Renesas Flexible Software Package (FSP) Pack Generation

Introduction

This document discusses how to create and add user-made Flexible Software Package (FSP) modules to e2 studio. It follows the manual FSP pack creation process and details the file hierarchy, configuration modifications, and coding expectations necessary to create and import custom FSP packs into e2 studio.

Important: For custom FSP packs to work seamlessly with e2 studio and other existing FSP packs, source and header files must follow the Renesas programming standard and middleware hierarchy. This especially applies to any FSP packs that have dependencies on other existing FSP packs, such as connectivity packs.

Required Resources

- e2 studio version 2020-10 or later
- Flexible Software Package (FSP) v5.0.0 or later
- RA Flexible Software Package Documentation (optional, but highly recommended)

Contents

Figures

1. Overview

This document is the starting point for creating and importing custom FSP packs into e2 studio. It may also be used to modify existing FSP packs.

The steps to quickly create an FSP pack include:

1. Create the module file structure (see section 3)
2. Create and modify the path description .pdsc file (see section 4)
3. Create and modify the tooling support file (see section 5)
4. Create and modify the module description file (see sections 6.1 and 6.2)
5. Zip the custom FSP pack into a .pack file (section 7)

Appendix A provides a detailed explanation of the module description file. The appendix is not required to create a working custom FSP pack; however, it provides useful information for improving the functionality of a custom FSP pack.

Note: To best illustrate the FSP pack creation process, this document discusses how to edit the existing HS300X FSP pack and import it back into e2 studio.

2. FSP Overview

The Renesas FSP is an enhanced software package designed to provide easy-to-use, scalable, high-quality software for embedded system designs using Renesas RA family of Arm Microcontrollers. The FSP provides a versatile way to build secure, connected IoT devices using production ready drivers, RTOS, and other middleware stacks.

The FSP includes HAL drivers, middleware stacks with RTOS integration to ease implementation of complex modules like communication and security. The FSP uses an open software ecosystem and provides flexibility in using bare-metal programming as well as RTOS-based applications.

2.1 FSP Software Modules

This section introduces the individual components that combine to create a comprehensive and portable FSP.

Note: Most custom packs represent Middleware and/or Pmod drivers/modules. This document will emphasize this form of module. However, the process for creating packs related to HAL drivers, functional libraries, board support packages, and RTOS, follow identical steps. For more information on these items, refer to the resources noted in each of the following sections.

2.1.1 Board Support Package

Board Support Package (BSP), customized for every RA hardware kit and microcontroller. It includes the startup code for all supported blocks. Developers using custom hardware can take advantage of the BSP, as it can be tailored for end products and your own board by using the Custom BSP Creator built into e2 studio.

2.1.2 HAL Drivers

Independent **HAL Drivers**, providing efficient bare-metal code for all peripherals and systems services. These drivers eliminate a lot of deep study of the underlying hardware in the microcontroller as they abstract the bit-settings and register addresses from the user.

2.1.3 Functional Libraries

Functional **Libraries** containing, for example, specialized software for digital signal processing or security and encryption-related functions, also reduce development time and improve the stability of the end-application. Libraries can also be found in the form of middleware. For instance, the emWin graphical package from SEGGER is available in the FSP in the form of Libraries.

2.1.4 Real-Time Operating System

An **RTOS** (Real-Time Operating System) provides a multitasking real-time kernel with pre-emptive scheduling and a small memory footprint. Amazon FreeRTOS is one of the RTOS provided as part of the FSP.

2.1.5 Middleware

Middleware, including TCP/IP communication, file systems, graphical user interfaces, and USB.

2.2 FSP Packs

FSP packs are the delivery mechanism for software components, device parameters, and BSPs, and can be used across RA ARM Cortex-M microcontroller devices.

When the FSP is installed a variety of Pack files are extracted. These packs can be classified into different categories:

- Board Support Packs
- MCU packs
- Middleware packs
- Third-party or vendor packs

Note: For more information on the CMSIS packs, see “References”.

2.2.1 Overview of the FSP Packs

The installed FSP packs are available in the folder ***e2_studio\internal\projectgen\ra\packs***. It contains all the required and supported FSP packs to create embedded applications using RA MCUs.

Figure 1 shows a snapshot of the different types of pack files as part of the installation. The pack files start with the name of the vendor such as Amazon, ARM, SEGGER, etc. All the Renesas pack files start with the vendor’s name as Renesas. The file name also contains the features and a version associated with the pack.

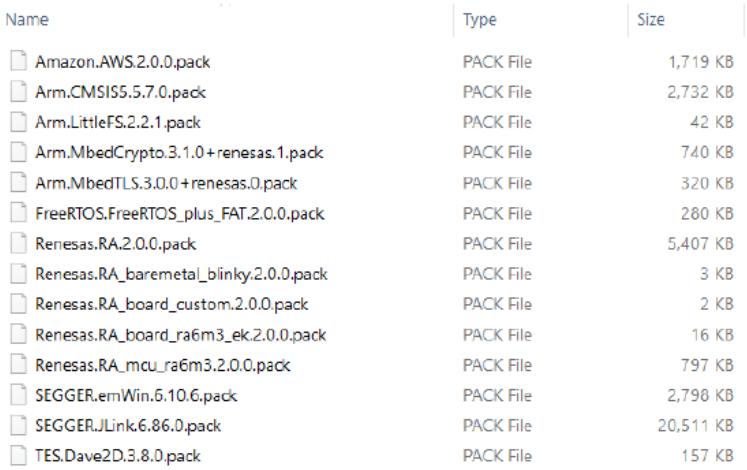

Figure 1. FSP Pack Files

The pack file typically contains the package description ***.pdsc*** file at the root, which is an XML file describing the content. The FSP pack also contains a software component under subfolders which may contain:

- Source code, header files, and software libraries
- Source code templates
- Device parameters along with startup code and programming algorithms

2.2.2 User Creatable FSP Packs

Users can create packs to support user-defined modules in addition to those available from the FSP. For example, if a company wants to create a custom board representing their microcontroller-based product, then a BSP can be created, verified, and then distributed to application developers in order to speed up development. In the case of custom communication modules where support is not available in the FSP, a separate pack file can be created to accelerate work with the new module.

2.2.3 User Pack Creation Tools

Pack creation can be done in different ways. It can be created through the integrated pack creation utility within e^2 studio, or manually modifying the existing FSP packs for the new module. This document describes the step-by-step method as outlined in section 1.

3. FSP Module File Structure

The first step in making an FSP pack is to create the file structure. FSP packs contain driver files that are added to a project when project content is generated.

The following sections explain the hierarchy of middleware/Pmods in e^2 studio and how to replicate the structure for the user's desired FSP pack.

3.1 HS300X e^2 studio File Structure

When the HS300x FSP pack is added to e^2 studio and the content is generated through the ***configuration.xml***, the HS300x source and header files are added to the project directory as shown in Figure 2. These files are contained inside the HS300x FSP pack in the same hierarchy.

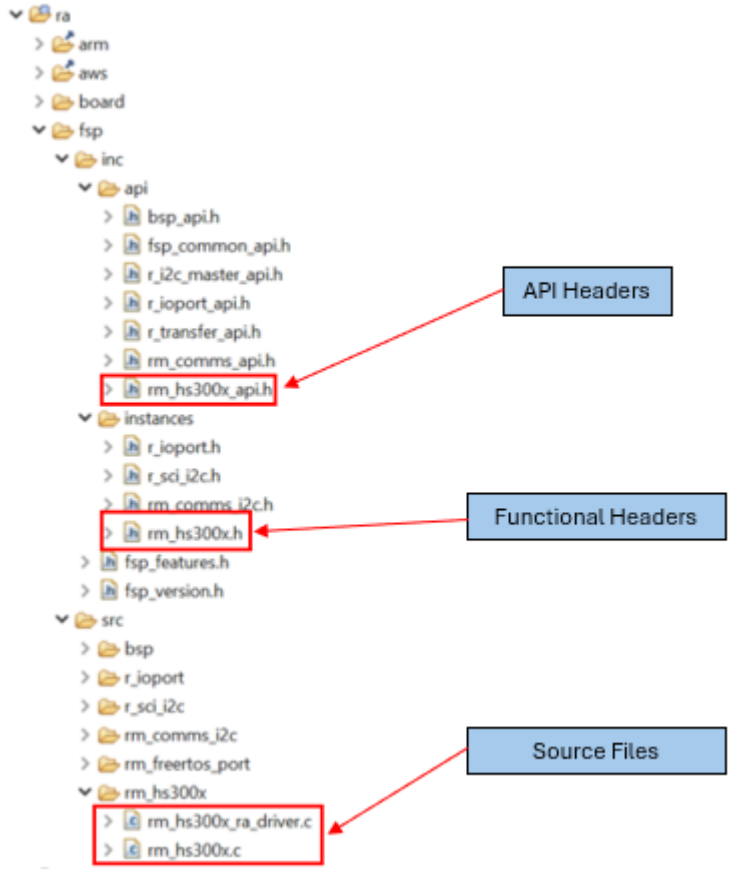

Figure 2. HS300x File Hierarchy

3.2 Creating the FSP File Structure

To begin creating the middleware/Pmod pack create a folder named ***Renesas.SST_rm_your_module_name.#.#.#*** where the '#' indicates the FSP version (does not need to match current FSP release version).

Note: The version number you choose must stay consistent throughout the steps in sections 4, 5, 6, and 7.

Next, create the following folders inside the directory you made above:

- .module_descriptions
- ra

Create the ***Renesas.SST_rm_your_module_name.pdsc*** and ***toolingSupport.xml*** files. The contents of these files will be discussed in sections 4 and 5, respectively.

The resulting folder should look like the directory in Figure 3.

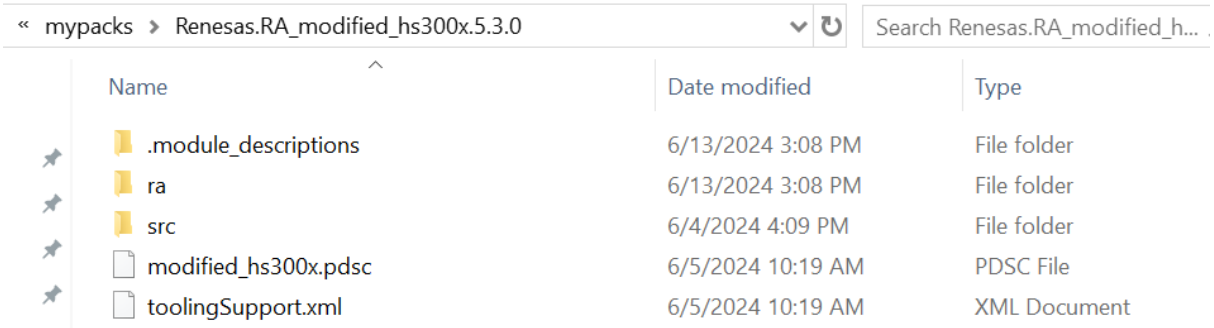

Figure 3. FSP Root Directory Contents

3.2.1 Adding the Source and Header Files to the Pack

Now add the source and header files to the ***ra*** directory you created. For the pack to generate properly, it is important to ensure that the files are placed identically to the structure shown in Figure 2.

- API header is placed in ***ra->fsp->inc->api***
- Middleware header is placed in ***ra->fsp->inc->instances***
- The middleware driver and source code are placed in ***ra->fsp->src->rm_your_module_name***

Note: The file structure is dependent on the sensor/Pmod you are using. For help determining the proper file structure for your custom pack, generate the code for a similar existing Pmod/sensor in e^2 studio and use an identical file structure to what is generated in the ***ra*** folder.

4. Creating the .pdsc File

The *.pdsc* file describes the hierarchy of your FSP module along with important FSP information such as the version number and module classification. The example HS300x *.pdsc* is displayed in Figure 4.

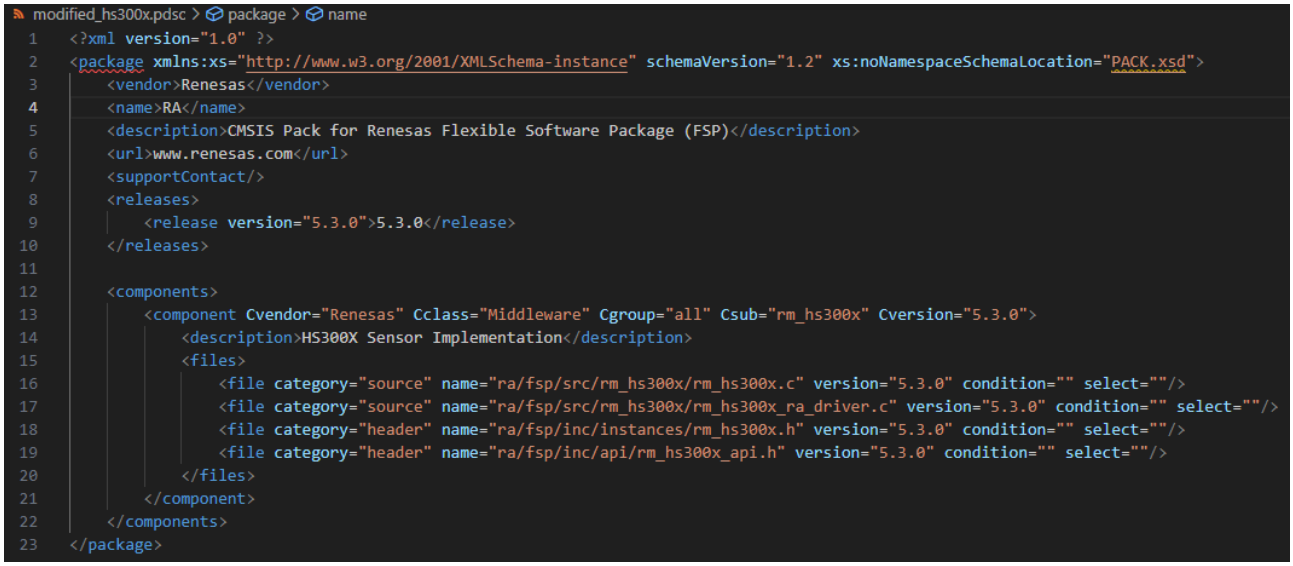

Figure 4. HS300x (.pdsc) File

4.1 Modifying the .pdsc

Open the *Renesas.SST_rm_yourModuleName.pdsc* file you created in section 3.2 into your preferred code editor. Copy the code snippet below and replace the bold text to fit your module.

```
<?xml version="1.0" ?>
<package xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.2" xs:noNamespaceSchemaLocation="PACK.xsd">
    <vendor>Renesas</vendor>
    <name>RA</name>
    <description>CMSIS Pack for Renesas Flexible Software Package (FSP)</description>
    <url>www.renesas.com</url>
    <supportContact/>
    <releases>
        <release version="5.3.0">5.3.0</release>
    </releases>

    <components>
        <component Cvendor="Renesas" Cclass="Middleware" Cgroup="all" Csub="rm_hs300x" Cversion="5.3.0">
            <description>HS300X Sensor Implementation</description>
            <files>
                <file category="source" name="ra/fsp/src/rm_hs300x/rm_hs300x.c" version="5.3.0" condition="" select=""/>
                <file category="source" name="ra/fsp/src/rm_hs300x/rm_hs300x_ra_driver.c" version="5.3.0" condition=""
select=""/>
                <file category="header" name="ra/fsp/inc/instances/rm_hs300x.h" version="5.3.0" condition="" select=""/>
                <file category="header" name="ra/fsp/inc/api/rm_hs300x_api.h" version="5.3.0" condition="" select=""/>
            </files>
        </component>
    </components>
</package>
```

The version numbers in the code above should be changed to the version number you chose in section 3 (i.e., 1.0.0, this does not need to match the current release version of the FSP). You can safely ignore the ***xml version*** and ***schemaVersion*** attribute.

If your FSP module has more (or less) files than the HS300x example, add/remove files in the ***<file> </file>*** header as needed. However, straying from the standard programming style and file hierarchy may create problems when adding the FSP pack to e^2 studio.

4.2 Important Notes when Making the .pdsc

If you change ***Cvendor***, ***Cclass***, ***Cgroup***, ***Csub***, or ***Cversion*** then the module description file, explained in section 6, must be named accordingly. For example, the HS300x module description file name is ***Renesas##Middleware##all##rm_hs300x####5.3.0.xml***.

Important: The naming convention is ***Cvendor##Cclass##Cgroup##Csub####Cversion.xml***. This step is critical for e^2 studio to use your FSP pack correctly. If formatted incorrectly, the pack will not show up in the Configuration editor in e^2 studio.

5. Creating the Tooling Support File

The tooling support file indicates the necessary toolchains required for your module and the location of your module description file. The HS300x ***toolingSuppot.xml*** file is shown in Figure 5.

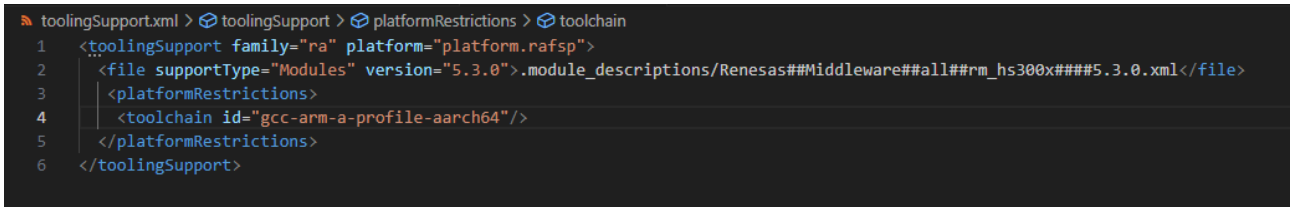

Figure 5. HS300x ToolingSuppot.xml File

Open the “toolingSupport.xml” file that you created in section 3.2. Copy the following code to your file. Change ***Renesas##Middleware##all##rm_hs300x####5.3.0.xml*** to the name of the module you intend to create. The name should look like ***Renesas##Middleware##all##rm_your_module_name####1.0.0.xml***. Replace the version numbers with the version number you chose in section 3.

```
<toolingSupport family="ra" platform="platform.rafsp">
  <file supportType="Modules" version="5.3.0">.module_descriptions/Renesas##Middleware##all##rm_hs300x####5.3.0.xml</file>
   <platformRestrictions>
    <toolchain id="gcc-arm-a-profile-aarch64"/>
  </platformRestrictions>
</toolingSupport>
```

6. Quick Module Configuration Setup

This section discusses the creation/modification of the module description *.xml* file. The module description file has the following functionality:

- Set module/sensor configuration
- Establish and configure resource dependencies (i.e., rm_comms layer)
- Developer Assistance code

This section is divided into two parts: the first covers how to find existing FSP module description files for quick modification (section 6.1), and the second discusses how to quickly modify an existing FSP module description file to work with your module (section 6.2). For an in-depth explanation of the module description file, see Appendix A.

Note: As mentioned earlier, this document uses the HS300x to explain the functionality of the module description file. When creating a custom pack it is highly recommended to pull a module description file from an existing sensor/module that is similar to the one you are creating (e.g., for a sensor Pmod that uses I^2C, the HS300x would be a good choice to modify).

6.1 Locating Module Description Files

The simplest way to create a module description file is to modify an existing module description file.

6.1.1 Navigate to FSP Installation

To find existing module description files, first navigate to your installation of e^2 studio. To find the correct directory, in e^2 Studio, go to ***help->about e^2 Studio->Installation Details->Support Folders***. This is displayed in Figure 6, Figure 7, and Figure 8.

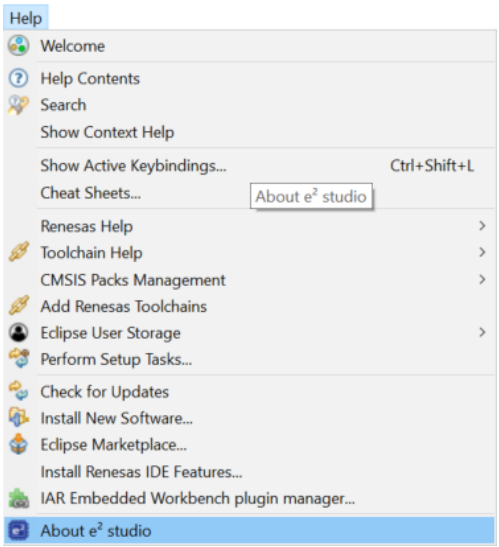

Figure 6. Help Window

Figure 7. About e² Studio

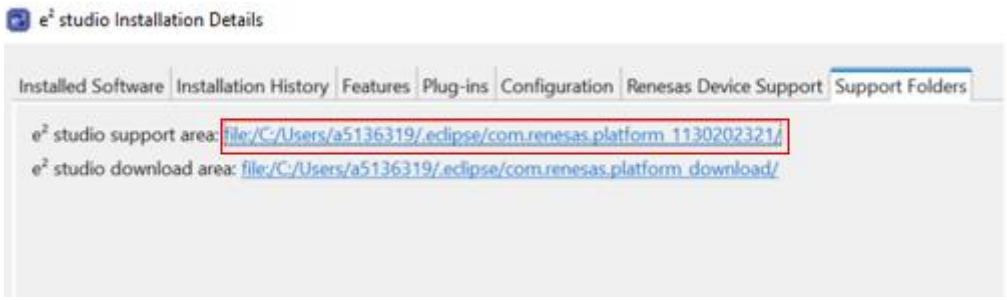

Figure 8. Current e² Studio FSP Reference

Navigate to the packs folder as follows, e2_studio->internal->projectgen->ra->packs.

6.1.2 Extract FSP Pack

The packs folder contains all FSP packs for the current FSP release version. With the current release being 5.3.0, locate ***Renesas.RA.5.3.0.pack*** and extract it to an arbitrary empty folder.

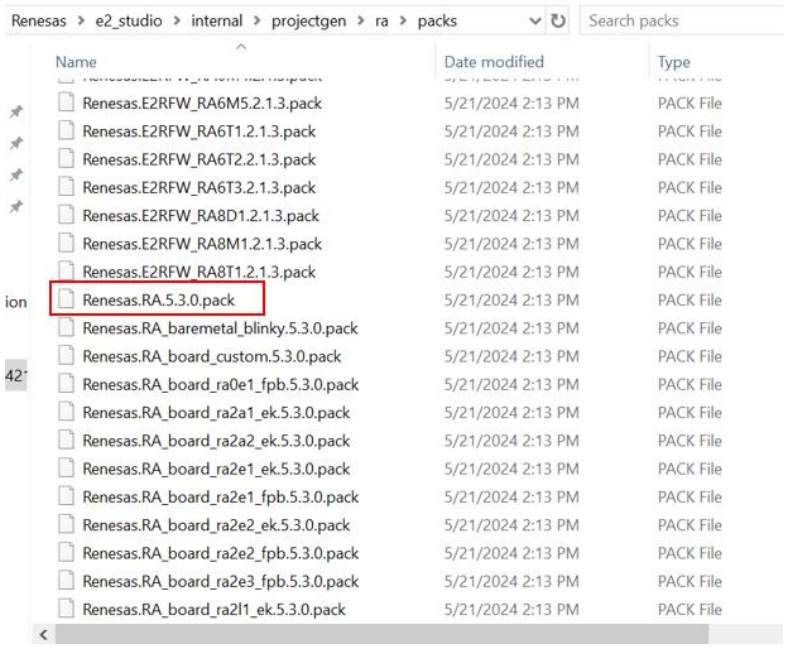

Figure 9. RA FSP Pack

After extracting the pack, the contents should be as displayed in Figure 10.

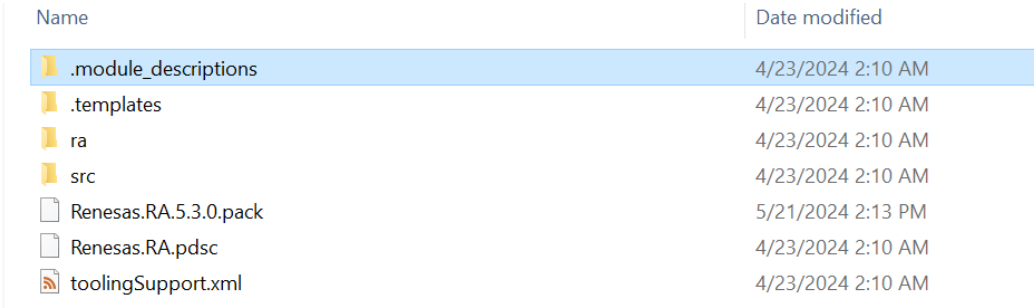

Figure 10. Pack Extraction Contents

6.1.3 Locating Desired Module .xml

Inside ***.module_descriptions*** you will find all the module description files for the current release version of the FSP. This document will copy and modify the HS300x ***.xml*** file as shown in Figure 11.

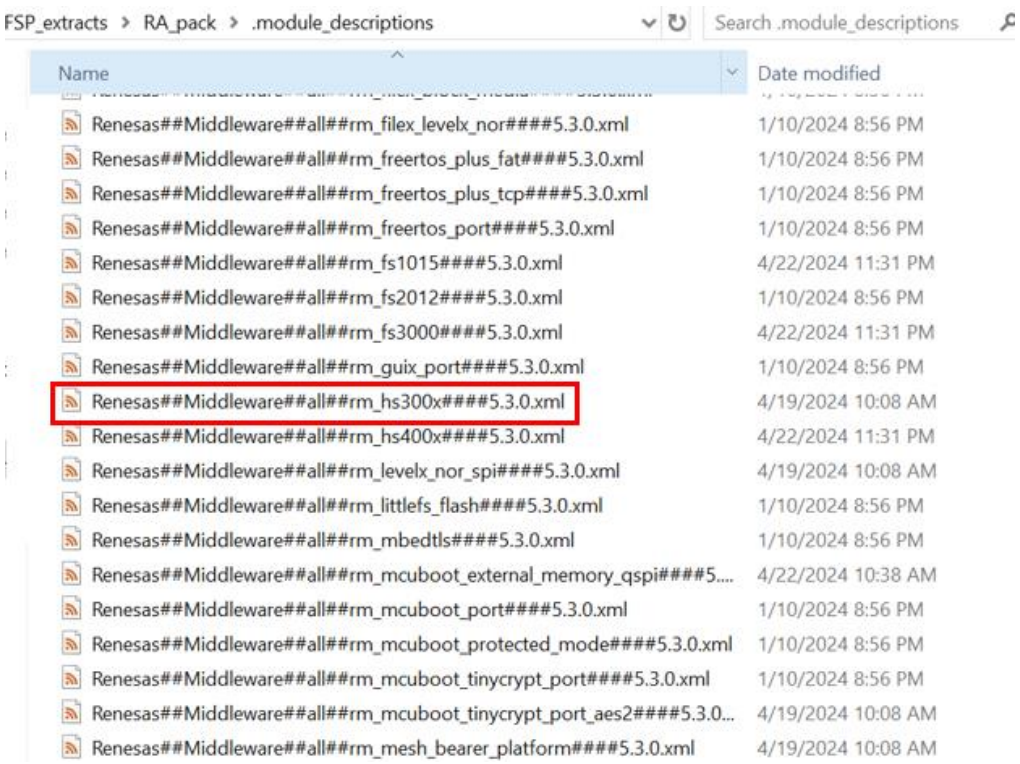

Figure 11. HS300x *.xml* Location

Copy the desired ***.xml*** file to the ***.module_descriptions*** folder created in section 3.2. Change the name of this file to ***Renesas##Middleware##all#rm_yourModuleName####5.3.0.xml***. 5.3.0 should be changed to match the version number chosen in section 3.

6.2 Quick Module Description Setup

After the description file is copied to the ***.module_descriptions*** folder, you must modify it to work with the sensor/Pmod. This section explains how to quickly modify the module description file to suit the target device. For a detailed explanation of the module description file and how to modify each part, see Appendix A, section A.

If using the HS300x module description file ***.xml***, I2C support is implemented. This section will show how to modify the necessary parameters to obtain functionality with your device but will not explain every detail of the I2C configuration. For details on how this is implemented see Appendix section A.2.

6.2.1 Replacing the Module Name

Open the module description file you copied in section 6.1 in your preferred code editor.

Find and replace the ***current_module_name*** with ***your_module_name***. For example, Figure 12 demonstrates replacing ***hs300x*** with ***isl28022*** using Visual Studio Code. You can also use (Ctrl+H) to replace ***hs300x*** with ***your_module_name***.

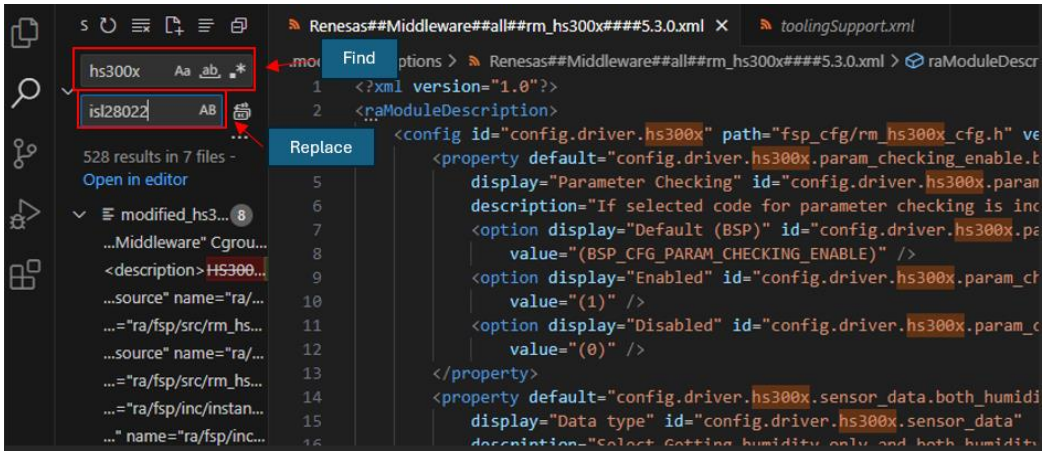

Figure 12. Find and Replace Module Name

6.2.2 Module Description Property Attributes

Inside the ***<config>*** elements, ***<property>*** elements can be used to add user configurable flags to enable/disable options related to the device. For example, the HS300x has flags for parameter checking, programming mode, and temperature/humidity selection.

These flags are referenced in the sensors firmware to enable or disable certain features. The ***id*** attribute is used as a placeholder for inserting code into the code template in ***<content>*** sections. For information, see Appendix section A.1.2.

If the device does not require any user configurable options such as programming mode or parameter checking, then blocks 1 and 2 in the following figure can safely be deleted.

DO NOT remove the ***<config>*** or the ***<content>*** elements; only remove the ***<property>*** elements and the corresponding ***#define*** between the ***<content>*** elements.

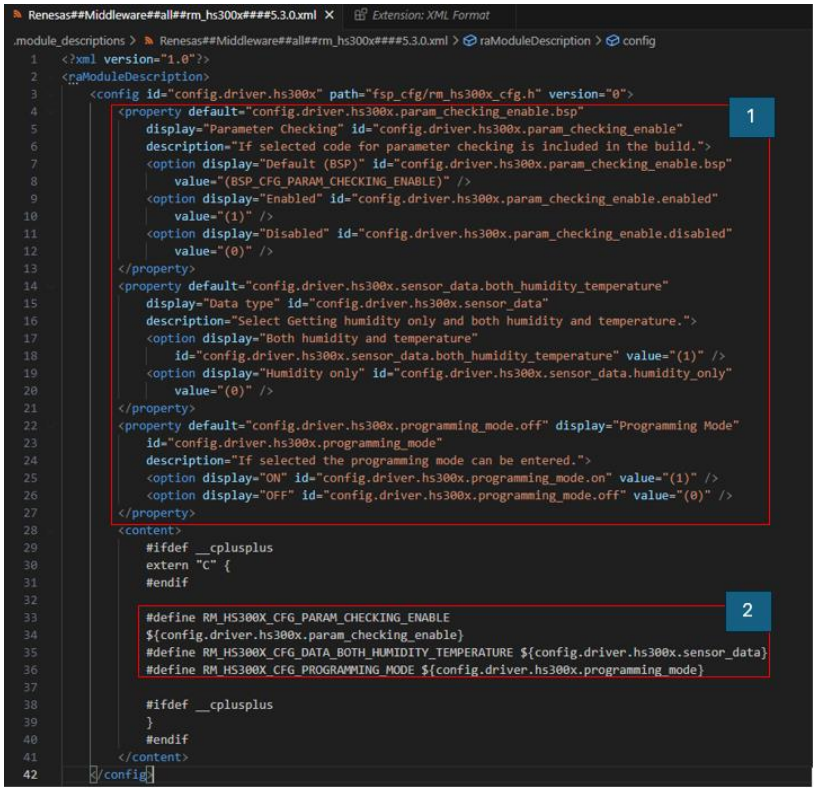

Figure 13. Module Description *.xml* Config Section

6.2.3 Module Naming and I2C Configuration

Figure 14 shows the module description code that configures the name given to your module in the e² studio ***configuration.xml*** stack window.

Block 2 in Figure 13 shows the name given to the stack when the FSP is added to the e² studio ***configuration.xml***. Edit “Temperature/Humidity Sensor” to describe your Pmod.

Change the value shown in block 3 of Figure 14 to the slave address of your sensor/Pmod.

Figure 14. Module Description(.xml) Module Naming

6.2.4 Developer Assistance

Developer Assistance allows users to drag and drop pre-defined functions into their project in e² studio. In this section Developer Assistance is removed to simplify the FSP pack creation process. However, Developer Assistance is highly recommended as it provides enhanced usability to a sensor/Pmod during application development.

Appendix section A.3 provides a detailed look into Developer Assistance and describes how to add it to a custom FSP pack.

To remove Developer Assistance, delete the ***<developerSupport>*** elements and the content within these elements.

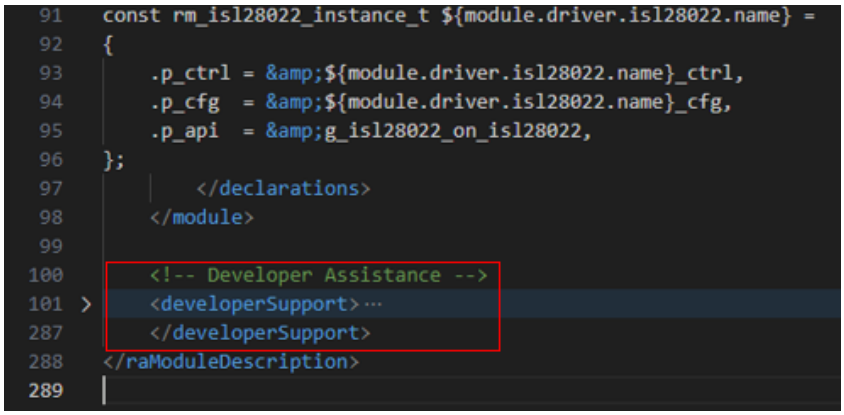

Figure 15. Developer Assistance

The module description file is now ready to be used with your FSP pack. To add Developer Assistance to the FSP pack and an overall more detailed explanation of this file, see Appendix A; otherwise, continue to section 7.

7. Creating and Importing the .pack File

FSP packs are contained inside *.pack* files. This section describes how to create the *.pack* file and how to import your *.pack* file into e^2 studio. Once you have completed the steps outlined in sections 3 to 6, you are ready to create the *.pack* file and import into e^2 studio.

7.1 Creating the .pack file

With your FSP directory completed, zip the directory into a *.pack* file. This document shows the use of 7-zip to create the *.pack* file. Select all of the files in the pack directory you created in section 3, as displayed in Figure 16. Select ***7-zip->Add to archive…***.

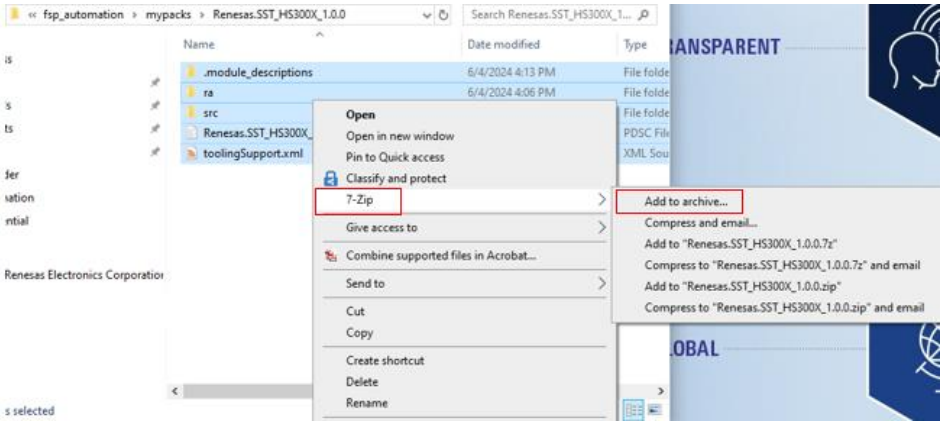

Figure 16. Zipping the Custom FSP Pack

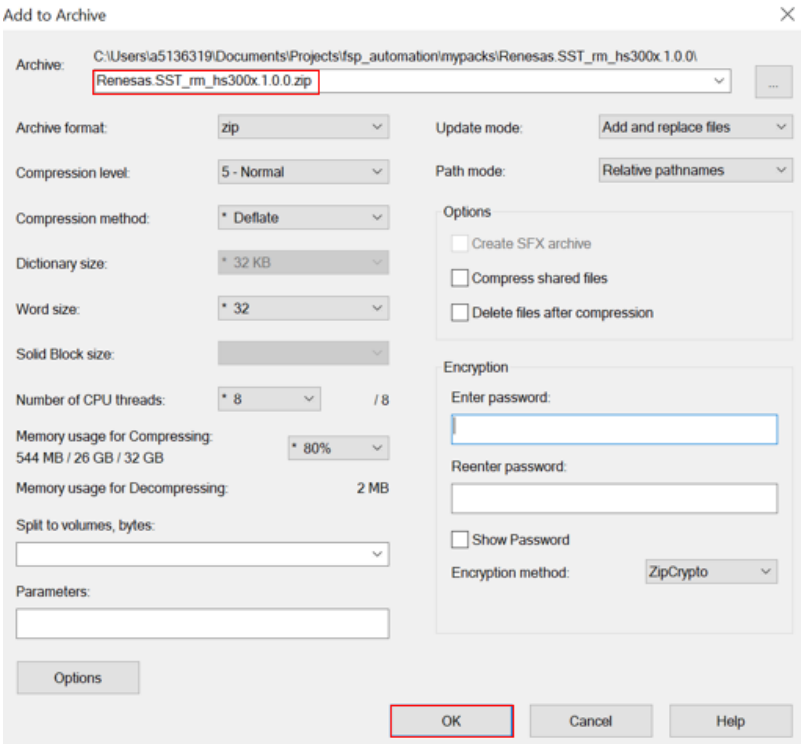

Figure 17. Zip Settings

Note: The name of your *.pack* file should match the following format ***Renesas.GROUP_yourModuleName.1.0.0.pack***, where **GROUP** represents who made the pack, and **1.0.0** represents the version numbers you chose in section 3.

When zipping your FSP pack ensure your settings match the settings in Figure 17. Change the file ending from *.zip* to *.pack*, then click **OK**.

7.2 Importing the Pack File

1. Open e^2 studio and click *File -> Import…*

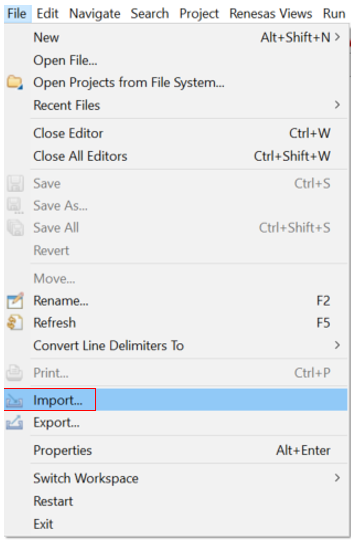

Figure 18. e^2 Studio Import Window

2. Select CMSIS Pack.

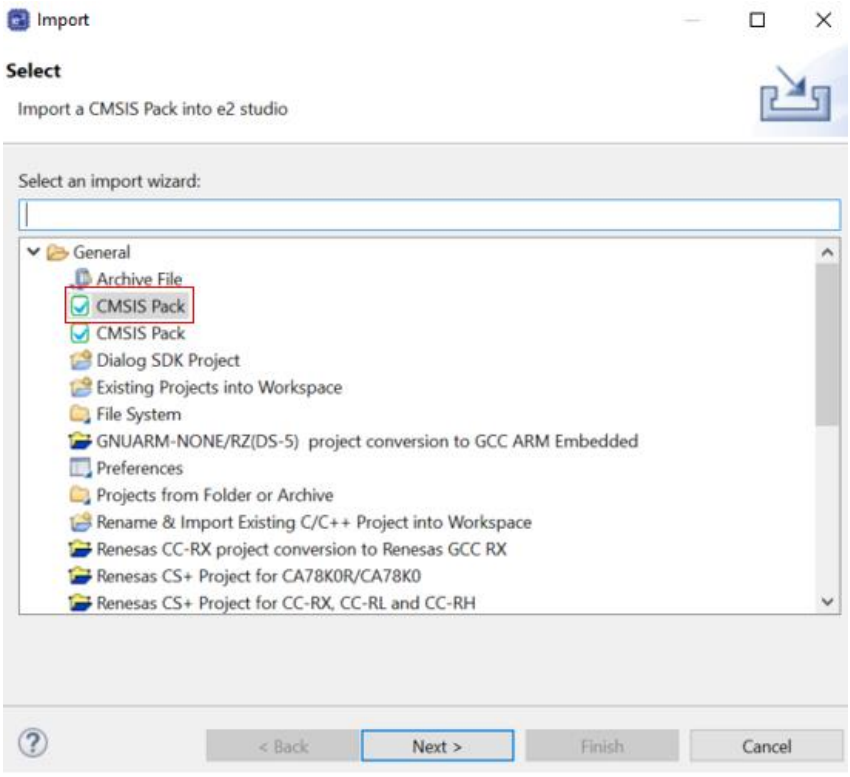

Figure 19. Import Type Selection

3. Specify your FSP *.pack* file location and the device family to which your pack applies to.

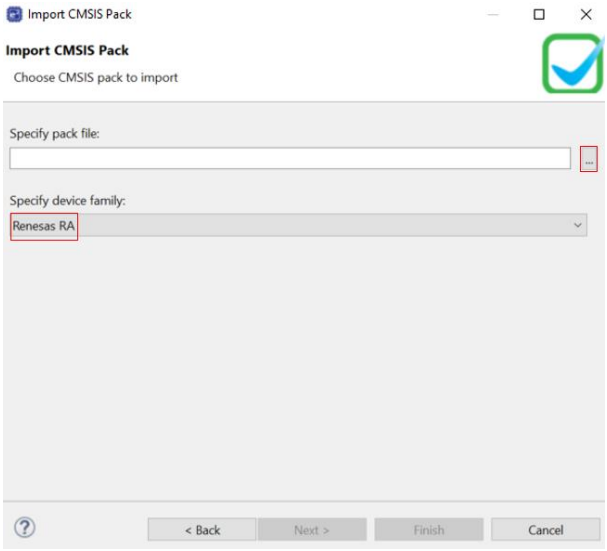

Figure 20. Pack Location and Device

4. Select your *.pack* file and click **open**.

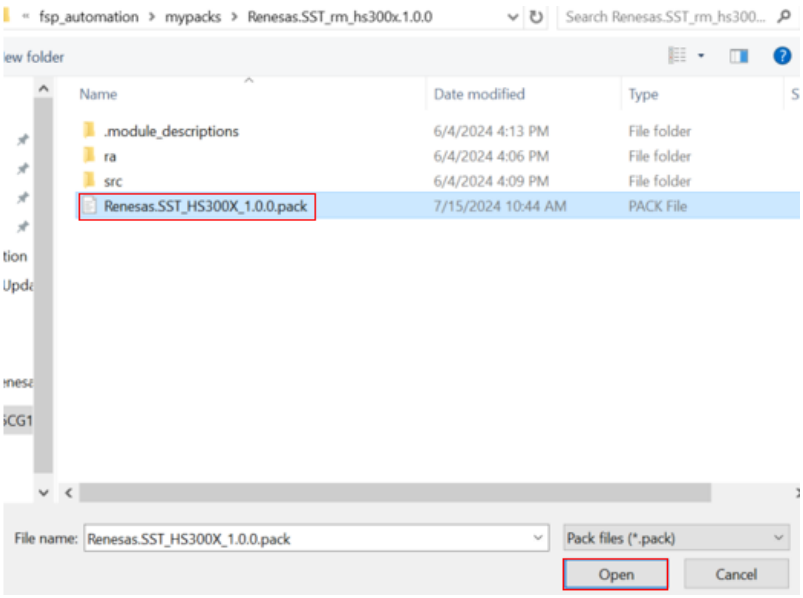

Figure 21. Select the FSP (.pack)

5. Click **Finish** and you should see the following message from e^2 studio.

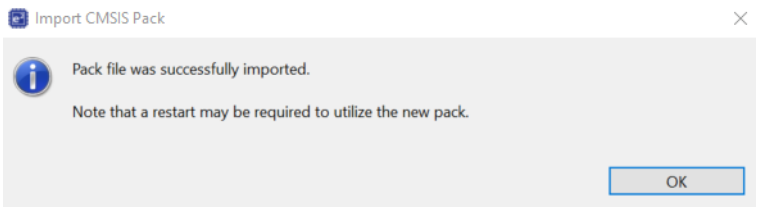

Figure 22. e^2 Studio Successful FSP Import

6. Close and re-open e^2 studio.

7.3 Verifying Pack Import

To verify the pack imported correctly, open a new project and check the ***Stacks*** window in the ***configuration.xml.*** The pack should be visible in the ***New Stack >*** selector under the category you selected in section 6.2.3 as shown in Figure 23.

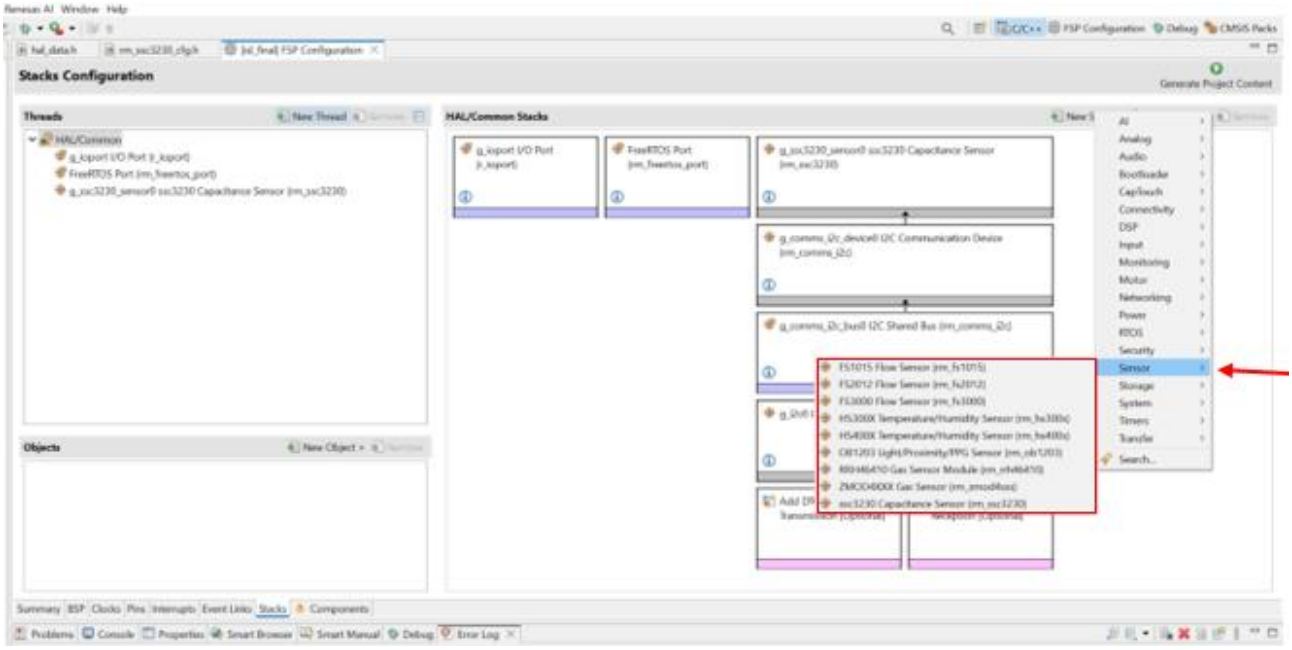

Figure 23. Stack Window Pack Verification

7.3.1 Pack Import Failure

If the pack does not appear in the ***Stacks*** window then there is an error in the FSP pack. Listed below are a few common errors that may occur when creating a custom FSP pack. Any error, such as spelling or syntax, in any of the three FSP pack support files will prevent the pack from showing up in the stack window in e^2 Studio:

- Check the ***.pdsc*** created in section 4.
  - Ensure that your file paths are typed in correctly and the file names match exactly what they are named in ***ra*** folder.
  - Check section 4.2, ensuring your module description ***.xml*** adheres to the naming convention shown.
  - Verify the version numbers all match the number you chose in section 3.
- Check the ***toolingSupport.xml*** created in section 5.
  - Make sure the version numbers match the number you chose in section 3.
  - Check the module description ***.xml*** name matches the name of your module description ***.xml***.
- Check the module description ***.xml*** created in section 6.
  - Verify the spelling of your sensor/pmod module name and ensure it stays consistent throughout the file.
  - Check each section you modified, looking for spelling or syntax errors.
- Use the Error Log in e^2 studio to help diagnose problems
  - To open go to Window > Show View > Other… > General > Error Log

7.3.2 Resetting the e^2 Studio FSP directory

If your pack fails to show up in the ***Stacks*** window, before you re-upload your updated/debugged FSP pack, the e^2 Studio FSP directory must be reset. This section explains the steps to reset the e^2 Studio FSP directory.

Navigate to the ***.eclipse*** directory. The file path should be ***C:\Users\Username\.eclipse***. The contents of this directory should be like those shown in the following figure.

Name	Date modified	Type
com.renesas.platform_271177703	7/10/2024 4:39 PM	File folder
com.renesas.platform_922758438	6/11/2024 4:53 PM	File folder
com.renesas.platform_1130202321	7/31/2024 4:45 PM	File folder
com.renesas.platform_1130202321.backup	5/21/2024 4:42 PM	File folder
com.renesas.platform_download	5/21/2024 4:42 PM	File folder
org.eclipse.equinox.security	5/21/2024 4:14 PM	File folder
org.eclipse.oomph.p2	8/19/2024 11:52 AM	File folder
org.eclipse.oomph.setup	5/21/2024 4:14 PM	File folder

Figure 24. .eclipse Directory

Each of the ***com.renesas.platform_########*** directories contains the FSP reference for a specific instance of e^2 Studio. If you have multiple e^2 Studio instances then there will be multiple FSP reference directories, one for each e^2 Studio instance.

These directories are generated upon launching e^2 Studio from a main reference directory tied to each e^2 Studio instance.

To find the correct directory, in e^2 Studio, go to ***help->about e^2 Studio->Installation Details->Support Folders***. Shown below in Figure 25, Figure 26, and Figure 27.

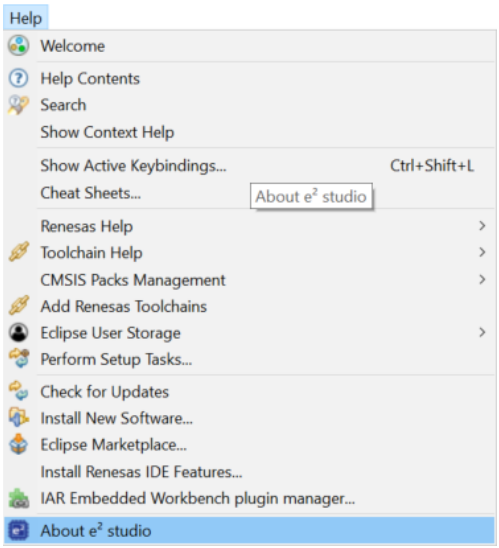

Figure 25. Help Window

Figure 26. About e² Studio

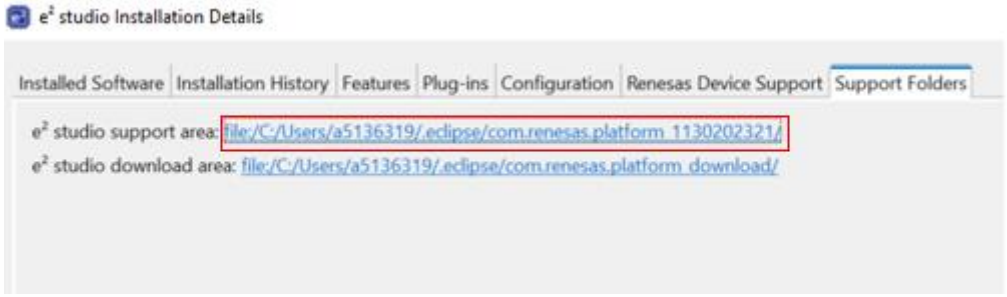

Figure 27. Current e² Studio FSP Reference

Delete the FSP reference tied to the e² Studio instance in which you want to add your custom FSP pack.

8. References

- Flexible Software Package (FSP) | Renesas
- RA Flexible Software Package Documentation: FSP Architecture (renesas.github.io)

9. Website and Support

Visit the following URLs to learn about key elements of the RA family, download components and related documentation, and get support:

- RA Product Information
- RA Product Support Forum
- RA Flexible Software Package
- Renesas Support

10. Revision History

Revision	Date	Description
1.00	Dec 12, 2024	Initial release.

A. Detailed Look into Module Description Files

This appendix describes the functionality of the module description file and explains each section of code and its correlation to e² studio. It is divided into three parts to cover the three basic functions of the module description *.xml* file and how to modify them:

- Module Configuration
- Resource (comms) Configuration and Naming
- Developer Assistance

A.1 Module Common Configuration

This section discusses the module description code related to user build-time configuration options for *fsp_cfg* header generation in e² studio.

Figure 28 and Figure 29 show the user configuration code in the module description file and its corresponding output in e² studio.

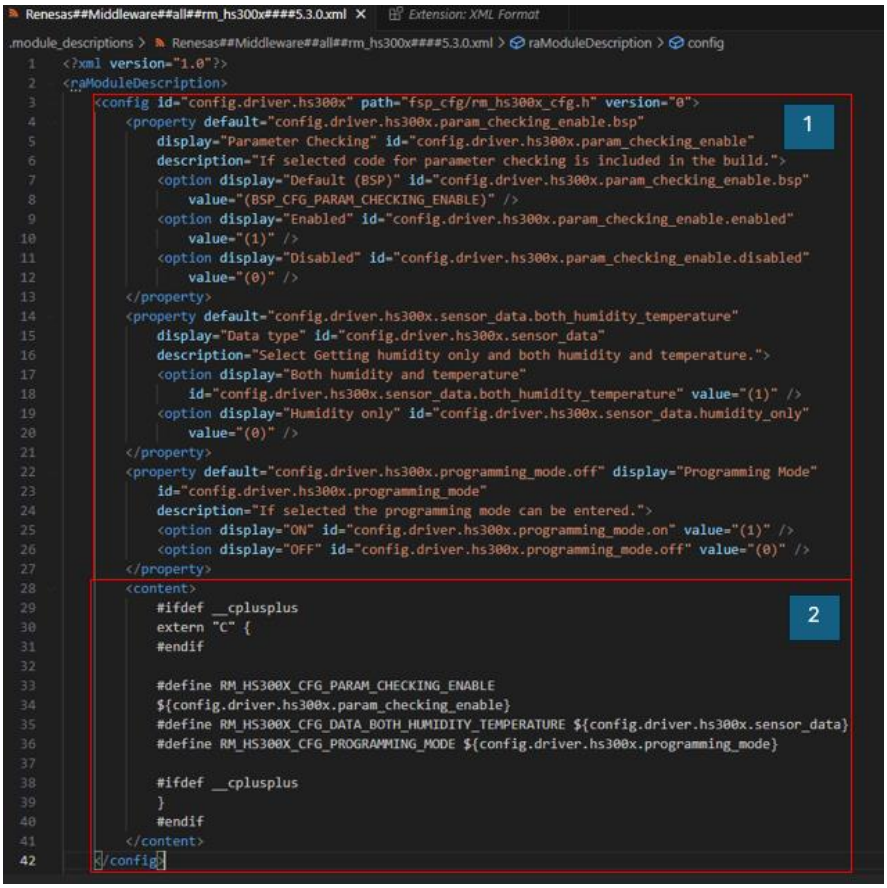

Figure 28. (.xml) Module Configuration

Figure 29. e² Studio Module Configuration

A.1.1 Configuration <config> Element

The **<config>** element identifies the following code as configuration settings. Inside the **<config>** elements are **<property>** and **<content>** elements. The **<config>** element contains attributes ***id***, ***path***, and ***version***.

id="config.driver.hs300x" specifies the internal ID that will be used to track the config. When modifying or creating your ***.xml*** change ***config.driver.hs300*** to ***config.driver.your_module_name***.

The ***path*** attribute identifies where the config file will be generated. In this case, when the module is selected in e^2 studio and the ***Generate Project Content*** button is pressed, ***rm_hs300x_cfg.h*** will be generated under the ***fsp_cfg*** directory.

For your module, change ***rm_hs300x_cfg.h*** to ***your_module_name_cfg.h***. The contents of the ***rm_hs300x_cfg.h*** are contained inside the **<content>** elements shown in block 2 of Figure 28. This section will be further explained the Content Element section below.

Version can be left as version="0".

A.1.2 Configuration <property> Element

The **<property>** elements create a configurable attribute that can be set in e^2 studio by the user. In the HS300x example, parameter checking, data type support, and programing mode are configured in block 1 of Figure 28. The display in e^2 studio is shown in Figure 29.

Each element will have a ***default***, ***display***, ***id***, and ***description*** attribute, as well as **<option>** elements if necessary. When creating an element:

- Set default="config.driver.your_module_name.property_name.default_value_name" where default_value_name is the id of the option you want assigned as default.
- Set ***display="Name of option"***, this is the property name that will appear in e^2 studio.
- Set ***id="config.driver.your_module_name.property_name"***, where ***property_name*** is the name of the property you want the user to set.
- The ***description=""*** attribute provides the user with a description of what this attribute does in relation to the module/Pmod. The description is visible in e^2 studio when hovering over the attribute.
- Inside the **<property>** elements the **<option>** elements are placed. These indicate the different values that can be chosen for each property. Each option has a ***display***, ***id***, and ***value*** attribute. These are set similarly to the property attributes as shown in block 1 of Figure 28.

Modify/create the user-selectable properties necessary for your module function inside the **<config>** element. Your module middleware source code should include the ***your_module_name_cfg.h*** file where these flags are required.

A.1.3 Configuration <content> Element

The **<content>** elements identify the code that will be written to the ***your_module_name_cfg.h*** file. The content section of the HS300x is shown in block 2 of Figure 28. Create a #define for each property you added. Use the property id name to assign the associated value to the #define you create. Figure 30 shows the resulting ***fsp_cfg*** file generated for the HS300x.

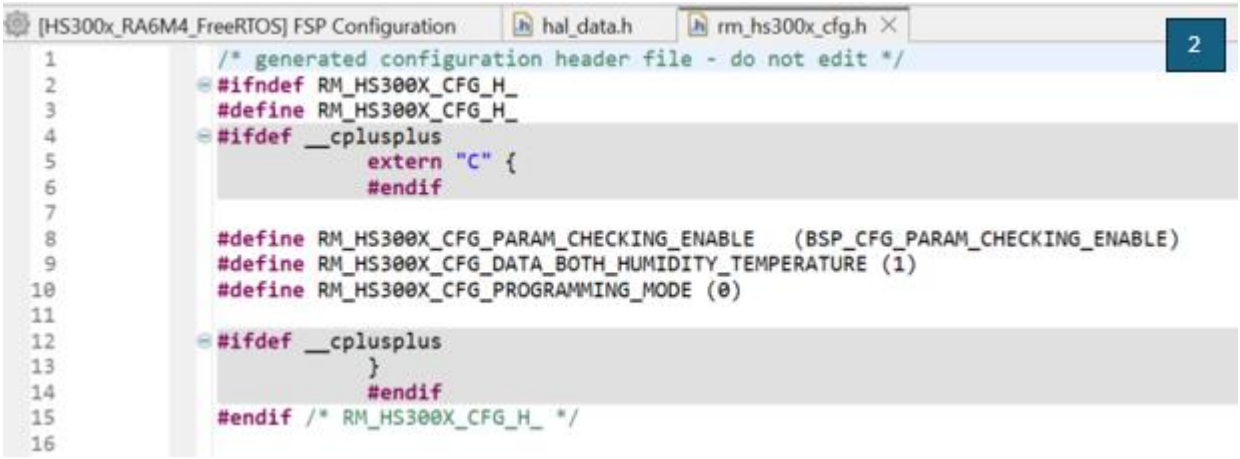

Figure 30. e² Studio Generated Configuration File

A.2 Instance Configuration

The next part of the module description ***.xml*** configures any resource requirements, such as I²C, and generates the necessary callback parameters for your module. This section also configures the displayed name of your module and its location in the ***New Stack*** panel in e² studio's ***configuration.xml*** file.

Figure 31, Figure 32, Figure 33, and Figure 34 show the HS300x module description code related to resource configuration and naming as well as the corresponding e² studio output.

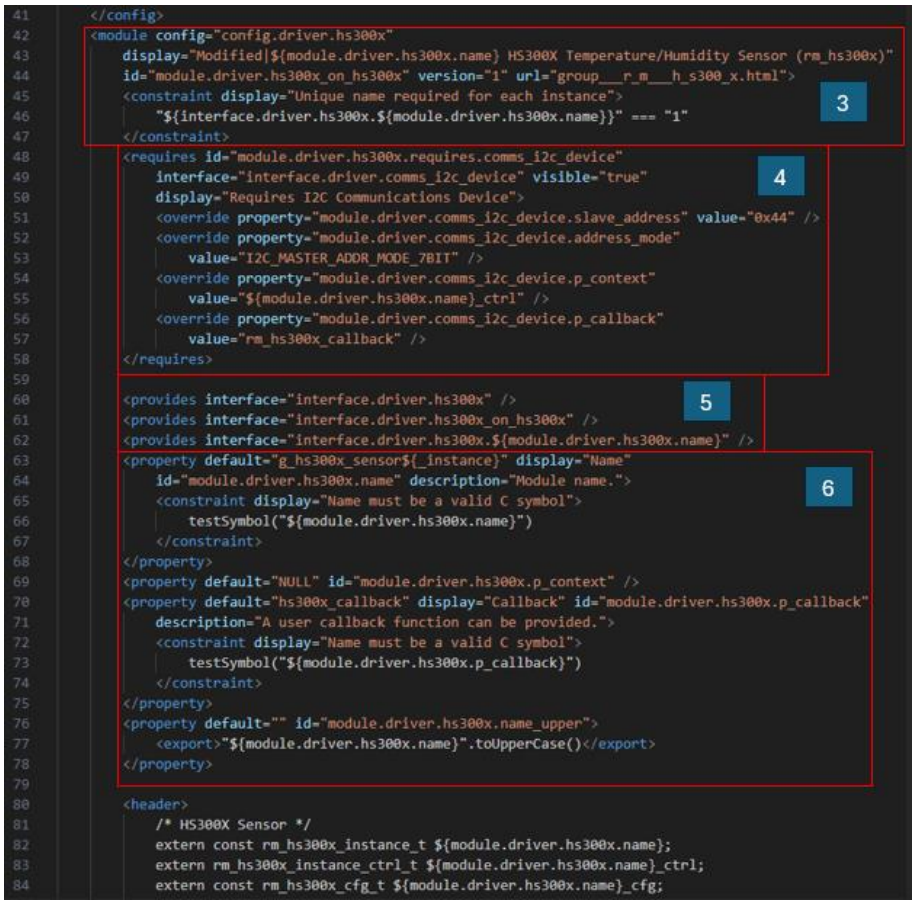

Figure 31. HS300x Module Description Resource Configuration Code

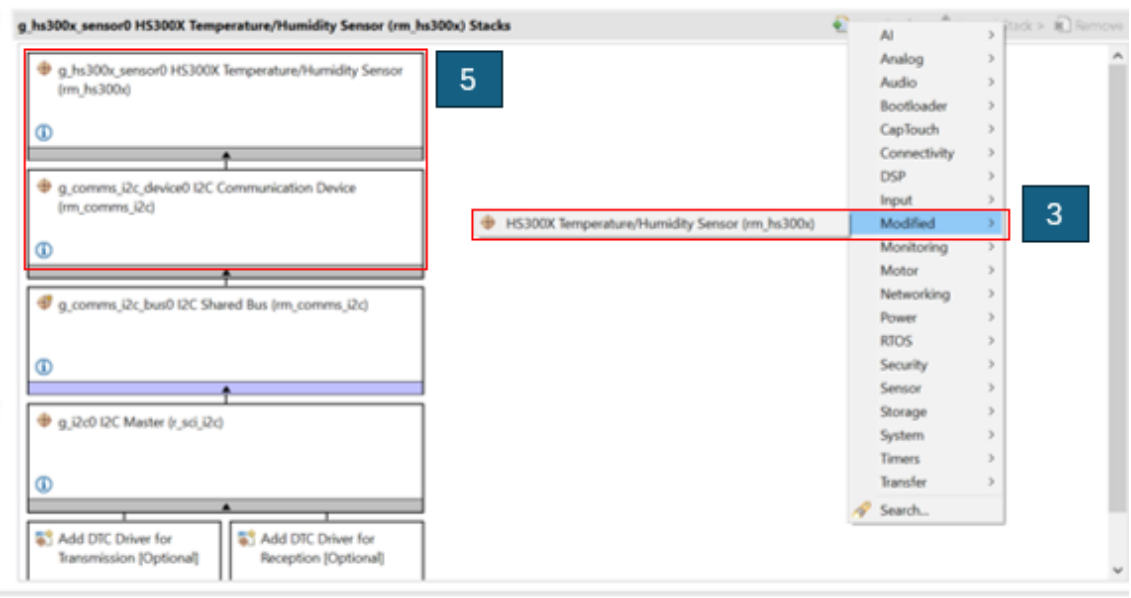

Figure 32. e² Studio Stack Selection GUI

A.2.1 Resource Configuration <module> Element

The resource and naming configuration code is contained within the ***<module>*** element as seen in block 3 of Figure 31. The module element has the following attributes: ***config***, ***display***, ***id***, ***version***, and ***url***. To work with your module these attributes should be modified as follows:

- Set ***config="config.driver.yourModuleName"***, this should be the same name as the id you created in the ***<config>*** element in A.1.
- The display attribute tells e² studio where to store the FSP module in the stack window. In the HS300x example, ***display="Modified | ${module.driver.hs300x.name} HS300X Temperature/Humidity Sensor (rm_hs300x)"*** tells e² studio to place the module under the ***Modified*** tab and names it accordingly as seen in Figure 32. If your module is a sensor then change ***Modified*** to ***Sensor***, otherwise identify the proper section your module should be under and name it accordingly. Change ***module.driver.hs300x.name*** to ***module.driver.your_module_name.name***. Finally, modify the remaining text to a name that suits your module/sensor.
- Change the id attribute to id="module.driver.your_module_name_on_your_module_name".
- Version represents the current version of the FSP module; this can be left as 1 for now.
- The ***url*** generates a link to the FSP module website which provides detailed information on the FSP module. This link can be seen in Figure 32, denoted by the blue "*i*" inside the blue circle in the stack window. This attribute can be removed until an information site is created for your module.

The following elements are located within the ***<module>*** element.

A.2.2 Resource Configuration <constraints> Element

The <constraint> element for this use case simply checks that each new instance of the module has a unique name. Change "${interface.driver.hs300x.${module.driver.hs300x.name}}" === "1" to "${interface.driver.your_module_name.${module.driver. your_module_name.name}}" === "1".

A.2.3 Resource Configuration <requires> Element

The ***<requires>*** element configures the resources required by your module. In the HS300x example, the HS300x requires I²C to function. As seen in block 4 of the following figure, this code tells e² studio a ***comms_i2c_device*** is required for the HS300x module.

The required attributes to configure this section include ***id***, ***interface***, ***visible***, and ***display***. This section also discusses the ***<override>*** element that resides inside the ***<requires>*** element:

- Set the ***id*** to the resource required by your module. For modules such as the HS300x that require an I²C connection, the id is as follows; ***id="module.driver.your_module_name.requires.comms_i2c_device"***
- If your device requires an I²C connection then the ***interface***, ***visible***, and ***display*** attributes can remain as shown in Figure 31, block 4.
- The ***<override>*** elements are used to configure the required resource; in this case, it configures the ***rm_comms_i2c*** layer. The necessary properties to override include the following:
  - The ***slave address*** property value should be set to the slave address of your module/sensor.
  - The ***address_mode*** value should be set to your module/sensors addressing mode.
  - For the .p_context and .p_callback properties change hs300x to your_module_name.

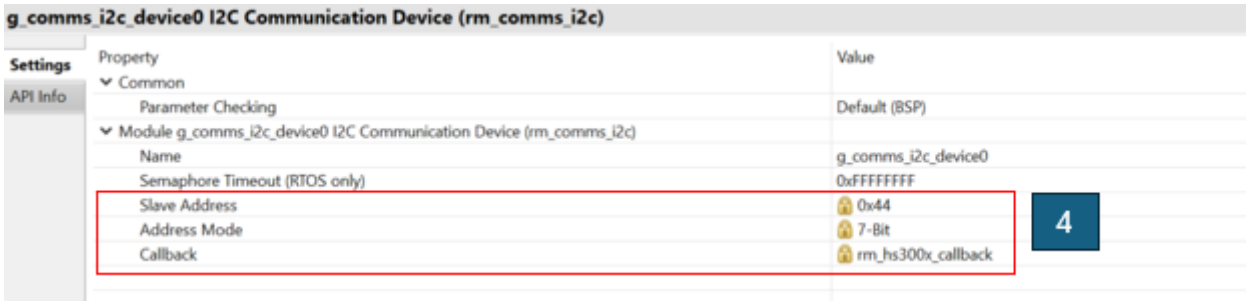

Figure 33. e² Studio I2C Configuration GUI

A.2.4 Resource Configuration <provides> Element

The ***<provides>*** element, as seen in block 5 of Figure 31, simply generates the stacks in the e² studio stack window shown in block 5 of Figure 32. Replace any instance of ***hs300x*** with ***your_module_name***.

A.2.5 Resource Configuration <property> Element

The ***<property>*** elements in this section, shown in block 6 of Figure 31, generate the instance name and callback name when the module is added to the stack configuration window. Figure 34 shows the corresponding e² studio configuration window.

Figure 34. e² Studio Module Configuration

The ***<property>*** elements in this section have the following attributes.

- default – specifies the default value assigned to the associated id.
- display – specifies the name shown in e² studio for the associated property,
- id – a unique identifier to use when referencing the property
- description – creates the text shown when this property is hovered over with the mouse in e² studio. The description also appears in the status bar when the property is selected.

For each of these properties, change any reference to ***hs300x*** to ***your_module_name***.

A.2.6 Resource Configuration Code Generation

This section of the module description file generates the control structure for your sensor/Pmod based on the properties configured above. The code is separated into two sections:

- Header code
- Declaration code

Figure 35 shows the element code in block 7 and the declaration code in block 8. When this code is generated, it allows the sensor/Pmod module to communicate with the rm_comms layer using the existing I2C module.

```
78      </property>
79
80      <header>
81          /* HS300X Sensor */                                          7
82          extern const rm_hs300x_instance_t ${module.driver.hs300x.name};
83          extern rm_hs300x_instance_ctrl_t ${module.driver.hs300x.name}_ctrl;
84          extern const rm_hs300x_cfg_t ${module.driver.hs300x.name}_cfg;
85          #ifndef ${module.driver.hs300x.p_callback}
86          void ${module.driver.hs300x.p_callback}(rm_hs300x_callback_args_t * p_args);
87          #endif
88      </header>
89
90      <includes>
91          #include &amp;quot;rm_hs300x.h&amp;quot;
92          #include &amp;quot;rm_hs300x_api.h&amp;quot;
93      </includes>
94
95      <declarations>                                                   8
96          rm_hs300x_instance_ctrl_t ${module.driver.hs300x.name}_ctrl;
97          const rm_hs300x_cfg_t ${module.driver.hs300x.name}_cfg =
98          {
99          .p_instance =
100         &amp;${module.driver.hs300x.requires.comms_i2c_device::module.driver.comms_i2c_device.name},
101         .p_callback = ${module.driver.hs300x.p_callback},
102         #if defined(${module.driver.hs300x.p_context})
103         .p_context = ${module.driver.hs300x.p_context},
104         #else
105         .p_context = &amp;${module.driver.hs300x.p_context},
106         #endif
107         };
108
109         const rm_hs300x_instance_t ${module.driver.hs300x.name} =
110         {
111         .p_ctrl = &amp;${module.driver.hs300x.name}_ctrl,
112         .p_cfg = &amp;${module.driver.hs300x.name}_cfg,
113         .p_api = &amp;g_hs300x_on_hs300x,
114         };
115     </declarations>
116 </module>
```

Figure 35. I²C Configuration Code

The element code, block 7 of Figure 35, is generated in the element file of the corresponding thread your module is added to in the stack window in e² studio. Figure 36 shows the ***hal_data.h*** file after the HS300x module is added to the stack under ***HAL/Common*** and ***Generate Project Content*** is pressed.

[HS300x_RA6M4_FreeRTOS] FSP Configuration | hal_data.h

```
1   /* generated HAL header file - do not edit */
2   #ifndef HAL_DATA_H_
3   #define HAL_DATA_H_
4   #include <stdint.h>
5   #include "bsp_api.h"
6   #include "common_data.h"
7   #include "rm_comms_i2c.h"
8   #include "rm_comms_api.h"
9   #include "rm_hs300x.h"                                  7
10  #include "rm_hs300x_api.h"
11  FSP_HEADER
12  /* I2C Communication Device */
13  extern const rm_comms_instance_t g_comms_i2c_device0;
14  extern rm_comms_i2c_instance_ctrl_t g_comms_i2c_device0_ctrl;
15  extern const rm_comms_cfg_t g_comms_i2c_device0_cfg;
16  #ifndef rm_hs300x_callback
17  void rm_hs300x_callback(rm_comms_callback_args_t *p_args);
18  #endif
19  /* HS300X Sensor */
20  extern const rm_hs300x_instance_t g_hs300x_sensor0;
21  extern rm_hs300x_instance_ctrl_t g_hs300x_sensor0_ctrl;
22  extern const rm_hs300x_cfg_t g_hs300x_sensor0_cfg;
23  #ifndef hs300x_callback
24  void hs300x_callback(rm_hs300x_callback_args_t *p_args);
25  #endif
26  void hal_entry(void);
27  void g_hal_init(void);
28  FSP_FOOTER
29  #endif /* HAL_DATA_H_ */
```

Figure 36. Thread (.h) Configuration Code

The declaration code, block 8 of Figure 35, is generated in the source file of the corresponding thread your module is added to in the stack window in e² studio. Figure 37 shows the ***hal_data.c*** file after the HS300x module is added to the stack under ***HAL/Common*** and ***Generate Project Content*** is pressed.

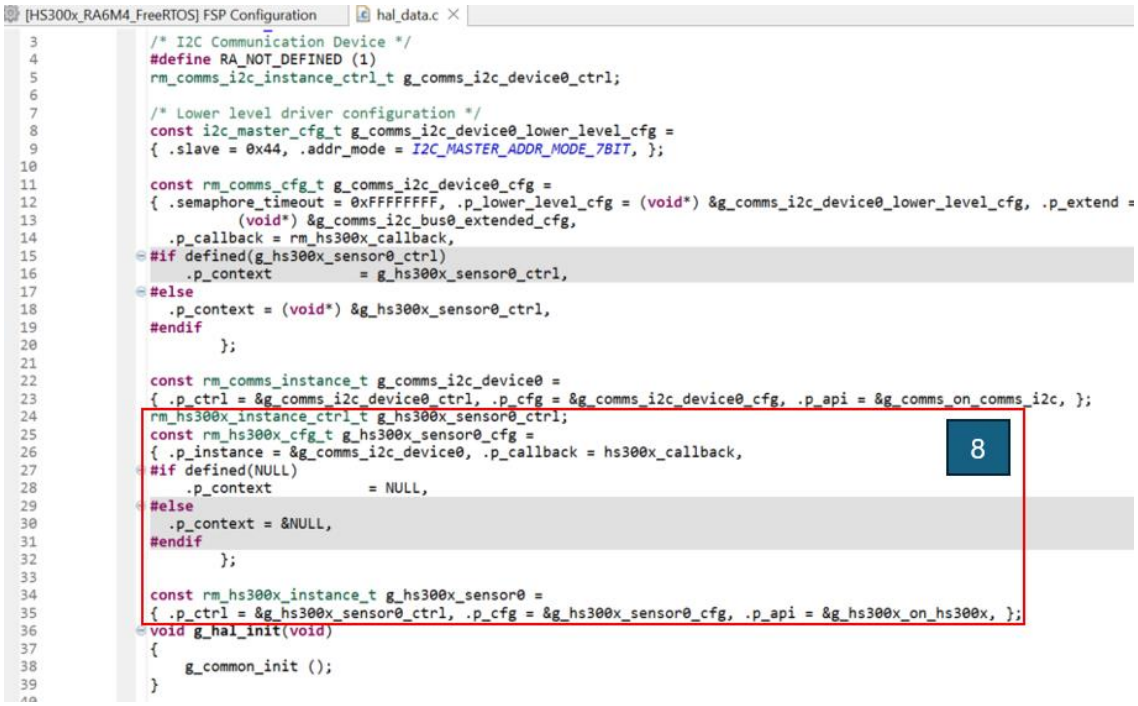

Figure 37. Thread (.c) Configuration Code

A.3 Developer Assistance

This section provides an overview of the developer assistance portion of the module description file. Developer Assistance allows users to drag and drop pre-defined functions to their project in e² studio. The most common use of Developer Assistance is to provide ***Quick Setup*** and ***Quick Get Sensor Data*** functions as seen in the HS300x example shown in Figure 38.

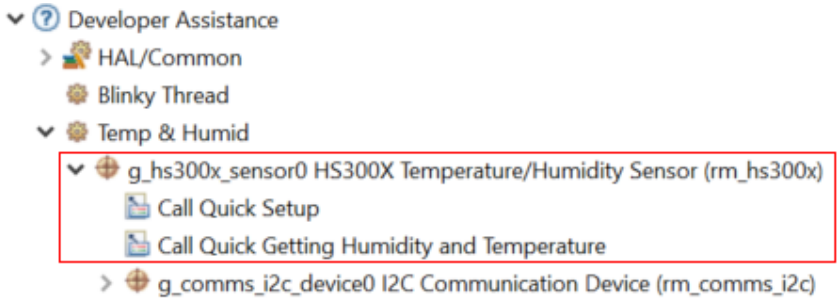

Figure 38. HS300x Developer Assistance in e² Studio

A.3.1 <api> and <template> Elements

The developer assistance section is separated into ***<api>*** and ***<template>*** elements. The ***<api>*** element identifies which module the developer assistance code falls under as well as creating the GUI in e² studio shown above in Figure 38. The ***<template>*** element identifies the code that will be imported to e² studio when an assistance function is dropped into a user's project.

```
<!-- Developer Assistance -->
<developerSupport>
    <api version="1">
        <platform id="fsp" max="" min="1.1.0" />
        <moduleRef id="module.driver.hs300x_on_hs300x" />
        <description>
            <![CDATA[<form><p><span color="header" font="header">Overview</span></p>
            <p>HS300X provides snippets of code for operating the HS300X Middleware.</p></form>]]>
        </description>
        <function display="Quick Setup" id="hs300x_quick_setup">
            <description>
                <![CDATA[<form>
                <p>This function provides quick setup for HS300X using the properties from the RA configurator.</p>
                </form>]]>
            </description>
        </function>
        <function display="Quick Getting Humidity and Temperature"
            id="hs300x_quick_getting_humidity_and_temperature">
            <description>
                <![CDATA[<form>
                <p>This function provides quick getting humidity and temperature values for HS300X using the properties from the RA configurator.</p>
                </form>]]>
            </description>
        </function>
    </api>
```

Figure 39. Developer Assistance API Instantiation

Edit the ***display*** attributes to match the function you wish to implement with your sensor. For example, if implementing the FSP for a Digital Power Monitor, you may want to add a ***Quick Getting Monitor Reading*** function.

```
        <template category="function_call" display="Call Quick Setup"
            id="module.driver.hs300x.quick_setup" version="1">
            <platform id="fsp" max="" min="1.1.0" />
            <moduleRef id="module.driver.hs300x_on_hs300x">
                <function id="hs300x_quick_setup" />
            </moduleRef>
            <content>
                /* TODO: Enable if you want to open HS300X */
                #define
                ${hs300x_name_upper:raProperty(module.driver.hs300x.name_upper)}_NON_BLOCKING (0)

                #if ${hs300x_name_upper}_NON_BLOCKING
                volatile bool g_hs300x_completed = false;
                #endif

                #if RM_HS300X_CFG_PROGRAMMING_MODE
                uint32_t g_hs300x_sensor_id;
                #endif

                /* TODO: Enable if you want to use a callback */
                #define ${hs300x_name_upper}_CALLBACK_ENABLE (0)
                #if ${hs300x_name_upper}_CALLBACK_ENABLE
                void
                ${hs300x_callback:raProperty(module.driver.hs300x.p_callback)}(rm_hs300x_callback_args_t
                * p_args)
                {
                #if ${hs300x_name_upper}_NON_BLOCKING
                if (RM_HS300X_EVENT_SUCCESS == p_args->event)
                {
                g_hs300x_completed = true;
                }
                #else
                FSP_PARAMETER_NOT_USED(p_args);
                #endif
                }
                #endif
```

Figure 40. Developer Assistance Template and Code Elements

The ***<template>*** element contains the following attributes:

- display – name of the function shown in the e^2 studio Developer Assistance GUI.
- id – establishes the connection between the ***<template>*** and ***<function>*** element declared under the ***<api>*** element shown in Figure 39.
- Implement the ***<platform>***, ***<moduleRef>***, and ***<function>*** elements as shown in Figure 40.
- The ***<content>*** element contains the code imported to e^2 studio when the function is selected.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Disclaimer Rev.5.0-1)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.