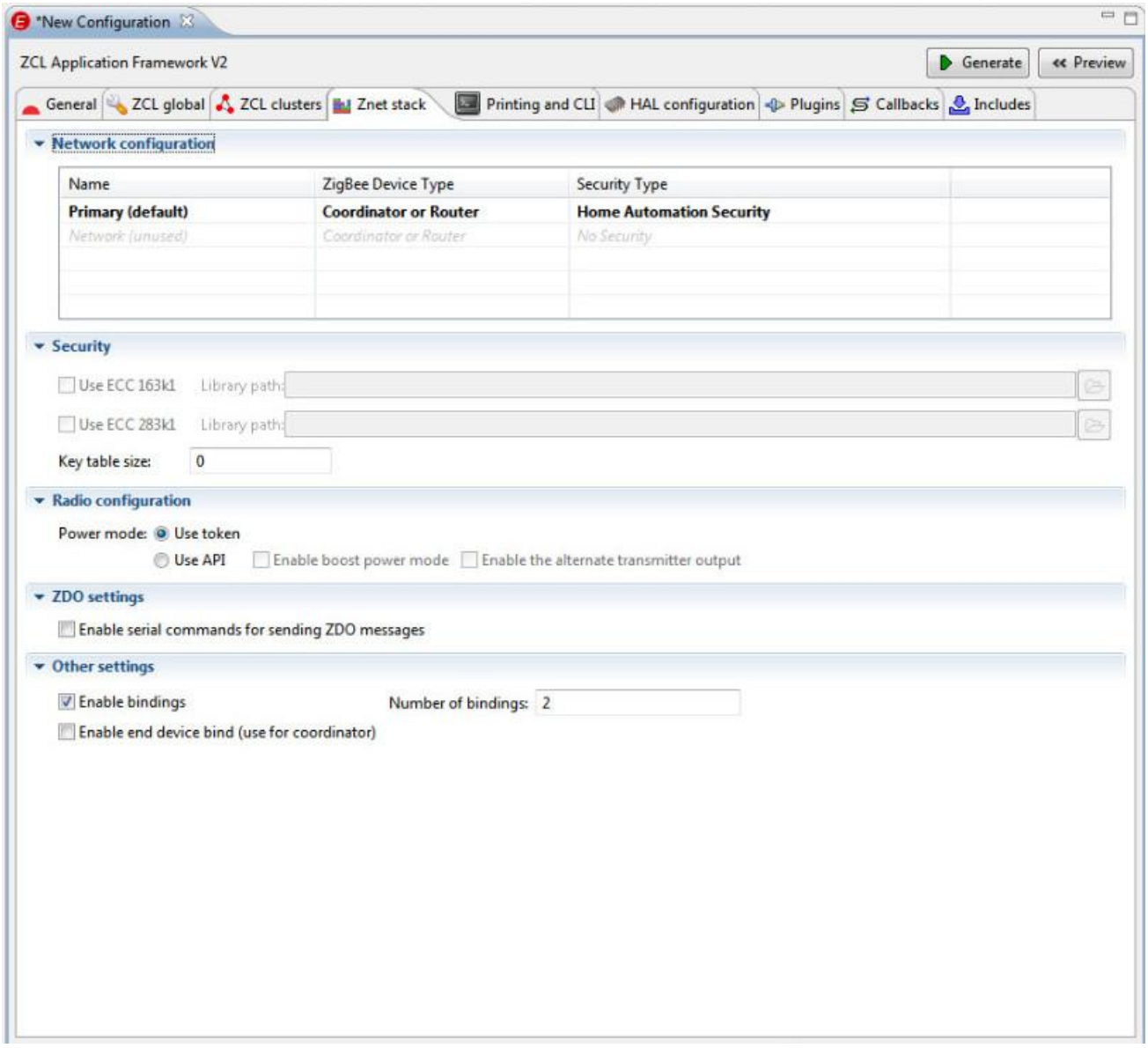
10.7 Singleton attribute storage

Most attributes are stored as a separate value for each endpoint on which they are implemented. This does not make sense for some attributes, which may exist across endpoints. For instance, it makes little sense to store multiple copies of the Basic Cluster's ZCL Version since the version pertains to the entire application and not to each individual endpoint. Attribute values may be shared across endpoints by indicating that the attribute is a "singleton."

10.8 Bounding attributes to their Min and Max values

By default, the Ember application framework allows you to write any value (within the limits of the size of the attribute) into the attribute table. However, you may request that the application framework reject any attribute value outside the Min and Max defined by the ZigBee specification. by selecting the checkbox in the B column on the Attribute Table. The application framework then stores the min and max values for the chosen attribute in the attribute table and rejects any write request, whether from the CLI or an external device, that is outside the attribute value range.

11 Stack Configuration Tab



The **Stack configuration** tab allows you to configure all of the stack-specific settings for your device. Included under this tab are:

- Network configuration
- Security
- Radio configuration
- ZDO settings
- Other settings

To expand and collapse the sections under this tab, click the arrow to the left of each section header.

11.1 Network configuration

The ZigBee device type and security profile of the network are configured here. As of EmberZNet PRO 4.7, some platforms support the ability to operate on multiple networks simultaneously. The EmberZNet PRO 4.7 release limits this multi-network behavior to two networks but this will be extended in the future. On these

platforms, the ZigBee device type and security profile can be configured per network.

11.1.1 ZigBee Device Type

- Coordinator or Router
- Router
- End Device
- Mobile End Device
- Sleepy End Device

11.1.2 Security Type

Choose the security configuration you wish to use for your network. This section allows you to choose a preconfigured security setup for your device based on either the HA or SE security standard. You may also optionally set up custom security for your network.

11.1.2.1 No Security

Turns off security on the device. This means that the device will not use network or link layer security.

11.1.2.2 Home Automation Security

Turns on Home Automation security on the device as defined by the ZigBee HA specification.

11.1.2.3 Smart Energy Security (Full)

Turns on full SE security as defined by the ZigBee SE specification. This security requires that you set up predefined link keys running on your network. The coordinator of the network needs to be set up with individual link keys for each device.

11.1.2.4 Smart Energy Security (Test)

Turns on the test version of SE Security. This setting should only be used for development. In this case, the coordinator uses a well-known global default link key for each device.

11.1.2.5 Custom Security

Prior to EmberZNet PRO 4.7, devices were able to configure custom security in Simplicity Studio AppBuilder.

Link and Network Keys

Allows you to set up custom network and link keys for your devices. If the device is a coordinator you can set both the network and link keys for your network. If the device is a router or end device you can set the link key used within the network.

Network Key Switching

For added security, you can configure your ZigBee trust center to change the network key used on the network either periodically or manually. To do this, select the checkbox "Enable network key switch." Then configure the key switch mechanism by using the dropdowns to the right of the checkbox.

The application framework comes with two strategies for key switching, Random and Ping-Pong. If you select the Ping-Pong method, the coordinator periodically switches the network key used between the two preconfigured network keys provided in the security settings.

11.2 Security

11.2.1 ECC Library Path

If you are using Smart Energy security with ECC on an Ember SOC you must provide a path to the ECC library you wish to include in your binary image.

11.3 Radio configuration

Prior to EmberZNet PRO 4.7, you can choose the channels and power setting for your device. As of 4.7, these options are set in plugins that need them.

As of EmberZNet PRO 4.6, you can choose the power mode for your device.

11.4 ZDO settings

Choose ZDO settings for your application here.

11.5 Inter-PAN settings

Prior to EmberZNet PRO 4.5, you can choose inter-PAN settings for your application here. As of 4.5, these options are set in plugins that need them.

11.6 Other settings

This section includes the attributes for enabling bindings, enabling receiving statistics, and enabling end device bind.

11.6.1 Enable bindings

Use this setting to allow your device to maintain bindings to other devices within the network. For more information on stack bindings, see document 120-3029-000, Application Development Fundamentals.

11.6.2 Address table size

As of EmberZNet PRO 4.3, you can set the address table size and trust center cache size for your application here.

11.6.3 Enable end device bind

Turns on end device binding for a coordinator.

11.6.4 Enable receive statistics

Turns on the recording of receive statistics within your application. Receive statistics include: # packets received in incoming message handler, # packets passed to ZCL processing, # packets dropped due to length errors. These statistics can be accessed using the 'stats' command on the command line.

11.6.5 Use Fragmentation

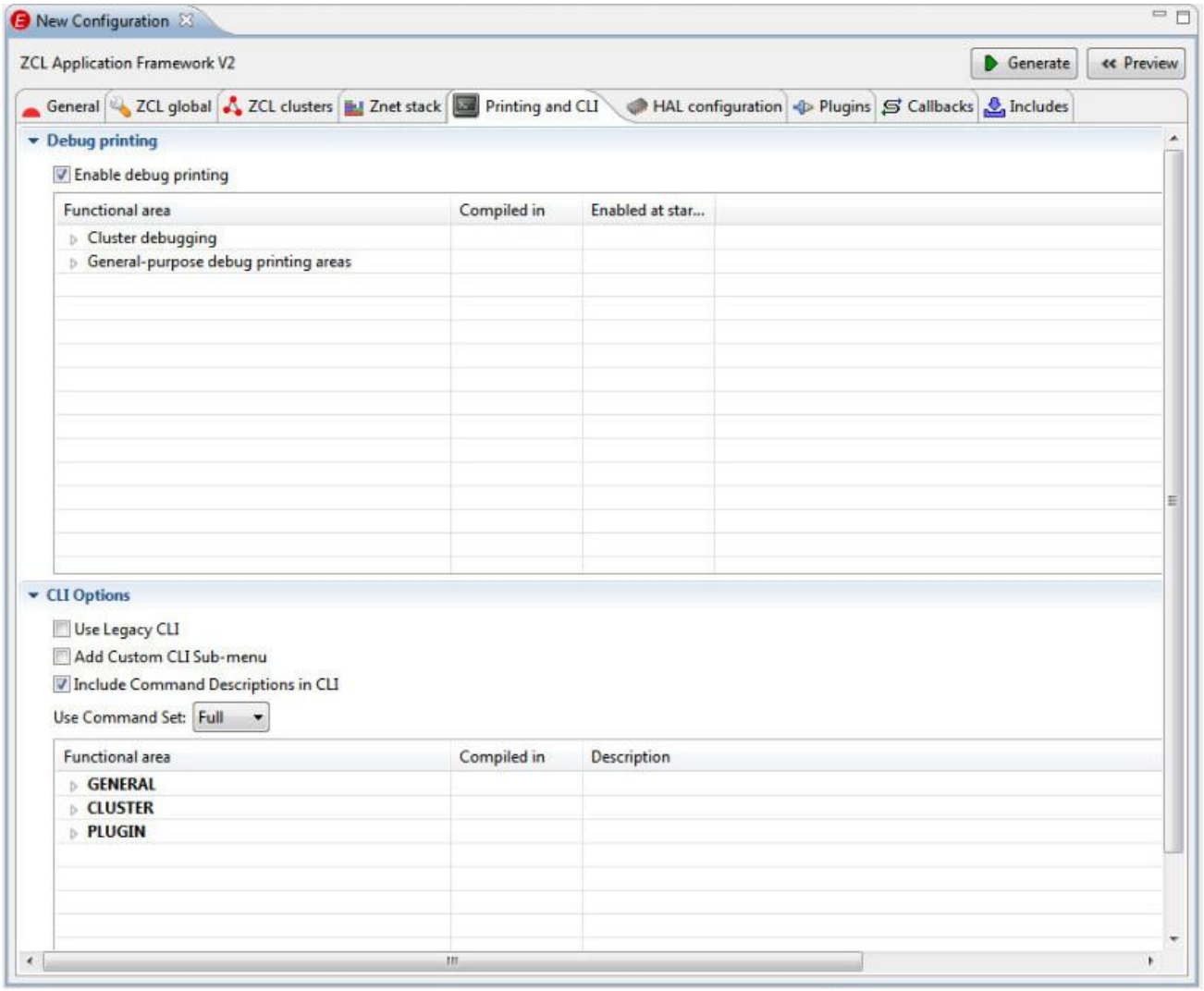
Prior to EmberZNet PRO 4.6, you can turn on support for the transmission and reception of fragmented messages. A fragmented message is one which exceeds the maximum message size, is broken up and re-assembled by the Ember stack. A buffer must be allocated on the application in order to receive the de-fragmented message. You may also set the maximum message size for this buffer. As of 4.6, these options

are set in plugins that need them.

11.6.6 Concentrator Support

Prior to EmberZNet PRO 4.7, you can turn on code associated with source routing and keeping track of source routes. If this device is a concentrator it will send out periodic many-to-one route requests so that it can record routes to the devices it will communicate with. When you turn on concentrator support, you also have the option to choose the type of concentrator your device will be - either a Low RAM or High RAM concentrator. As of 4.7, these options are set in plugins that need them. For more information on source routing and data aggregation see document 120-3029-000, Application Development Fundamentals.

12 Printing and Command Line Configuration Tab



12.1 Debug printing

The Ember application framework can output application and stack debug information over the serial port. To choose the information to include in the debug output, select from the serial printing checkboxes.

If you are not concerned with debug output, and/or would prefer to conserve the flash and RAM associated with debugging, turn off the serial printing options by deselecting them.

12.2 Command Line Interface (CLI) Options

Simplicity Studio AppBuilder is capable of outputting either a "Legacy" or "Generated" command line interface.

12.2.1 Use Legacy CLI

When this checkbox is selected, Simplicity Studio AppBuilder reverts to using the hard coded CLI interface included in the Application Framework. When it is not selected, AppBuilder generates a Command Line Interface along with other generated files based on the CLI interface options selected in the CLI configuration window.

12.2.2 Add Custom CLI Sub-menu

Adds the ability to include user defined CLI commands into the Command Line Interface.

12.2.3 Include Command Descriptions in CLI

Includes a description for each command into the Command Line Interface. Turning this on will increase the size of your application but will make it easier to use the Command Line Interface since all commands will be self described when a command is not recognized by the application.

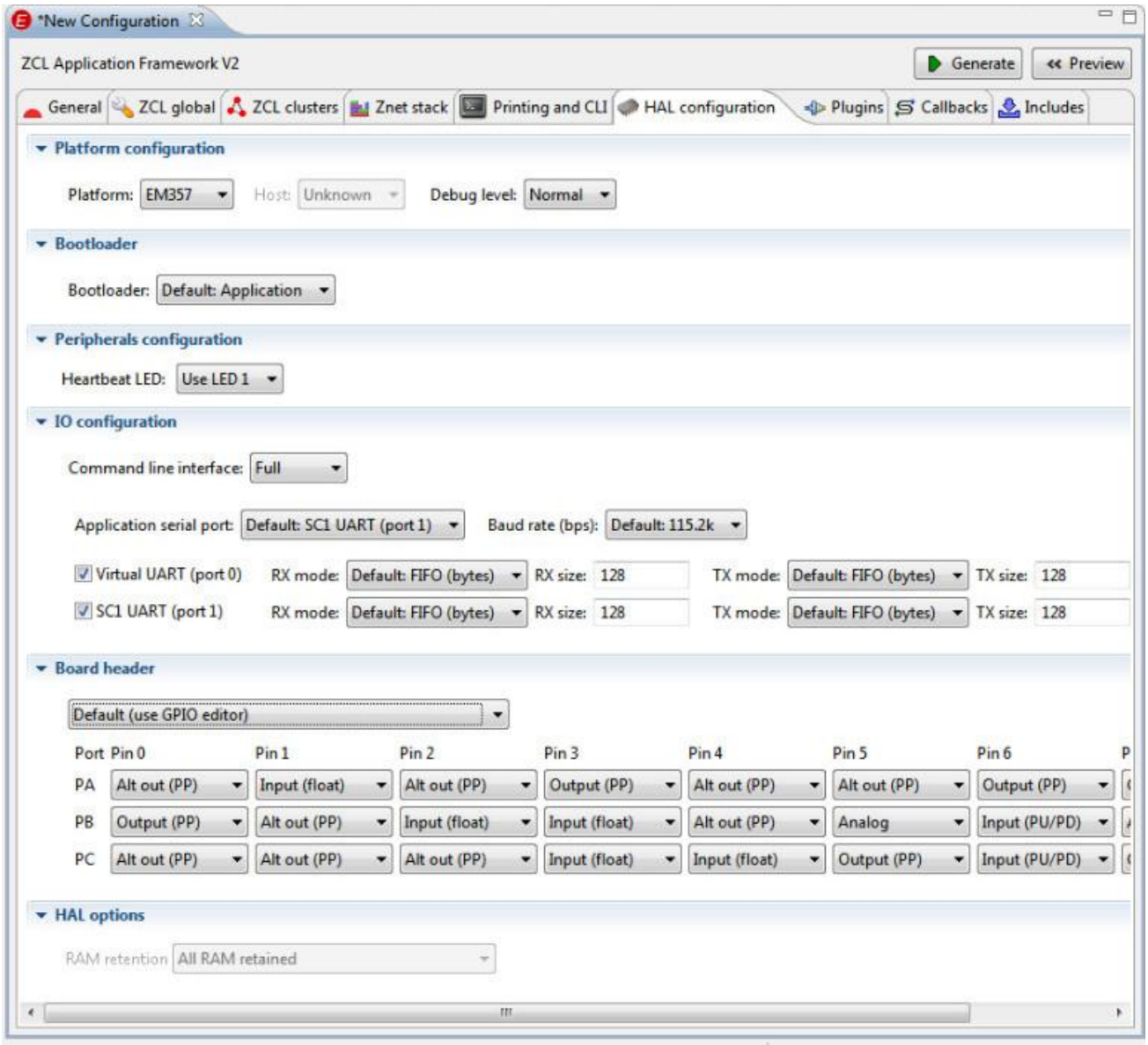
12.2.4 Use Comamnd Set

Makes it possible to turn the command line interface on and off and choose a pre-defined subset of the CLI for use in your application

12.2.5 CLI Configuration Window

The CLI Configuration Window allows the user to select exactly what CLI commands are supported in the application. By default, the Full command line interface command set selects all General commands and all cluster and plugin commands supported by clusters and plugins included in the application.

13 HAL Configuration



The **HAL configuration** tab allows you to configure hardware-specific options for your device. You will want to make sure that the binary image you create works not only as a ZigBee device, but also with your own hardware. The hardware configuration process can be somewhat confusing. The HAL tab remedies that problem by providing many of the most commonly used settings as a default.

Note: There are many options within the Board Header file for configuring GPIO differently for the needs of your board. Silicon Labs recommends you review the generated xxx_board.h file and make changes as needed for your target hardware.

The HAL tab offer options for:

- Platform configuration
- Peripherals configuration
- IO configuration
- GPIO register configuration

To expand and collapse the section for each option, click the arrow to the left of each section header.

13.1 Platform configuration

Specify the type of chip onto which your binary image will be loaded, and the libraries that will be loaded with it.

13.1.1 Platform

Enter the type of Ember chip for which your binary image will be built.

13.1.2 Bootloader

If you wish to include a bootloader on the chip with your binary image, select which type of bootloader you will be using. If you do not need a bootloader on the chip alongside your binary image, select None.

13.1.3 Debug level

The EmberZNet PRO stack includes several levels of debug. The more debug included in the stack, the larger the stack.

During the process of development, you may wish to include full debug and then reduce the level once your application is ready for deployment.

13.2 Peripherals configuration

13.2.1 Heartbeat LED

This configuration option allows you to either set or disable the execution of a heartbeat from the application. A heartbeat is a periodic indication that the application is running and is useful for debug purposes.

13.2.2 First application button

This configuration option allows you to either set or disable the development board button association with the first application button global defined in the application framework. Currently the application framework has two well-known application button values, the first and second application buttons. If you wish to use the board buttons for some other purpose you may disable them here and use them as you like within your application.

13.2.3 Second application button

This configuration option allows you to either set or disable the development board button association with the second application button global defined in the application framework. Currently the application framework has two well-known application button values, the first and second application buttons. If you wish to use the board buttons for some other purpose, you may disable them here and use them as you like within your application.

13.3 IO configuration

Use the IO configuration section to configure the input and output you will be using on your Ember chip of choice.

If you are using an NCP like the EM260, this section is greyed out, because your application does not have to define an IO configuration for that chip.

13.3.1 Application serial port

Specifies the serial port and baud rate over which your application will output data from `emberSerialPrintf()` and all of the application framework prints such as emberAfCorePrintln().

The command line interface pulldown allows you to configure the type of CLI that will be included in your application. There are three different types of CLI: Full, Minimal and Tiny.

- **Full CLI** provides a complete set of CLI commands enabled by your application configuration. This option is the most user-friendly but also takes up the most space on your application.
- **Minimal CLI** provides a subset of commonly used CLI commands available to the Full CLI to conserve on CONST and RAM.
- **Tiny CLI** provides a special set of single character commands used to form and set up a network on the predetermined channel 11 with PanId 0x00ab. This can be used for testing highly memory-constrained devices.

Documentation on the application framework CLI is included in document 120-3023-0000, Application Framework API Reference Guide.

13.3.2 Virtual UART port 0 mode

Enables the UART on port 0 and configures both the RX and TX queue sizes for it. In the case of a FIFO buffer, the queue size is the number of bytes allocated for the buffer. In the case of message buffering, it is the number of messages allocated to the buffer. This port can also be marked as unused in order to conserve the RAM normally allocated for its buffers.

13.3.3 SC1 UART port 1 mode

Enables the UART on port 1 and configures both the RX and TX queue sizes for it. In the case of a FIFO buffer, the queue size is the number of bytes allocated for the buffer. In the case of message buffering, it is the number of messages allocated to the buffer. This port can also be marked as unused in order to conserve the RAM normally allocated for its buffers.

13.4 GPIO register configuration

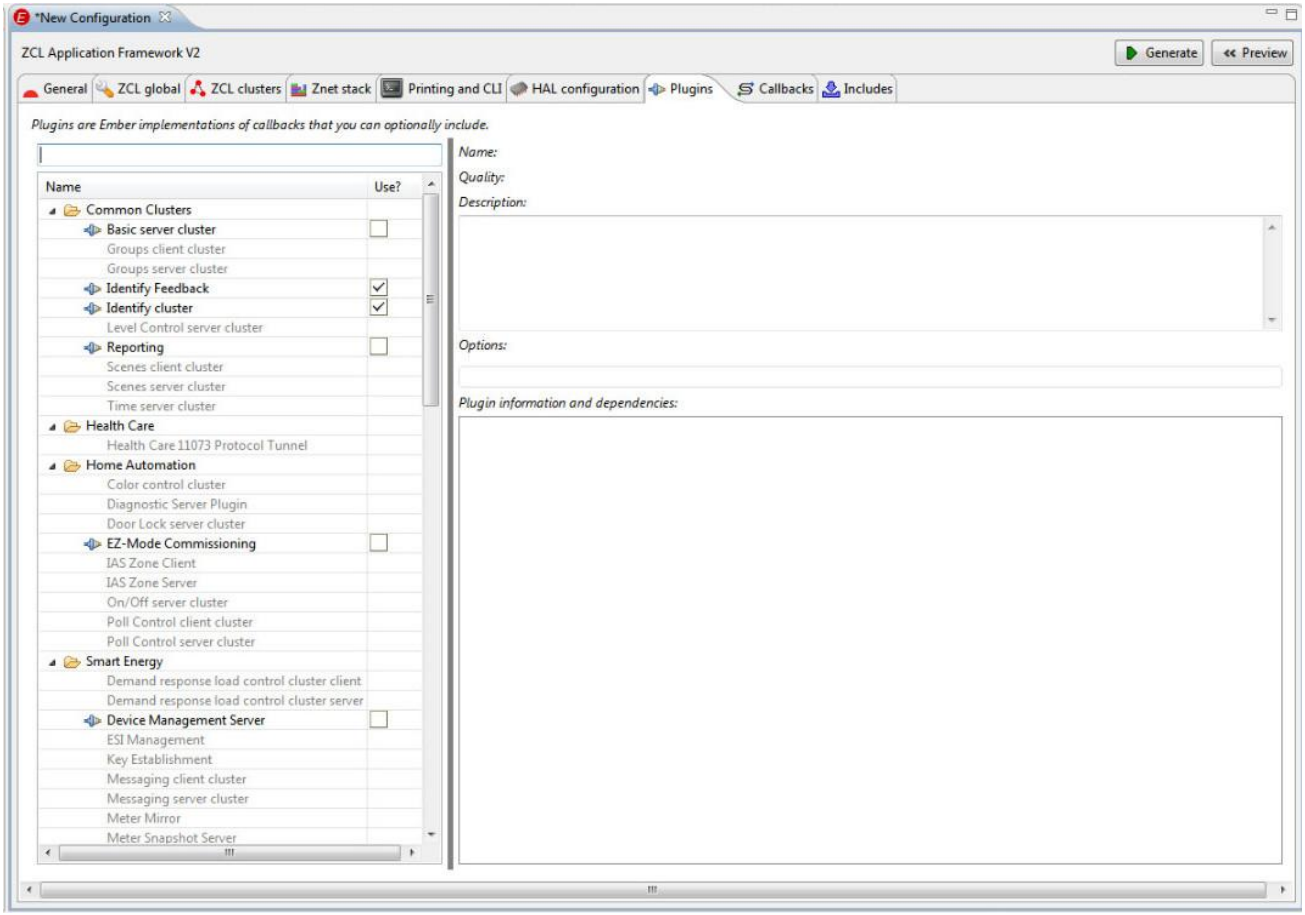
Use the interactive interface to set the GPIO bitmask used to configure your device.

If you are using an EM250, you will see two tabs that provide two different ways to configure the same information. The first tab provides a friendly interface, while the second provides a more advanced interface for users who are comfortable editing the GPIO bitmask by hand.

If you are using the EM35x family of chips, a simple granular set of pulldowns is provided. These allow you to set values for each pin and each port. The settings for the Ember 35x development board are provided by default. Please consult the 35x datasheet before changing any of these settings, as complex relationships between them must be considered.

For more information on GPIO settings for your Ember chip of choice consult the datasheet for your chip available through the ZigBee section of Silicon Labs' website at http://www.silabs.com/zigbee

14 Plugin Configuration Tab



The application framework allows you to include or exclude plugins. A plugin is a specific implementation of a cluster or some other functionality created by Silicon Labs that may be useful to your application.

If a plugin is included in your application, it will consume the callback interfaces that it cares about. The callbacks that are consumed by a plugin will have a small plugin icon to the left of the callback name. Callbacks that are consumed by a plugin are not available to the application. Thus, if you use a plugin, you must accept its use of the callbacks.

14.1 Plugin table

The plugin table on the right hand side of the plugin tab displays all of the plugins that are available for your application.

Many plugins are turned on by default. If you wish to implement your own functionality, you may deselect a plugin and select the relevant callbacks in the callbacks tab.

14.2 Plugin options:

Many plugins include configurable options. You may choose defaults for your plugin by selecting from the option menus on the right hand side of the plugin tab.

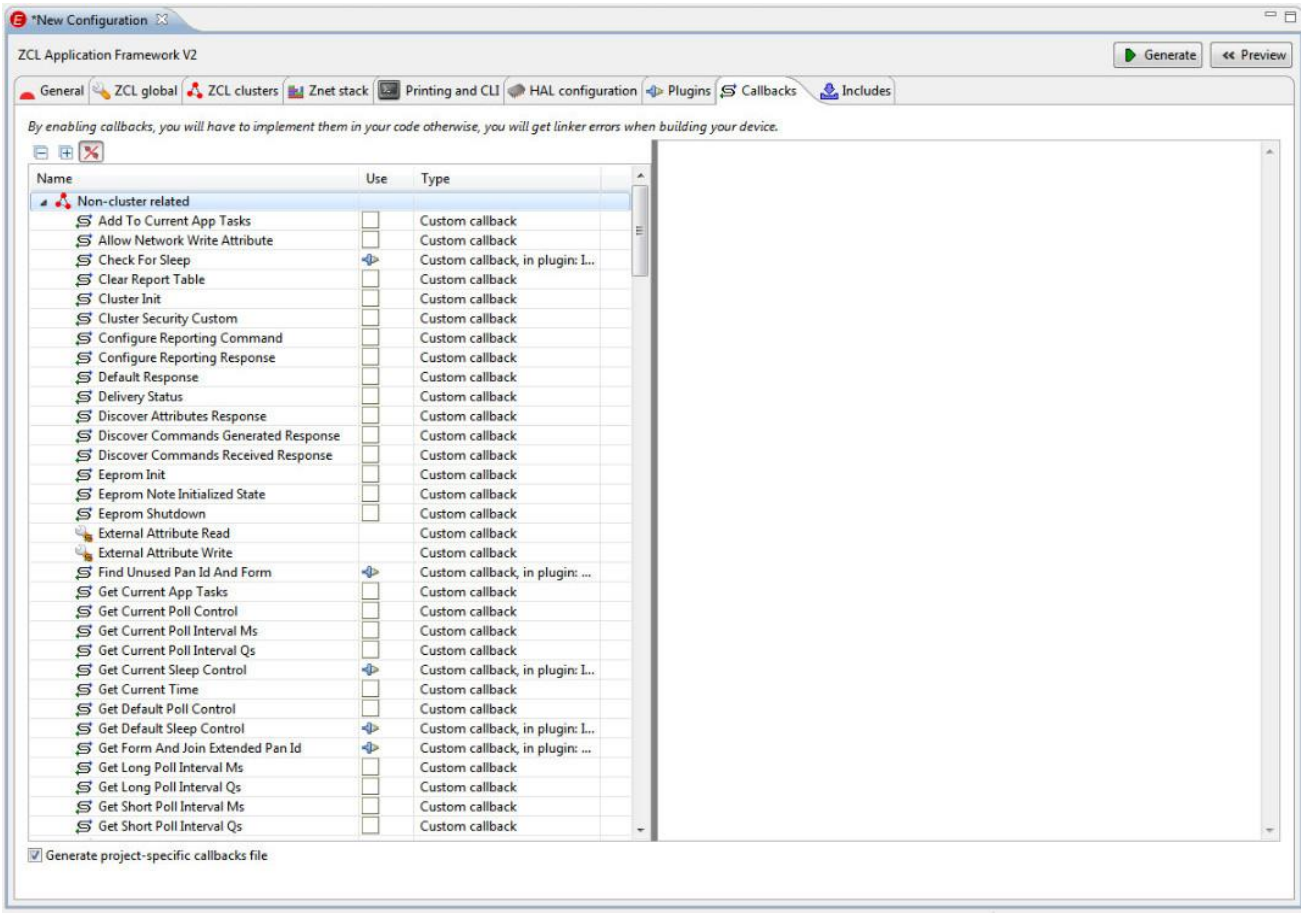
14.3 Plugin source files

The source files for each plugin are located in the Ember application framework under <stack install loc>/app/framework/plugin/<plugin name>. The source files included are listed in the plugins information area on the right hand side of the plugin tab.

14.4 Implemented callbacks

The callbacks consumed by each plugin are listed in the "Implemented callbacks" informational area on the right hand side of the plugins tab. As noted above, callbacks that are implemented by the plugin are not available to the application. If you wish to implement one of these callbacks you must deselect the plugin and implement the functionality within your application.

15 Callback Configuration Tab



The application framework gives you the option of implementing handlers for a set of predefined callbacks. The callback interface provides a clean separation between your code and the application framework.

Each callback is documented in the comment at the top of the callback prototype.

15.1 Callback grouping

Callbacks are grouped into logical sections in the callbacks tab for ease of navigation. All of the custom application framework callbacks are included in the "Non-cluster related" section. All of the other callbacks are grouped by cluster and are generally used for cluster command handling.

15.2 Cluster callbacks

Many of the cluster-specific callbacks are intended for command handling. When a command comes in, it is passed off to a cluster- specific callback for processing. If the callback returns TRUE then the internal handler for that command will not be called. Commands that include a default Ember implementation are marked with a plus symbol (+) to denote that the application framework cares about that particular command, and will handle it and return the necessary default response. Cluster-related callbacks are also only implemented for those client and server clusters that are included in the application. If a cluster is not included, the application is not expected to parse that cluster's commands. The application framework returns a default response of UNSUPPORTED_COMMAND.

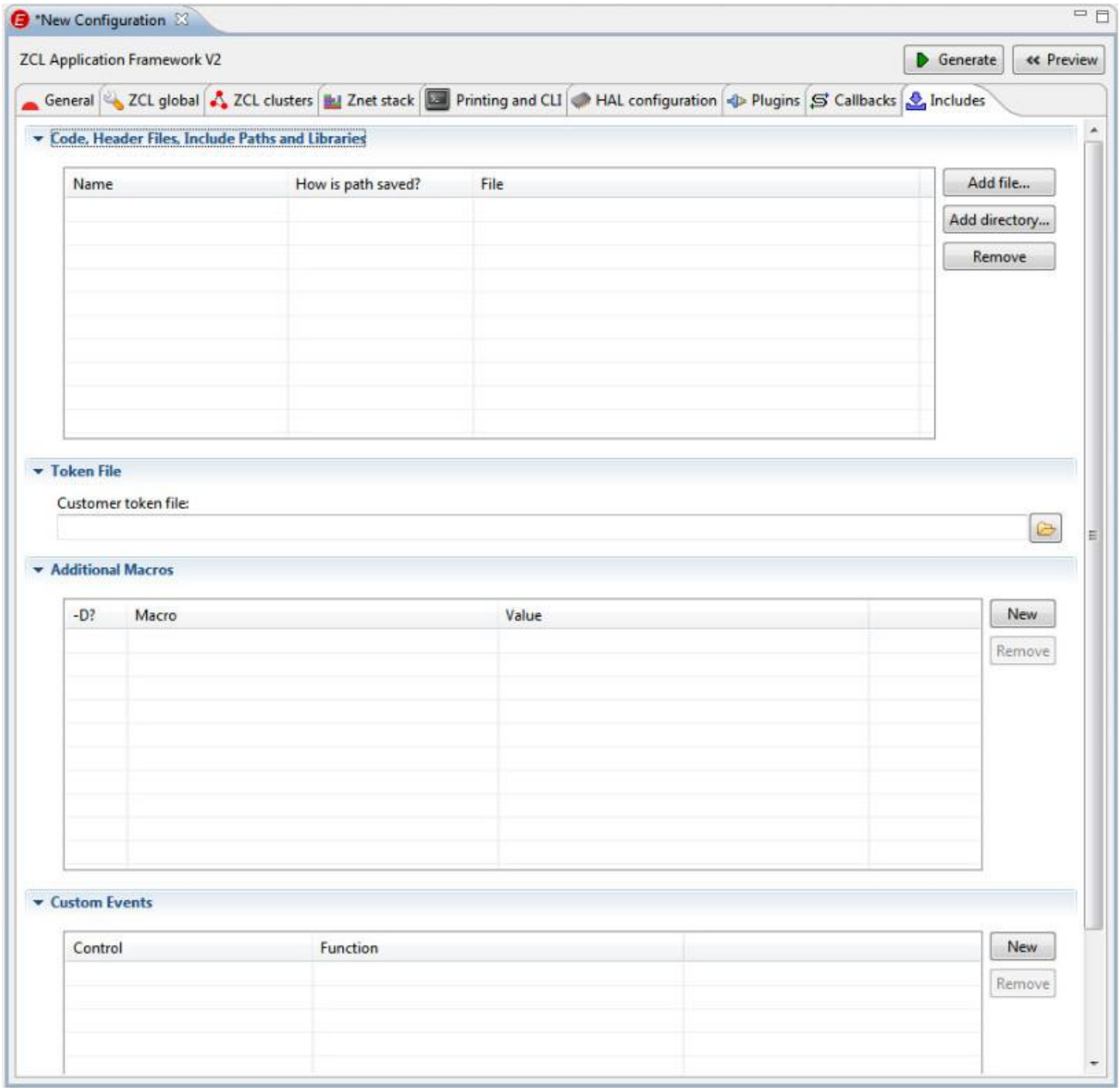
15.3 Callback generation

The first time you generate your application from Simplicity Studio AppBuilder, it automatically generates a callbacks.c file with the name <appname>_callbacks.c.

When you regenerate files in the future, Simplicity Studio AppBuilder protects your generated callbacks file from being overwritten by asking if you want to overwrite it. By default, Simplicity Studio AppBuilder will not overwrite any previously created callbacks file. If you choose to overwrite the file, Simplicity Studio AppBuilder backs up the previous version to the file <appname>_callbacks.bak.

16 ZCL Include Configuration Tab

The include configuration tab gives the user the opportunity to include any external files into the generated project.



16.1 File Include

When you generate a project file from Simplicity Studio AppBuilder, you often need to modify the project to include your own source files into the application image. This is a problem if the project file needs to be re-generated, since the newly generated project file will overwrite the old one, thereby removing any files that have been included.

Any files that you would like to compile along with your application can be included in the file include table. To include a new file, click on the "New" button. This button opens a file chooser dialog. Navigate to the location of the file you wish to include and click OK. The file will be included inside the generated project file with an absolute path to the location of the file.

16.2 Token File

In order to implement attributes in tokens, the Ember application framework places its own token file into the generated APPLICATION_TOKEN_HEADER #define within the generated project file.

The application framework's token header file, located at app/framework/util/tokens.h, includes a spot where you may place a relative path to your own token header. The token header files are chained off one another with the application framework's token header at the top of the chain. To include your own token header within your application, provide a path relative to app/framework/util inside the Token File text box.

16.3 Additional Macros

Much of the configuration for the application framework and the stack is done using macros documented within the API. Any additional macros that you wish to define, either for compile time or for run time, should be provided here.

Compile-time macros should include the "-D" option. These macros will be included in the compiler and linker definition sections in the generated project file. Any macros which do not include the "-D" option will be included in the generated application header file.

16.4 Custom Events

You can include custom application events into the application framework's event system. These events are run in the same manner as the application framework's own events as well as all cluster events implemented by the cluster plugins.

In order to include a custom event, click on the "New" button. Your custom event will be automatically included in the event configuration section of the generated endpoint_configuration.c file. Stubs for your event function and event control will be generated into the callbacks.c file. You may also change the name of the generated event control value and function to whatever you desire.

For more information on how events work please see document 120-3028-000, the Application Framework Developer Guide, which is available with your stack install documentation as well as from the Silicon Labs' website.

17 Saving Your Device Configuration

Once you have created a device configuration, you can save it at any time by choosing:

File | Save...

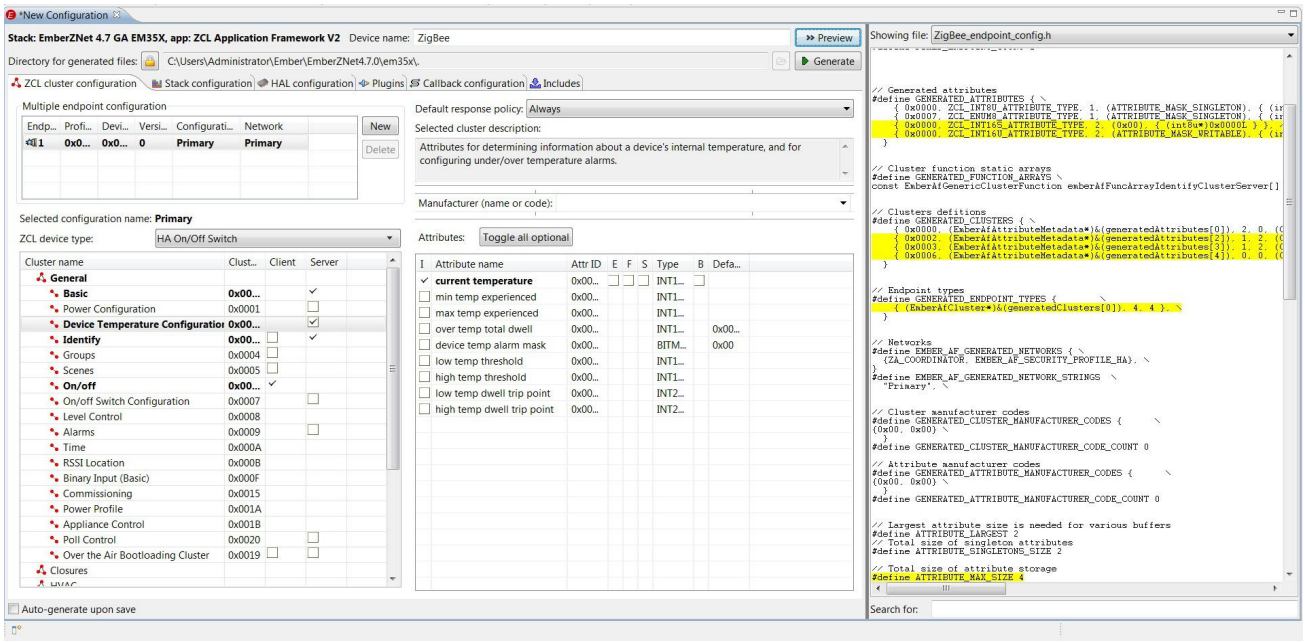
The .isc file format is AppBuilder's native format. All of the settings you have chosen for your configuration are saved out into this file and will be loaded in when you open the file in AppBuilder in the future.

To open a previously saved .isc file, choose:

File | Open...

18 Previewing your Device Build Files

Simplicity Studio AppBuilder comes equipped with a great debug and learning tool called the Preview Pane. When the Preview Pane is open, any changes to the output files are highlighted in yellow. This highlighting is useful for checking that your settings are being reflected in the configuration files you will ultimately generate to build your application. The Preview Pane is also useful for helping you learn the application framework's inner structure and about ZigBee in general.

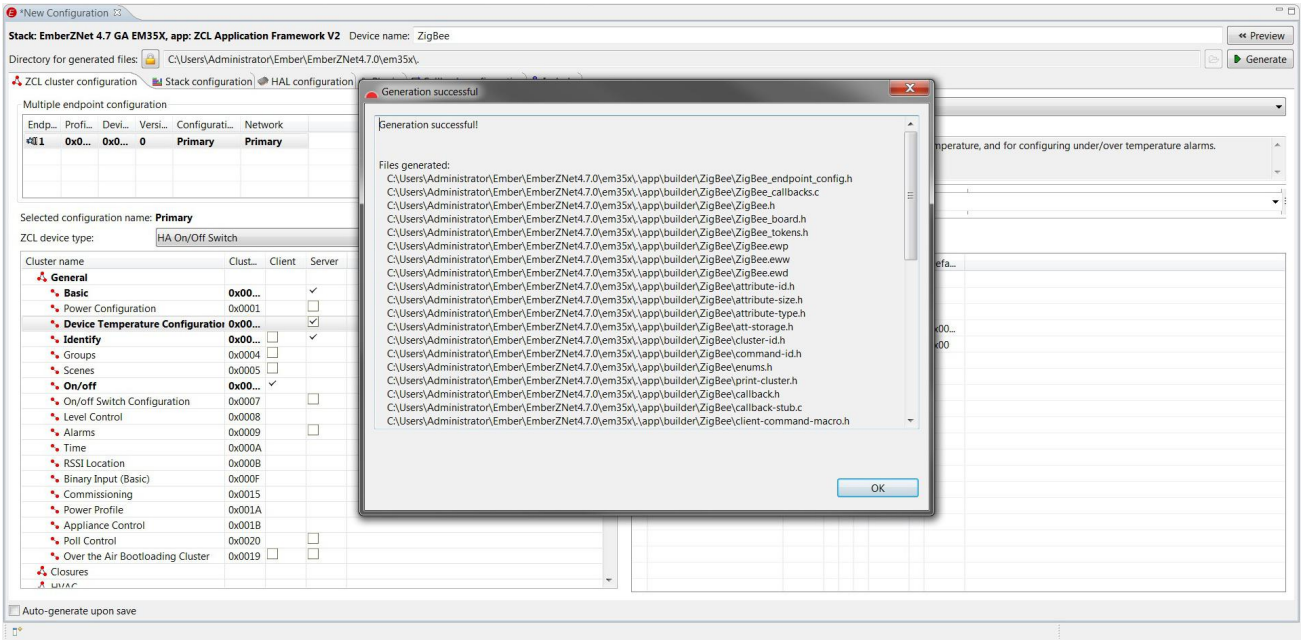


19 Generating Your Device Build Files

Once you have completed specifying your device configuration in the **ZCL**, **Stack**, **HAL**, **Plugins**, **Callback**, and **Includes** tabs, you are ready to generate your device build files. You do this simply by clicking the **Generate** button.

19.1 Generation Directory

The generation directory needs to be the same as the stack directory. This is true because the generated project files expect to find the included stack libraries and framework files in a particular location relative to their own. So, while Simplicity Studio AppBuilder allows you to indicate where you would like to generate the files, you can not just stick the generated files wherever you like and trust that the project files will compile properly.



19.1.1 Simplicity Studio AppBuilder generated files

When you click the **Generate** button, Simplicity Studio AppBuilder generates device configuration files, some of which are described in this table. These files configure the Ember application framework at build time so that the appropriate code and settings are included in the compiled binary image.

#	File Name	Generation Directory
1	<device name>.h	<generation directory>/app/builder/<device name>/
	Application configuration header file. This is the main application configuration header file for your device. This file includes all of the #defines required to configure your device.	
2	<device name>_endpoint_config.h	<generation directory>/app/builder/<device name>/
	Endpoint configuration header file. This file specifies the structure of the attribute metadata and value storage table. This file essentially makes multiple endpoints possible by constructing the attribute table in such a way as to limit the amount of flash used.	
3	<device name>_callbacks.c	<generation directory>/app/builder/<device name>/
	This is the only .c file created by Simplicity Studio AppBuilder. It is only created in the case	

	where you want Simplicity Studio AppBuilder to create a stub file for handling callbacks you have activated in the callbacks tab.	
4	<device name>_board.h	<generation directory>/app/builder/<device name>/
	This is the board header file for your device. This file includes all the #defines required to configure the hardware abstraction layer to work with your device.	
5	<device name>_tokens.h	<generation directory>/app/builder/<device name>/<device name>_tokens.h
	This is a token header file used to configure storing data in tokens (SIMEEPROM). If you have chosen to have any attributes persist across reboots, the configuration data for these settings are stored in this file.	

20 Building A Binary Image

After you generate your build and configuration files by clicking the **Generate** button, you are ready to build your binary image. Simplicity Studio AppBuilder supports several compilers including:

- IAR Embedded Workbench for the Cortex (EM35x, stm32f, stm32w)
- GCC ARM Embedded

You have the option of either compiling your application inside Simplicity Studio by clicking on the hammer icon to build in the toolbar or by opening the generated project file in the compiler of your choice and building there.

21 Loading A Binary Image

After you have created your binary image, the next step is to load it onto your device. Silicon Labs provides four tools that you can use for this purpose:

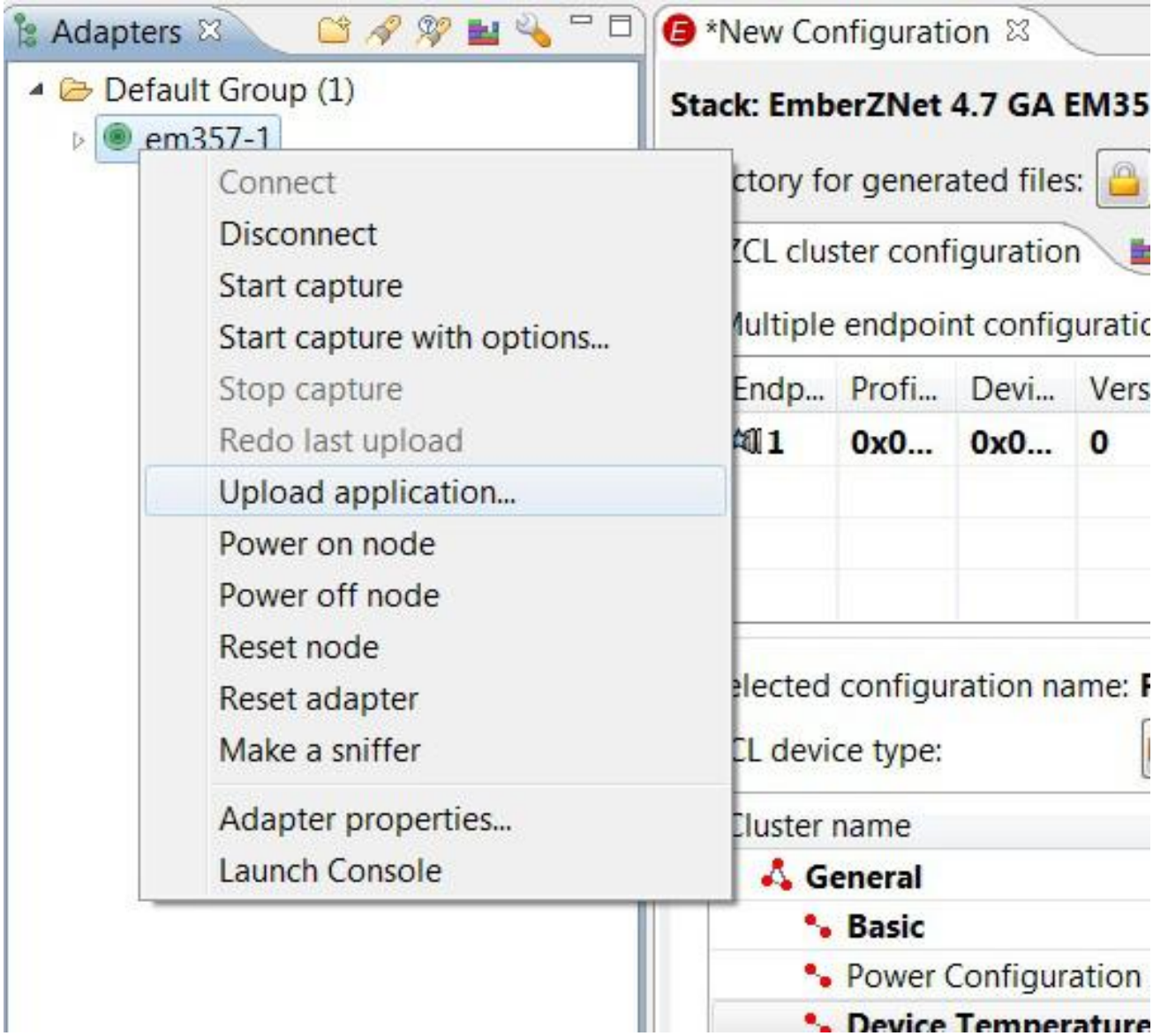
- em2xx_load.exe
- EM2USBLoad.exe
- EM2ISALoad.exe
- em3xx_load.exe

You can also use Simplicity Studio to load binary images onto your devices. Simplicity Studio uses the 2xx and 3xx series tools by proxy to load images on your devices. See the Simplicity Studio User's guide (Ember_Desktop_Users_Guide.pdf) and online help for more information.

For information on loading images onto your devices using one of the four tools, see the following documentation:

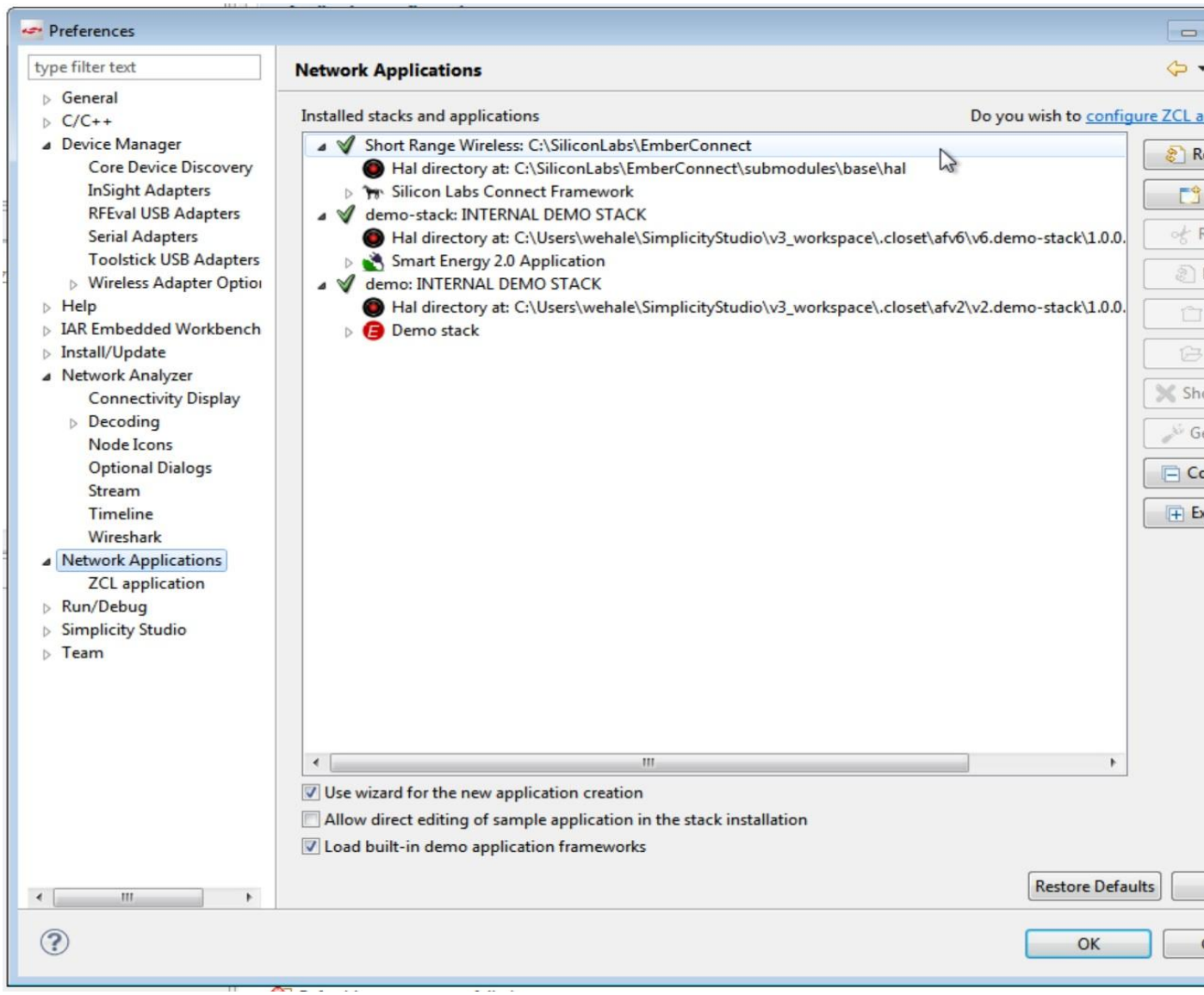
- **em2xx_load.exe**: 120_4020_000_2xx_Utilities_Guide.pdf
  Used to load images onto 2xx devices connected to an Ember InSight Adapter from the command line.
- **EM2USBLoad.exe**: 120_4022_000USB_Link_Users_Guide.pdf
  Used to load images onto devices connected to an InSight USBLink from the command line.
  Note: InSight USB Link is legacy software. While it is still maintained and supported, it is not being actively developed. It is highly recommended that customers work with InSight Adapter or with one of our gang programming partners for improved reliability.
- **EM2ISALoad.exe**: 120_4022_000USB_Link_Users_Guide.pdf
  Used to load images onto devices connected to InSight Adapters. This program is similar to em2xx_load, but with the added advantage of being able to load over the Ethernet through an Ember Debug Adapter or over USB through a USBLink.
- **em3xx_load.exe**: 120_4032_3xx_Utilities_Guide.pdf
  Used to load images onto 35x devices connected to an Ember Debug Adapter (ISA3) from the command line.

You can find these guides through the ZigBee section of Silicon Labs' website at http://www.silabs.com/zigbee.


Adapters
Default Group (1)
em357-1
Connect
Disconnect
Start capture
Start capture with options...
Stop capture
Redo last upload
Upload application...
Power on node
Power off node
Reset node
Reset adapter
Make a sniffer
Adapter properties...
Launch Console
*New Configuration
Stack: EmberZNet 4.7 GA EM35
ctory for generated files:
ZCL cluster configuration
ultiple endpoint configuratio
Endp...
Profi...
Devi...
Vers
1
0x0...
0x0...
0
elected configuration name:
CL device type:
luster name
General
Basic
Power Configuration
Device Temperature

22 Setting Simplicity Studio AppBuilder Preferences

To set preferences for Simplicity Studio AppBuilder, choose **File | Preferences | Network Applications**.



This preferences dialog allows you to:

- Add a new stack version to the versions known to Simplicity Studio AppBuilder
- Add custom clusters to any stack version you have installed
- Regenerate the application framework's generated code
- Inspect the completeness of the ZigBee Cluster Library for a chosen stack

In order to generate a working application using Simplicity Studio AppBuilder, you *must* have a valid version of the EmberZNet PRO stack, and you *must* identify the location of the stack for Simplicity Studio AppBuilder, using the "Add Stack Version" dialog. Once you create a configuration, its stack version is locked.

YOU CANNOT CHANGE THE STACK VERSION ON AN EXISTING CONFIGURATION FILE.

If you wish to use a different stack version, you must create a new configuration with the stack version of your choice by choosing **File | New**.

Default Path for Generated Files

The default path for generated files is at the root of your chosen stack. You may change this location in your configuration file.

22.1 Custom Clusters

You may add your own custom clusters to any (non-Demo) stack configurations located in the Simplicity Studio AppBuilder Preferences, File | Preferences | Simplicity Studio AppBuilder. To add your custom cluster, you must describe your cluster in an xml file like the ones used by the stack configuration. A sample custom xml description file is provided below. For a more detailed example of how the Simplicity Studio AppBuilder's cluster xml description files are formatted, see the file ami.xml, located in the /tool/appbuilder directory of your chosen stack.

Note: The application framework expects that all attributes and commands will be unique based on their cluster id, attribute id, command id and direction (client to server / server to client).

Sample cluster description file

```
<?xml version="1.0"?>
<configurator>
  <cluster>
    <name>Custom Cluster</name>
    <domain>Custom Domain</domain>
    <description>This cluster provides an example of
      how a custom cluster is created.</description>
    <code>0xfc00</code>
    <define>CUSTOM_CLUSTER</define>
    <client init="false" tick="true">true</client>
    <server init="false" tick="false">true</server>
    <attribute side="server" type="INT8U" code="0x0000"
      writable="true" default="0x1e" min="0x00" max="0xff"
      define="CUSTOM_ATTRIBUTE_1">custom attribute 1</attribute>
    <attribute side="server" type="INT8U" code="0x0001"
      writable="true" default="0x1e" min="0x00" max="0xff"
      define="CUSTOM_ATTRIBUTE_2">custom attribute 2</attribute>
    <command code="0x0" name="CustomCommand" source="server">
      <description>
        A custom command
      </description>
      <arg name="customArg1" type="INT8U"/>
      <arg name="customArg2" type="INT8U"/>
      <arg name="customArg3" type="INT8U"/>
    </command>
  </cluster>
  <!-- Custom Devices -->
  <deviceType>
    <name>Custom Device</name>
    <domain>Custom Domain</domain>
    <typeName>Custom type name</typeName>
    <zigbeeType editable="true">Coordinator</zigbeeType>
    <!-- Manufacturer specific application profiles start at 0xC000 -->
    <profileId editable="false">0xC000</profileId>
    <deviceId editable="false">0x0000</deviceId>
    <channels editable="false"><channel>11</channel><channel>14</channel>
    <channel>15</channel><channel>19</channel><channel>20</channel>
    <channel>24</channel><channel>25</channel></channels>
    <clusters lockOthers="true">
      <include client="false" server="true"
        clientLocked="true" serverLocked="true" >Basic</include>
      <include client="false" server="true" clientLocked="true"
        serverLocked="true" >Identify</include>
      <include client="false" server="true" clientLocked="true"
```

```
        serverLocked="true">Custom Cluster</include>
    </clusters>
  </deviceType>
</configurator>
```

Installing your custom cluster

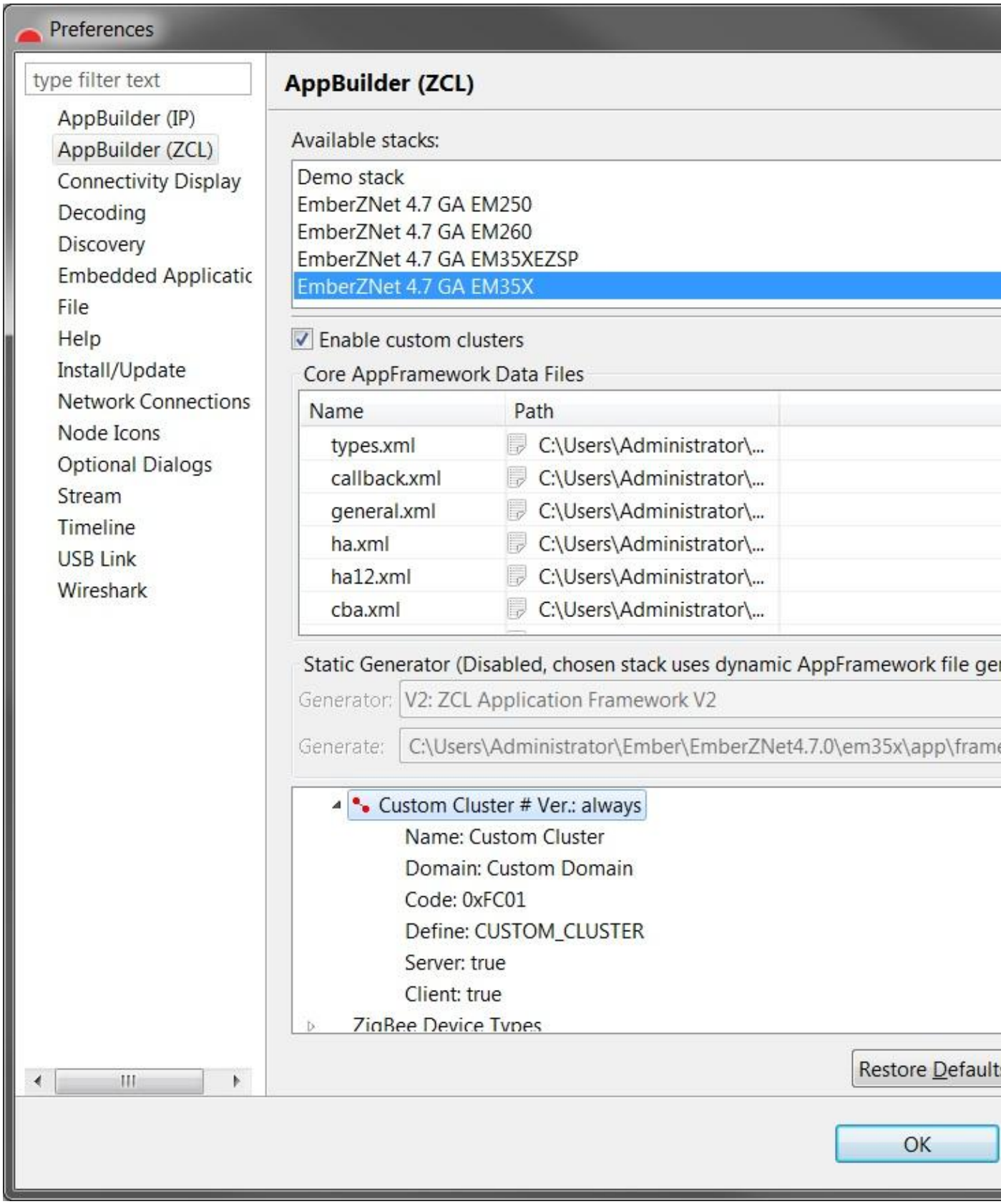
The installation of your custom cluster data involves several steps:

1. Describe your cluster in a custom cluster xml file like the one provided above.
2. Select the stack to which you will be adding your custom cluster. Note that you cannot add a custom cluster to the Demo stack profile.
3. Turn on custom clusters for your chosen stack configuration by selecting the "Enable custom clusters" checkbox.
4. Import your custom cluster xml file into Simplicity Studio AppBuilder by clicking the **New** button to the right of the Data File table.
5. Prior to EmberZNet PRO 4.6, regenerate the application framework data files by clicking the green arrow in the bottom right corner of the "Core AppFramework Data Files". Please note that this step is often overlooked but is very important. Your device will not compile without the properly-generated header files associated with your custom clusters. This step is not necessary as of 4.6.

Configuration file information

At the bottom of the preferences dialog is a tree display called the Configuration File Information table. This table displays a tree view of the ZigBee Cluster Library functionality supported by your chosen stack version. This view is not an interactive display. It merely provides a convenient way to view the ZCL internals of your chosen stack.

Once you have loaded your custom stack information, you should see it included in the Configuration File Information table, as shown below.



22.2 New Configuration Tab

22.2.1 Generate Button

The generate button is used to generate code from the Simplicity Studio AppBuilder. The process of code generation is described in detail in the Generate section of the Simplicity Studio AppBuilder help:

Generating your device build files.

22.3 Multiple Endpoint Configuration

The Multiple Endpoint Configuration table allows you to configure the number and identity of the endpoints on your device. This table is discussed in greater detail in the cluster configuration area.

22.4 Configuring Endpoint

Indicates the endpoint type currently presented in the device type pulldown and cluster configuration table.

22.5 ZCL Device Type

The ZCL Device Type pulldown allows you to choose a specific preconfigured ZigBee device as a starting point for your endpoint type. Note that the ZCL device type is different from the ZigBee device type. The ZCL Device type represents how the device behaves within the application, while the ZigBee Device Type represents how the device behaves within the ZigBee network.

22.6 Reporting table size

Reporting table size defines the maximum number of the reported items in the generated application. Increasing this number will increase the amount of RAM the application requires.

22.7 Manufacturer Code

Use the Manufacturer Code to differentiate between standard zcl attribute reporting and reporting that is specific to a particular manufacturer. If you do not provide one explicitly, the application framework will supply one for you. The default Manufacturer Code is 0xABCD.

22.8 Enable link keys

The enable link keys check box allows the user to turn on APS Security on their device. APS Security is used to provide end to end encryption for traffic which passes over multiple hops. For a more detailed description of APS Security and Link Keys, see document 120-3029-000, Application Development Fundamentals.

22.9 Link key table size

Since link keys are used to encrypt messages from end to end between devices in the network, the device may save a separate link key for each device in the network. The device does this by storing the 128 bit link keys in a table. The size of this table determines how many different link keys the device can store in memory.

22.10 Platform configuration

Specify the type of chip onto which your binary image will be loaded, and the libraries that will be loaded with it.

22.10.1 Platform

Enter the type of Ember chip for which your binary image will be built.

22.10.2 Bootloader

If you wish to include a bootloader on the chip with your binary image, select which type of bootloader you will be using. If you do not need a bootloader on the chip alongside your binary image, select None.

22.10.3 Debug level

The EmberZNet PRO stack includes several levels of debug. The more debug included in the stack, the larger the stack.

During the process of development, you may wish to include full debug and then reduce the level once your application is ready for deployment.

22.11 IO configuration

Use the IO configuration section to configure the input and output you will be using on the system-on-a-chip (SOC) platforms.

If you are using a network coprocessor (NCP) and host, this section will be greyed out, because your application will not have to define an IO configuration for that chip.

22.11.1 Application serial port

Specifies the serial port and baud rate over which your application will output data from `emberSerialPrintf()`

22.11.2 Virtual UART port 0 mode

Enables the UART on port 0 and configures the RX queue size for it.

22.11.3 SC1 UART port 1 mode

Enables the UART on port 1 and configures the RX queue size for it.

22.12 GPIO register configuration

Use the interactive interface to set the GPIO bitmask used to configure your device.

The two tabs provide two different ways to configure the same information. The first tab provides a friendly interface, while the second provides a more advanced interface for users who are comfortable editing the GPIO bitmask by hand.

For more information on GPIO settings for the EM250, see document 120-3029-000, Application Development Fundamentals, available through the ZigBee section of Silicon Labs' website at http://www.silabs.com/zigbee