

How to create a software pack enhanced for STM32CubeMX using STM32 Pack Creator tool

Introduction

STM32CubeMX is part of the STM32Cube initiative designed to simplify and accelerate the development of applications for STM32 microcontrollers.

STM32CubeMX offers the possibility to generate C projects using embedded software offers packages as CMSIS-Pack compliant software packs.

Starting with the 6.0.0 revision, STM32CubeMX is delivered with STM32PackCreator, an STM32 pack creation graphical companion tool, whose main purpose is the creation of software packs.

The generated software packs are:

- Arm® CMSIS-Pack-compliant
- Optionally STM32Cube rules-compliant. Such compliance is mandatory for publishing a pack as an STM32Cube Expansion Package.
- Optionally enhanced for STM32CubeMX. Such enhancements allow the pack to be configured in the STM32CubeMX user interface and for STM32CubeMX to generate custom code in line with the user's configuration.

This document describes what a software pack is, how to create a software pack from scratch using STM32PackCreator and how to verify the generated pack using STM32CubeMX.

It also provides the list of reference material and specifications that are useful when considering the creation of an STM32Cube Expansion Package.



1 References

- [STM32Cube Expansion Packages](#): development guidelines and development checklist for STM32Cube Expansion Packages
- User manual *Development checklist for STM32Cube Expansion Packages* (UM2312)
- User manual *STM32Cube Firmware Packs Specification* (UM2388)
- User manual *STM32Cube BSP drivers development guidelines* (UM2298)
- Wiki *How to develop an STM32Cube Expansion Package* (URL: https://wiki.st.com/stm32mcu/wiki/How_to_develop_a_STM32Cube_Expansion_Package)
- CMSIS-Pack description: Arm® CMSIS-Pack website <https://www.keil.com/pack/doc/CMSIS/Pack/html/index.html>
- Several videos and updates are accessible from STM32CubeMX **[Help>Video]** tutorial menu and from the STM32Cube Expansion wiki page.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.



2 STM32Cube overview

STM32Cube is an STMicroelectronics original initiative to improve designer productivity significantly by reducing development effort, time, and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from conception to realization, among which are:
 - [STM32CubeMX](#), a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
 - [STM32CubeIDE](#), an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
 - [STM32CubeCLT](#), an all-in-one command-line development toolset with code compilation, board programming, and debug features
 - STM32CubeProgrammer ([STM32CubeProg](#)), a programming tool available in graphical and command-line versions
 - STM32CubeMonitor ([STM32CubeMonitor](#), [STM32CubeMonPwr](#), [STM32CubeMonRF](#), [STM32CubeMonUCPD](#)), powerful monitoring tools to fine-tune the behavior and performance of STM32 applications in real time
- [STM32Cube MCU and MPU Packages](#), comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeH7 for the STM32H7 series), which include:
 - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
 - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over hardware
 - A consistent set of middleware components such as RTOS, USB Host and Device, TCP/IP, mbedTLS, FAT file system, audio, and graphics
 - All embedded software utilities with full sets of peripheral and applicative examples
- [STM32Cube Expansion Packages](#), which contain embedded software components that complement the functionalities of the STM32Cube MCU and MPU Packages with:
 - Middleware extensions and applicative layers
 - Examples running on some specific STMicroelectronics development boards

For more details visit [STM32Cube](#).

3 Software pack overview

3.1 Definition and CMSIS-Pack standard

A software pack is a complete file collection shipped in ZIP format (renamed to *.pack).

It complies with Arm® CMSIS-Pack specifications, which define a standardized way to deliver software components (see <https://www.keil.com/pack/doc/CMSIS/Pack/html/index.html>)

It includes:

- Source code, header files, and software libraries
- Documentation and source code templates
- Example projects
- The pack .pdsc file: this file is designed for software development environments. It is an XML-based package description (PDSC) file that describes the content of the software pack and the usage context for the files supplied within the pack (for example, on which conditions, they can be used, if any such condition exists)

For an introduction to CMSIS Packs structure and format of a pack description file (PDSC), check out:

- Keil® website: <https://www.keil.com/pack/doc/CMSIS/Pack/html/index.html>
- Keil® tutorial page: https://www.keil.com/pack/doc/CMSIS/Pack/html/cp_SWComponents.html

Figure 1. Keil® tutorial cover page



The Arm® CMSIS-Pack system solves several problems:

- It provides meta-data of files that relate to a software component. All files that belong to a software component can be identified and information about the original provider is preserved.
- It enables consistent software component upgrades and identifies incompatible configuration files that may be part of the user application.
- Software component providers can specify the interfaces and relationship to other software components.
- The meta-data of a software component can include dependency information for toolchains, devices, and processors, which simplifies integration into application programs.
- Thousands of software packs have been created by Arm® and its partners: these can be easily downloaded, installed, and used in software projects, using Arm® development tools, STM32CubeMX, STM32CubeIDE and any other tool supporting the standard (some support in IAR-EWARM).
- Tutorial page to introduce CMSIS-Pack structure and format of a pack description file (PDSC) for beginners: https://www.keil.com/pack/doc/CMSIS/Pack/html/cp_SWComponents.html

STM32PackCreator allows generating such Arm® CMSIS-Pack compliant packs.

3.2 STM32Cube Expansion Packages

On top of the CMSIS-Pack standard, STMicroelectronics specifies some rules to create STM32Cube Expansion Packages. Refer to the STM32Cube Expansion page on www.st.com and the Wiki link in [Section 1](#)

When the option to create an STM32Cube Expansion Package is enabled for the project, STM32PackCreator implements the relevant constraints, like fields that become mandatory, and specific file paths.

3.3 Software pack creation cycle

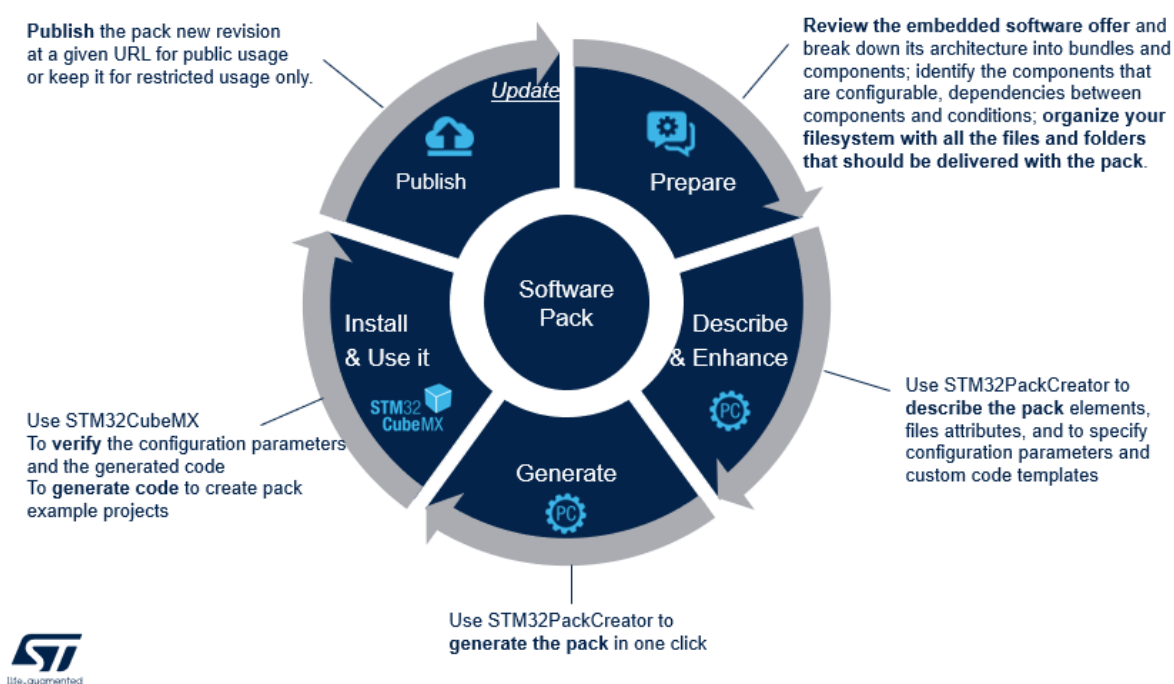
According to the CMSIS-Pack standard, pack owners are responsible for their pack hosting and maintenance. Refer to [Figure 2](#).

The Arm® CMSIS-Pack standard is designed as a web-based distribution network. Each provider of a CMSIS-Pack (also referred to as vendor) is responsible for hosting, maintaining, and publishing unique versions of a CMSIS-Pack.

A CMSIS-Pack is uniquely identified by <vendor>.<pack name>.<version>.pack. All published versions of a pack and the PDSC file need to be available in the same web folder specified by <url>. Multiple different packs may be located in the same web folder.

Refer to <https://www.keil.com/pack/doc/CMSIS/Pack/html/packIndexFile.html>.

Figure 2. Creation cycle for software pack enhanced for STM32CubeMX



4 STM32PackCreator overview

4.1 Principles

Historically, pack developers must perform manual updates of XML files, run several scripts and launch different toolsets to ensure things worked out as expected.

Today pack developers can rely on a single tool, STM32PackCreator, to get guidance, avoid errors and generate the pack in one click.

With STM32PackCreator, the pack developer may:

- Describe and package their software offers as packs,
- Enhance the pack to accelerate the process of application creation by end users,
 - Introduce configuration parameters for users to automatically generate the configuration relevant for their application using STM32CubeMX. From the STM32CubeMX user interface, the user can set the values of configurable software component parameters and retrieve them as C-code statements in the generated C projects, meaning as #define statements.
 - Introduce platform settings when one or more pack components need to interface with peripherals and GPIOs: users are able, from STM32CubeMX User Interface, to select among a choice of possible peripherals or GPIOs to interface the pack components with and retrieve the corresponding initialization C-code.
 - Introduce custom templates when specific code must be generated according to the user's configuration: users retrieve advanced C-code aligned with the configuration they required for their application.
- Check how the pack appears in the STM32CubeMX user interface using the STM32CubeMX preview feature,
- Generate new pack revisions by adding a new release entry in the CMSIS-Pack view and clicking *Save & Generate Pack from the File menu*.

Pack developers use STM32CubeMX to:

- Verify that the project is generated successfully when the pack is enabled,
- Create pack example projects based on STM32CubeMX-generated code.

4.2 Key features

STM32PackCreator allows to:

- Create, save, and open previously saved projects.
- Create new projects from scratch or existing packs.
- Create and update the pack description.
- Create and update the pack configuration parameters and platform settings. The users configure them from STM32CubeMX configuration panel when using the pack in their project.
- Select custom code templates. Such templates are used to generate pack-related code corresponding to the user's configuration.
- Generate Arm® CMSIS-Pack compliant packs.
- Generate STM32Cube Expansion compliant packs.

4.3 Rules and limitations

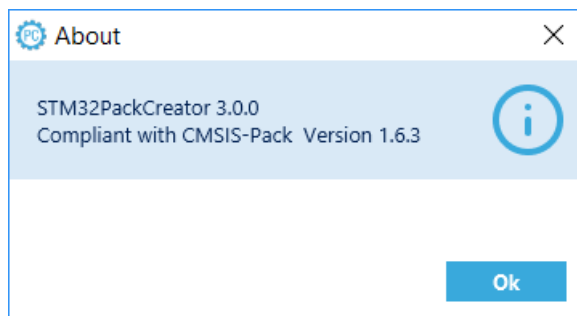
Compatibility with STM32CubeMX

STM32PackCreator implementation is linked to the STM32CubeMX version embedding it. Also, it is strongly advised to use the same STM32CubeMX version or higher to install packs generated with STM32PackCreator in STM32CubeMX.

Compatibility with CMSIS-pack standard

STM32PackCreator generates packs that are compliant with the 1.6.3 standard. Refer to [Figure 3](#).

Figure 3. CMSIS-Pack standard compatibility



Compliance with STM32Cube Expansion rules

STM32PackCreator implements constraints to generate packs that are compliant with STM32Cube Expansion rules. Refer to STM32Cube Expansion Packages in [Section 1](#)

Note: *STM32PackCreator cannot convert an existing pack to become an STM32Cube Expansion Package. The workaround is to create a new pack from scratch using STM32PackCreator and with the STM32Cube compliant option turned on. Refer to [Figure 4](#).*

Figure 4. Activation of STM32Cube compliant option

New Project From Scratch

☒ STM32Cube Expansion package [\(guidelines definitions\)](#)

☐ Free definition package

Project Directory Name *

Alphanum, blank and '._' chars. No leading or trailing space.

Project Parent Folder *

[Browse](#)

Pack Name *

- STMicroelectronics shall use X-CUBE-<Feature>, Partners shall use I-CUBE-<Feature> -

Alphanumeric, - and _ chars

Pack Vendor *

Pick one or enter a new one (Alphanumeric, - and _ chars) ▼

Pack Description *

Please write a significative description of the whole package content.

[Create Project](#)

Log

Please fill all fields to be able to create project.

4.4 System requirements

Supported operating systems and architectures are listed below:

- Windows® 10 and 11: 64 bits (x64)
- Linux®: Ubuntu® LTS 20.04 and LTS 22.04
- macOS® 12 (Monterey), macOS® 13 (Ventura)

STM32PackCreator requires a Java® runtime environment (JRE™) to execute.

When launched from the STM32CubeMX external tool panel on the [STM32CubeMX](#) home page, the STM32 PackCreator tool reuses the JRE™ running STM32CubeMX and no additional installation is required.

Starting with version 6.2.0, STM32CubeMX embeds the Java® runtime environment (JRE™) required for its execution and no longer uses the one installed on the user's machine.

Note: The bundled JRE™ is Temurin™ Adoptium® 17.0.6 and JavaFX-17.0.2.

Note: The STM32CubeMX previous versions require a Java® runtime environment (JRE™) to execute.

The JRE™ version constraints are:

- 64-bit version is mandatory; 32-bit version is not supported.
- JRE™ must support JavaFX
- The minimum JRE™ version is 17.0.2.
- Versions 7, 9, 10, 12, and upper are not supported.

STMicroelectronics promotes the use of the following JRE™s:

- Oracle® JRE™, subject to a license fee
- Amazon Corretto™ JRE™, a no-cost solution based on OpenJDK. The JDK installer is recommended.

STM32PackCreator operation is not guaranteed with other JRE™s.

Note: Windows is a trademark of the Microsoft group of companies.

Linux® is a registered trademark of Linus Torvalds.

Ubuntu® is a registered trademark of Canonical Ltd.

macOS® is a trademark of Apple Inc., registered in the U.S. and other countries and regions.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

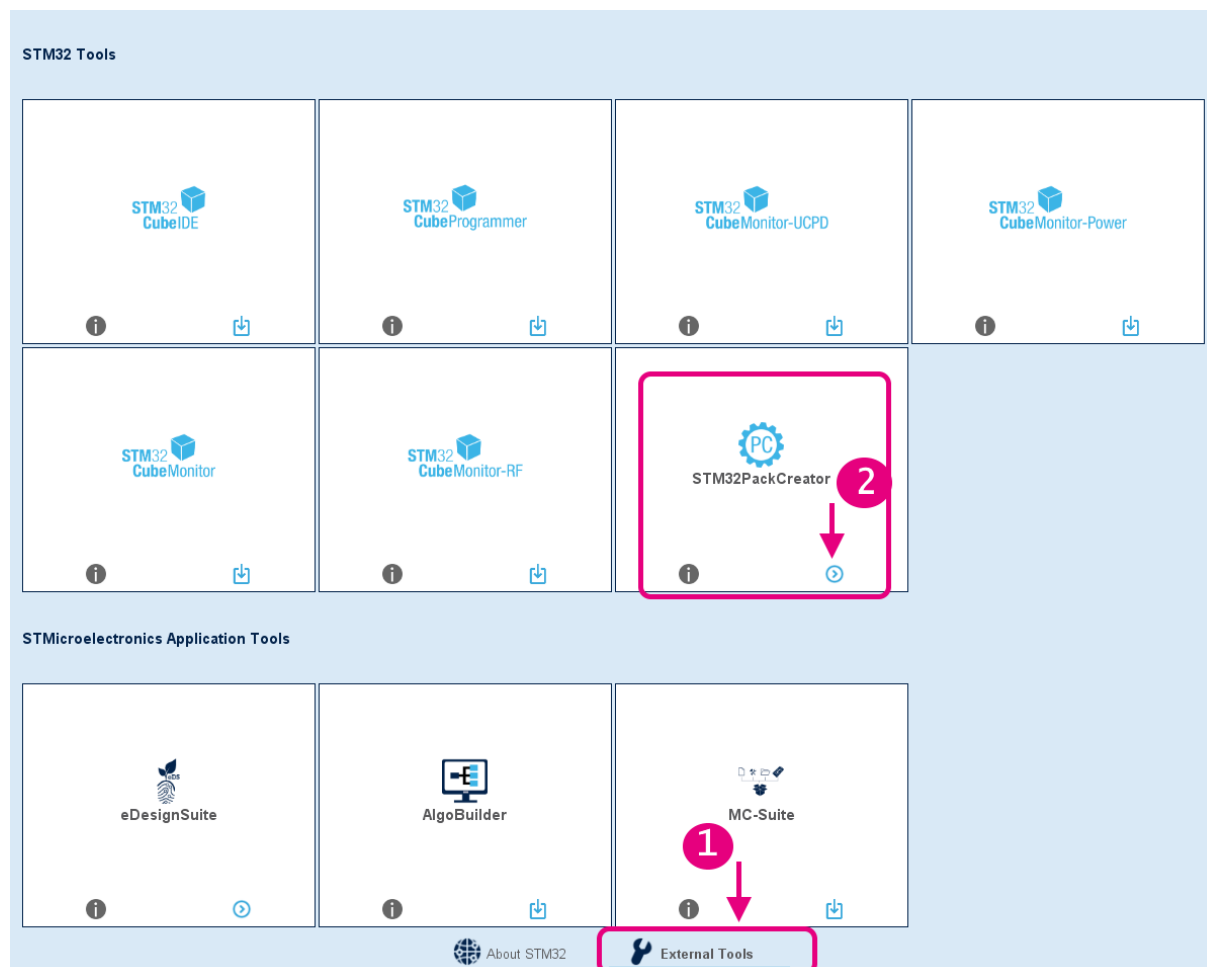
Amazon and Corretto are trademarks of Amazon in the United States and/or other countries.

4.5 Launching STM32PackCreator

4.5.1 From STM32CubeMX user interface

Launch STM32CubeMX. At the right bottom corner of the home page, select External Tools. Click the launch icon to launch STM32PackCreator as a standalone tool.

Figure 5. Access the Tools view



4.5.2 Standalone option

STM32PackCreator executable file can be found under the STM32CubeMX installation path, in the `Utilities` folder.

To launch STM32PackCreator:

- On Windows, double-click the `STM32PackCreator.exe` file.

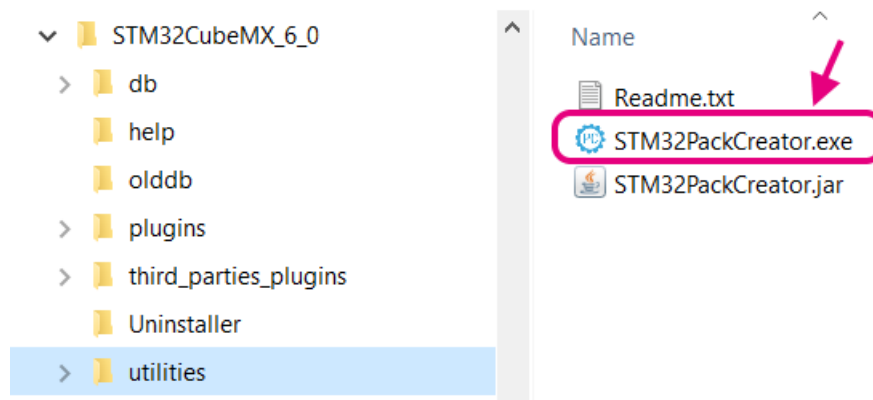
- On Linux and macOS®, use the following command from a terminal window:

```
java -jar STM32PackCreator.jar
```

Starting with STM32CubeMX 6.2, use:

```
<STM32CubeMX install path>/jre/bin/java -jar STM32PackCreator.jar
```

Figure 6. Launch STM32PackCreator



4.6 Main menus

STM32PackCreator comes with two top menus and sub-menus:

Table 1. File menu

| File menu | Purpose |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [Create New Project from scratch] | It creates an empty project ready to be filled with pack details. The pack developer can choose between creating an STM32Cube Expansion Package or a freely defined pack. |
| [New Project from pack] | It creates a new project that has been pre-filled with values inherited from an existing pack. |
| [Open recent] | It gives fast access to the list of most recent projects and opens a project from this list. |
| [Open a project] | Users have to browse the filesystem and select the project folder. |
| [Save Project] | It saves the project during project creation or before the exit to avoid losing changes. |
| [Clone Project As...] | It creates a new project based on the project currently opened and switches immediately to that cloned project. |
| [Close Project] | It closes the project and switches to another project without exiting the tool |
| [Save & Generate Pack] | It saves the project and generates the corresponding pack in one click. |

Table 2. Help menu

| Help menu | Purpose |
|-------------------|---------------------------|
| [About] | Shows version information |
| [Readme] | Opens the release note |
| [Getting started] | Opens this user manual |

4.7 Main views

STM32PackCreator comes with two main views.

4.7.1 CMSIS-Pack view

This view describes the pack contents according to the CMSIS-Pack standard and STM32CubeExpansion rules if the option is enabled, as shown in [Figure 7](#).

Figure 7. Project with STM32CubeExpansion rules enabled

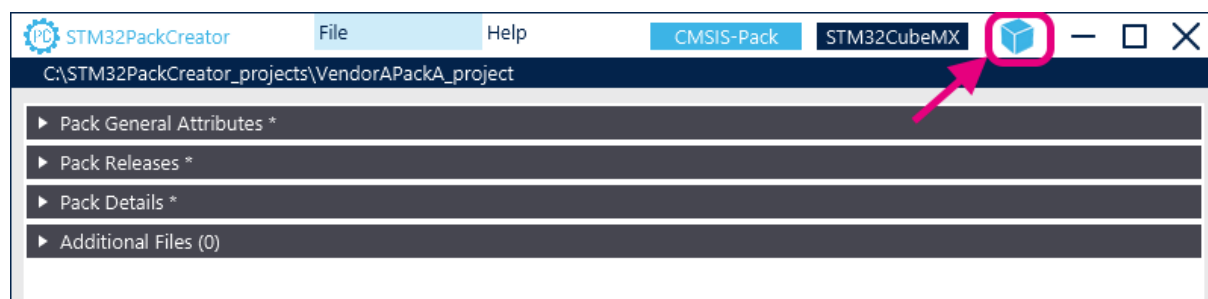
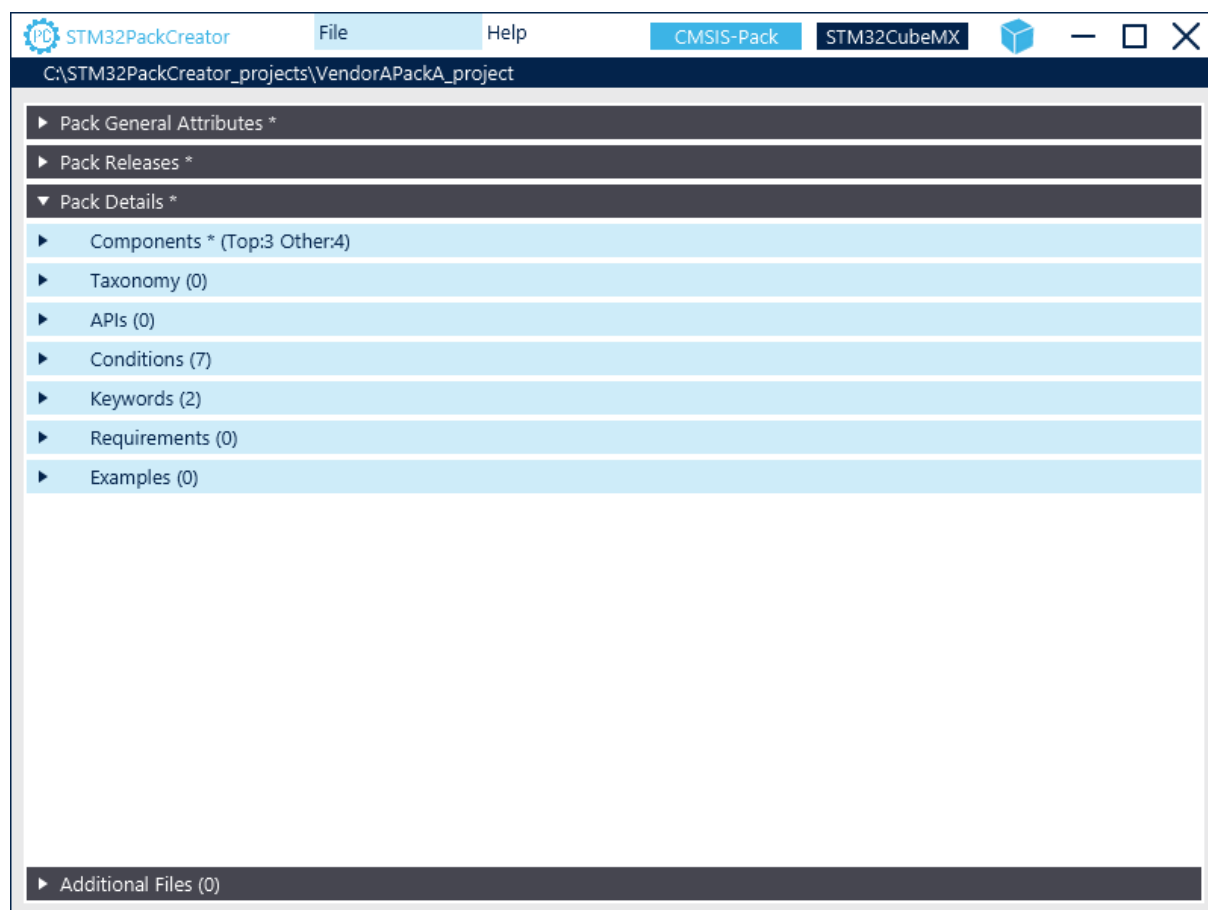


Figure 8. CMSIS-Pack view




It consists of the following entries. The different elements are described in the CMSIS-Pack standard. Tooltips are accessible by clicking the icon  and they provide details on each field.

Table 3. CMSIS-Pack standard elements

| First level | Second level | Usage |
|-------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pack General Attributes | - | Mandatory. Consist of pack names, version, license, and URL. |
| Pack Releases | - | Mandatory. Contains the release history. |
| Pack Details | Requirements | Optional. |
| - | Conditions | Optional/Advanced |
| - | Keywords | Optional |
| - | Taxonomy | Optional |
| - | APIs | Optional/Advanced |
| - | Components | Mandatory Cclass=Device Group=Application: optional, advanced |
| - | Examples | Optional. Consist of example descriptions and paths to their project files. |
| Additional Files | - | Optional. This is used to reference all files that are not listed in the CMSIS-PACK description file (.pdsc file) Remark: this field is not part of the Arm® CMSIS-Pack standard and is not saved in the .pdsc file but the STM32PackCreator project file. |

4.7.2 STM32CubeMX view

This view helps to specify additional items that configure pack components through the STM32CubeMX user interface and generate pack-specific code using the STM32CubeMX code generation feature.

Figure 9. STM32CubeMX view - Example of pack enhanced for STM32CubeMX

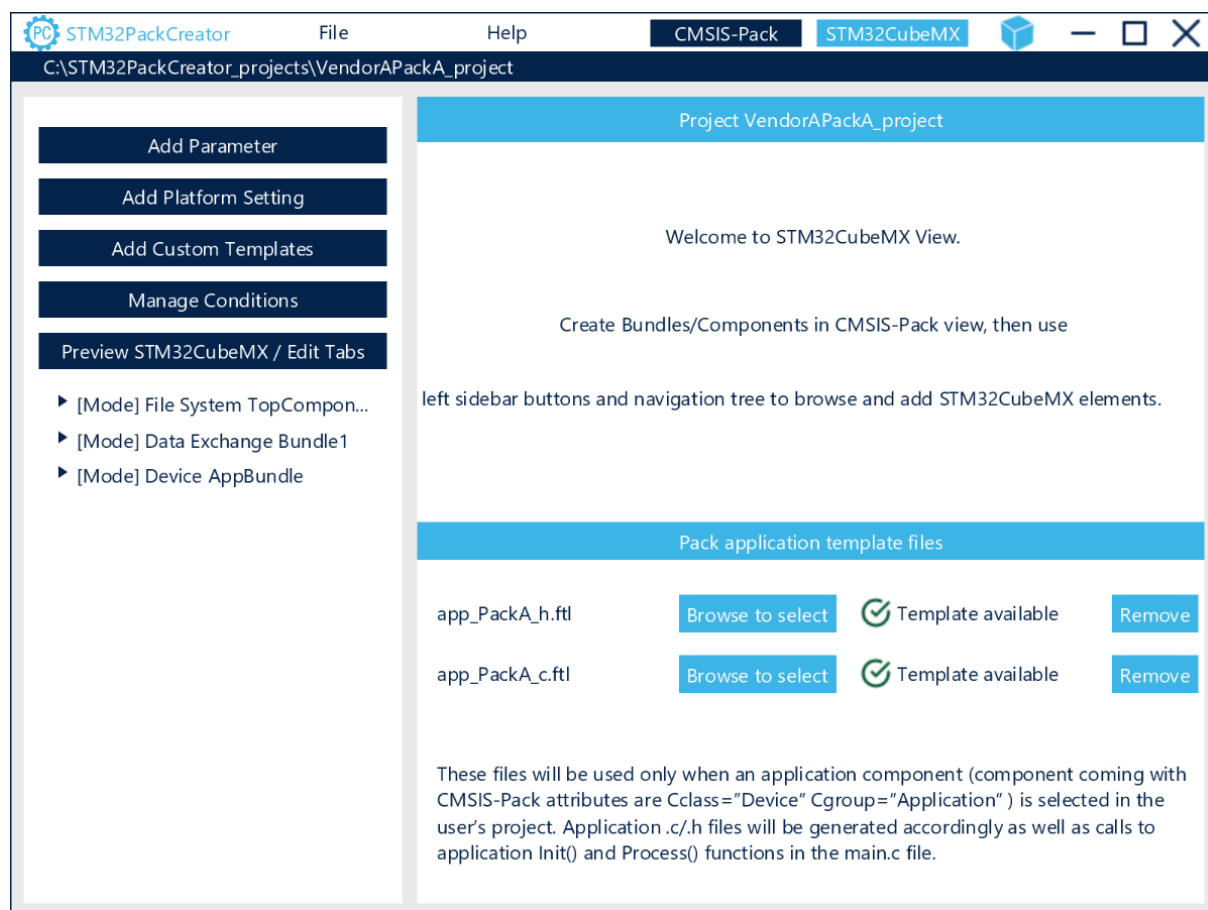


Table 4. STM32CubeMX view - Main features

| First level | Usage |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [Add Parameter] | <p>Mandatory for a pack to be configurable with STM32CubeMX.</p> <p>Creates parameters (name, possible values, location in the user interface, assignment to one or more Pack modes.</p> <p>Note: Pack modes match the Pack top components and bundle entries defined in the CMSIS pack view.</p> <p>It is possible to create parameters with conditions on other parameters or mathematical expressions. For example, enter MOD(32) if the parameter must be a multiple of 32.</p> <p>Note: An expression builder is available to guide the user.</p> <p>By choosing an <i>expression</i> as a parameter type, it is possible to define parameter values that are based on an expression. For, example a parameter can be derived from another parameter, called <i>Param1</i>, by using an expression such as <i>expression = Param1/4</i>.</p> <p>After applying the changes, the parameter is converted to the String type. Changing back to the expression type allows editing the expression previously defined.</p> |
| [Add Platform Setting] | <p>Optional/Advanced.</p> <p>Used to interface the software pack with the MCU configuration.</p> <p>Example: a pack component that requires an SPI in half-duplex mode.</p> <p>Specifies STM32 MCU peripherals or GPIOs functional modes required by the Pack modes.</p> |
| [Add Custom Templates] | <p>Optional/Advanced.</p> <p>Used to generate custom code for the current configuration.</p> <p>They are .ftl files written using Freemarker language. Some are provided by default with STM32</p> |
| [Manage Mode Conditions] | <p>Optional/Advanced.</p> <p>Builds conditions based on pack components.</p> <p>Parameters, platform settings, and custom templates defined under conditions are available only when the condition criteria are met.</p> |
| [Preview STM32CubeMX] | <p>Visualizes how the pack may look in the STM32CubeMX pinout and configuration view. Edits user-defined tabs and groups.</p> |

5 Creating a pack from scratch (step-by-step procedure)

This section shows the creation of a demonstration pack.

Tips and limitations are provided along the way.

5.1 Prepare for pack creation

5.1.1 General approach

In the following, elements highlighted with (*) indicate that they relate to an advanced pack feature and are optional.

The first question to answer is whether or not the pack is an STM32Cube Expansion Package. Going for an STM32Cube Expansion Package eases the next steps as guidelines are available on how the software offer may be packaged and described.

The first step is to assess the pack architecture and requirements:

- How must the software offer be organized: which components, which bundle of components, for which categories (Class, Group, etc...)?
- Are there dependencies between components? Are there conditions on components?
- Is there any API that must be advertised for other packs to use (*)?
- Which components are configurable? What are the configuration parameters?
- Are the parameters available on specific conditions only (*)?
- Are some components board-dependent (*)? Which GPIOs or peripheral modes are required?
- Is there any custom code that must be generated (*)?

Then, it is strongly recommended to keep one folder holding all the files to be delivered with the pack:

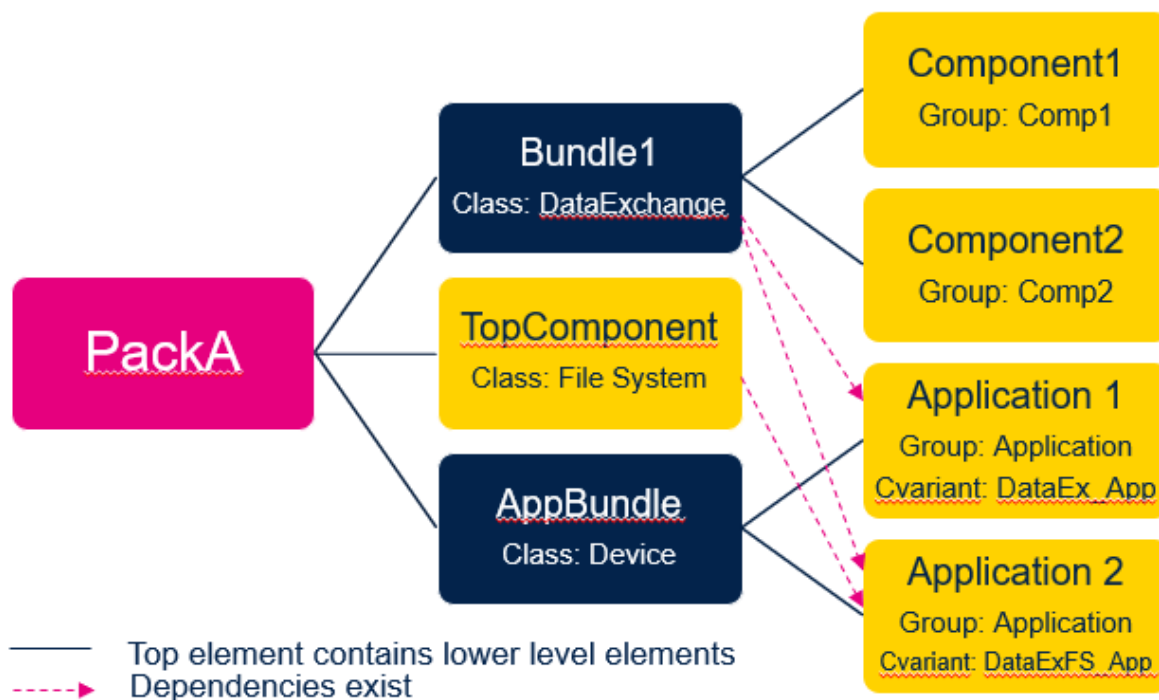
- Source, Header, and Documentation files
 - Which components/bundles they must belong to.
- Freemarker template files (*) used for custom code generation by STM32CubeMX.
- Additional files (*)
 - Files that are present in the pack but not referenced in the `.pdsc` file, such as the STM32 HAL driver files necessary for example projects but not required when generating C-projects with STM32CubeMX.

See [Section 1 References](#) for details.

5.1.2 Demonstration pack overview

Refer to Figure 10 for an overview of the demonstration pack.

Figure 10. Demonstration pack overview

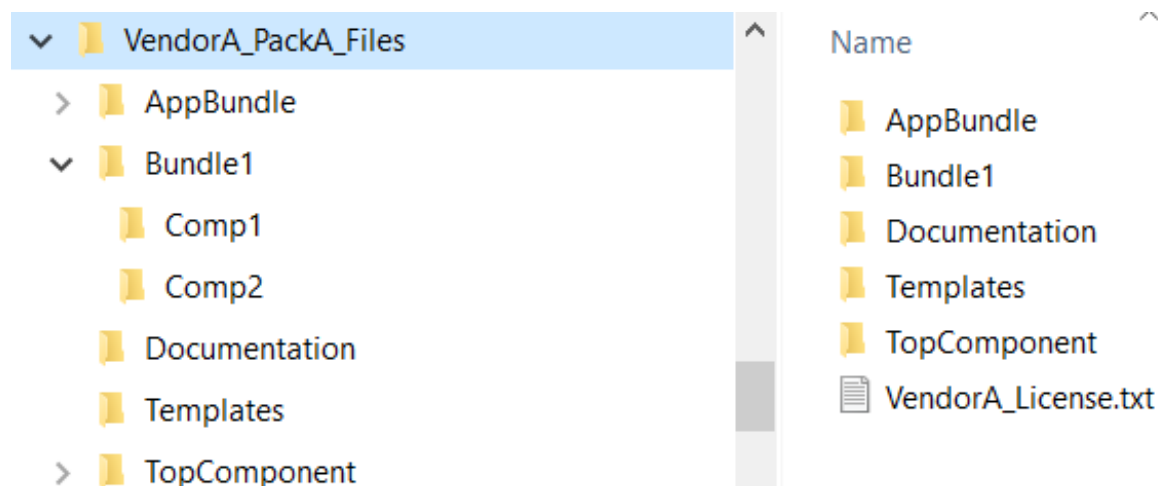


The pack comes with:

- CMSIS-Pack elements
 - One bundle with two components
 - One top component
 - One bundle with two application variants
 - Conditions on components
 - Conditions on component files
- STM32CubeMX enhancements
 - Configuration parameters
 - Platform settings
 - Custom templates
 - Conditions on parameters

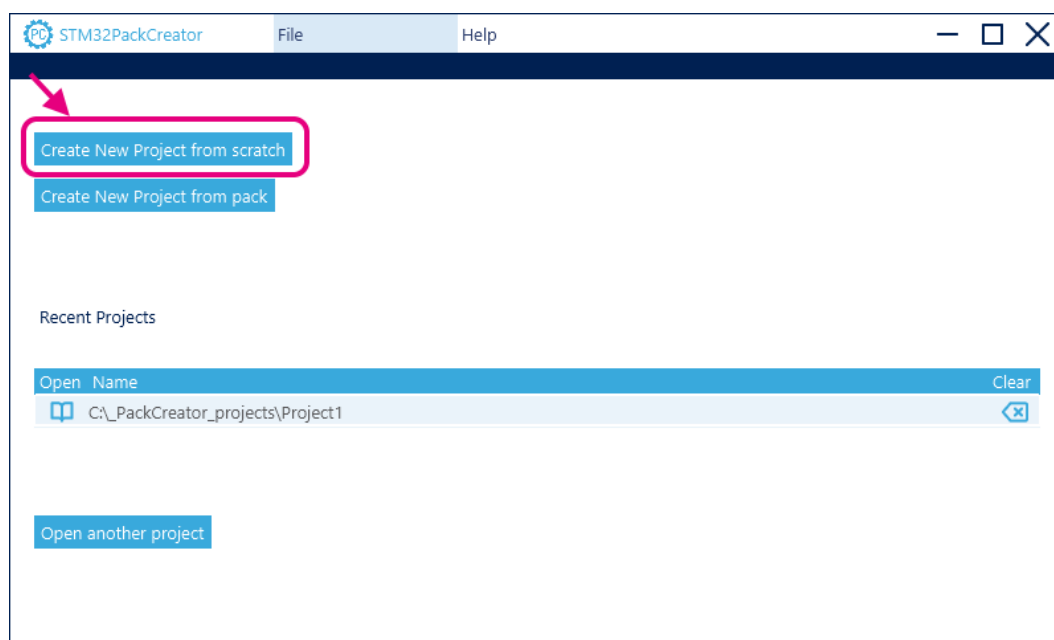
A folder is ready with all the files to be packaged in the pack.

Figure 11. Input folders and files used to produce the demonstration pack



5.2 Create a new project from scratch

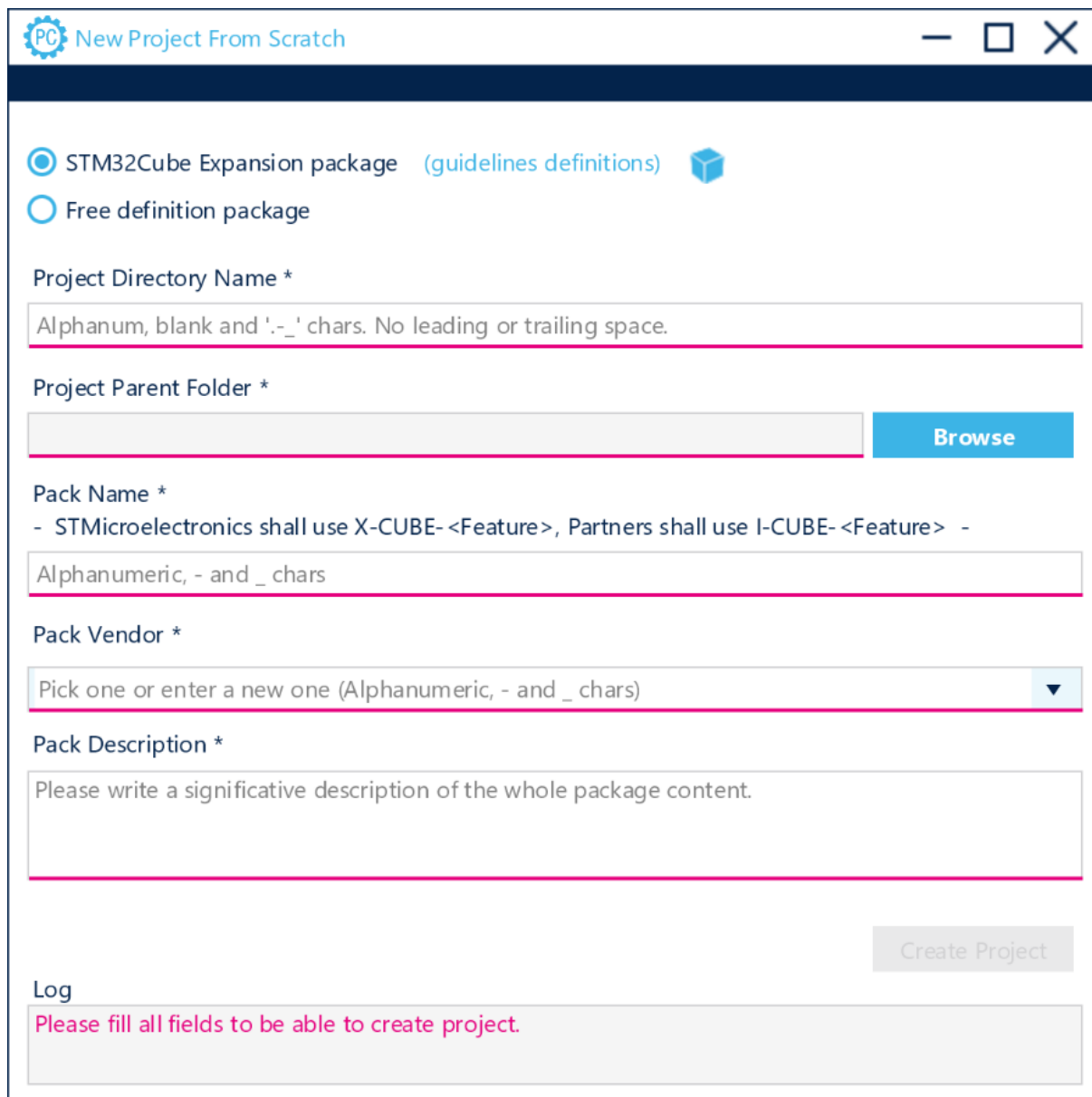
Figure 12. Create a new project from scratch



1. Launch STM32CubeMX.
2. From the STM32CubeMX home page, select the External Tools tab to display the dashboard of external tools. Click the arrow icon to launch STM32PackCreator.

3. Click *Create New Project from scratch* to open the *New Project From Scratch* window.

Figure 13. New project from scratch 1/2



PC New Project From Scratch

☒ STM32Cube Expansion package (guidelines definitions)
 ☐ Free definition package

Project Directory Name *

Alphanumeric, blank and '-' chars. No leading or trailing space.

Project Parent Folder *

Browse

Pack Name *

- STMicroelectronics shall use X-CUBE- <Feature>, Partners shall use I-CUBE- <Feature> -

Alphanumeric, - and _ chars

Pack Vendor *

Pick one or enter a new one (Alphanumeric, - and _ chars)

Pack Description *

Please write a significant description of the whole package content.

Create Project

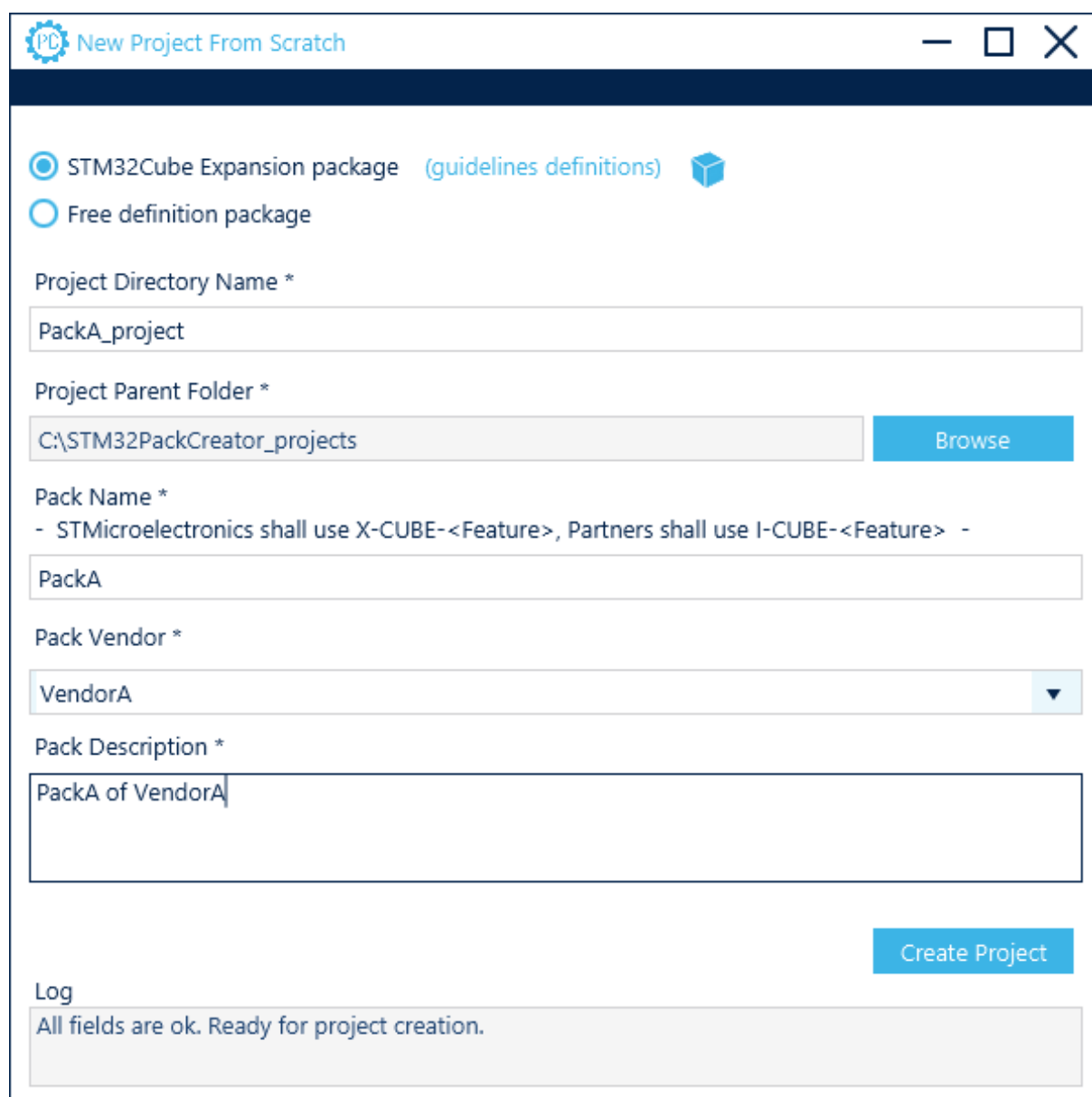
Log

Please fill all fields to be able to create project.


4. Select the option to generate an STM32Cube Expansion Package.

5. Fill in project details:
 - a. Project folder name
 - b. Project parent folder
 - c. Pack name
 - d. Vendor name
 - e. Pack description

Figure 14. New project from scratch 2/2



New Project From Scratch

☒ STM32Cube Expansion package [\(guidelines definitions\)](#) 

☐ Free definition package

Project Directory Name *

PackA_project

Project Parent Folder *

C:\STM32PackCreator_projects Browse

Pack Name *

- STMicroelectronics shall use X-CUBE-<Feature>, Partners shall use I-CUBE-<Feature> -

PackA

Pack Vendor *

VendorA ▼

Pack Description *

PackA of VendorA

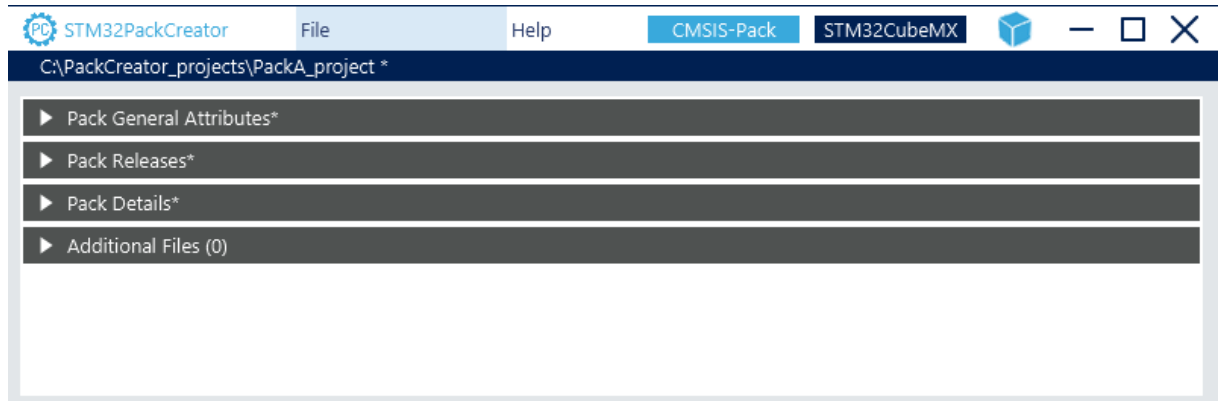
Create Project

Log

All fields are ok. Ready for project creation.

6. Finally, click **[Create Project]**. The project is created and opened on its CMSIS-Pack view.

Figure 15. Project created and opened in CMSIS-Pack view




5.3 Describe the pack using the CMSIS-Pack view

5.3.1 General information and tips

Graphical chart:

- Arrows are used to expand/collapse sections and reflect a hierarchy.
- Edit/Clone Icons are reused across the user interface.
- Fuchsia indicates field usage is mandatory or an error has been introduced

Contextual help:

- Clicking the help icon  shows details about the different elements to configure.
- Light gray helper text is also provided in some text boxes.

Field entries:

Most fields come with pre-defined choices in the dropdown list. Clicking inside the field allows entering custom text.

Field updates:

Most fields can be re-edited by clicking the edit icon.

Field usage:

Mandatory fields are highlighted with * and underlined in fuchsia. Changes can be applied to the project using the **Apply** button, only when all mandatory fields are set.

Errors:

Changes can be applied to the project using the **Apply** button, only when all issues are fixed.

(Warning) Apply versus Save:

Clicking the **Apply** button does not save the project. The user saves the project before exiting the tool or before closing the project: a reminder message shows if applied changes exist and are not saved yet. It is also required to save for generating the pack: it is done through the same **Save & Generate Pack** menu.

Managing components files:

Component files can be specified by clicking **Add Files** to select one or more files found at a given location or using **Add whole folder** to add all the files from a folder.

- STM32PackCreator sets automatically the attributes that are mandatory for the files. It is always possible to change the file attribute values: selecting the file checkbox and then clicking **Edit selected file Attributes** opens the editor window.
- It is possible to delete all files by clicking **Delete All**.
- It is possible to delete some files: select the checkboxes of the files to be deleted and click **Delete selected files**.

5.3.2 Fill in the pack general attributes

Click the left arrow to expand the Pack General Attributes section.

Pack license

STM32CubeExpansion packs must come with a license file

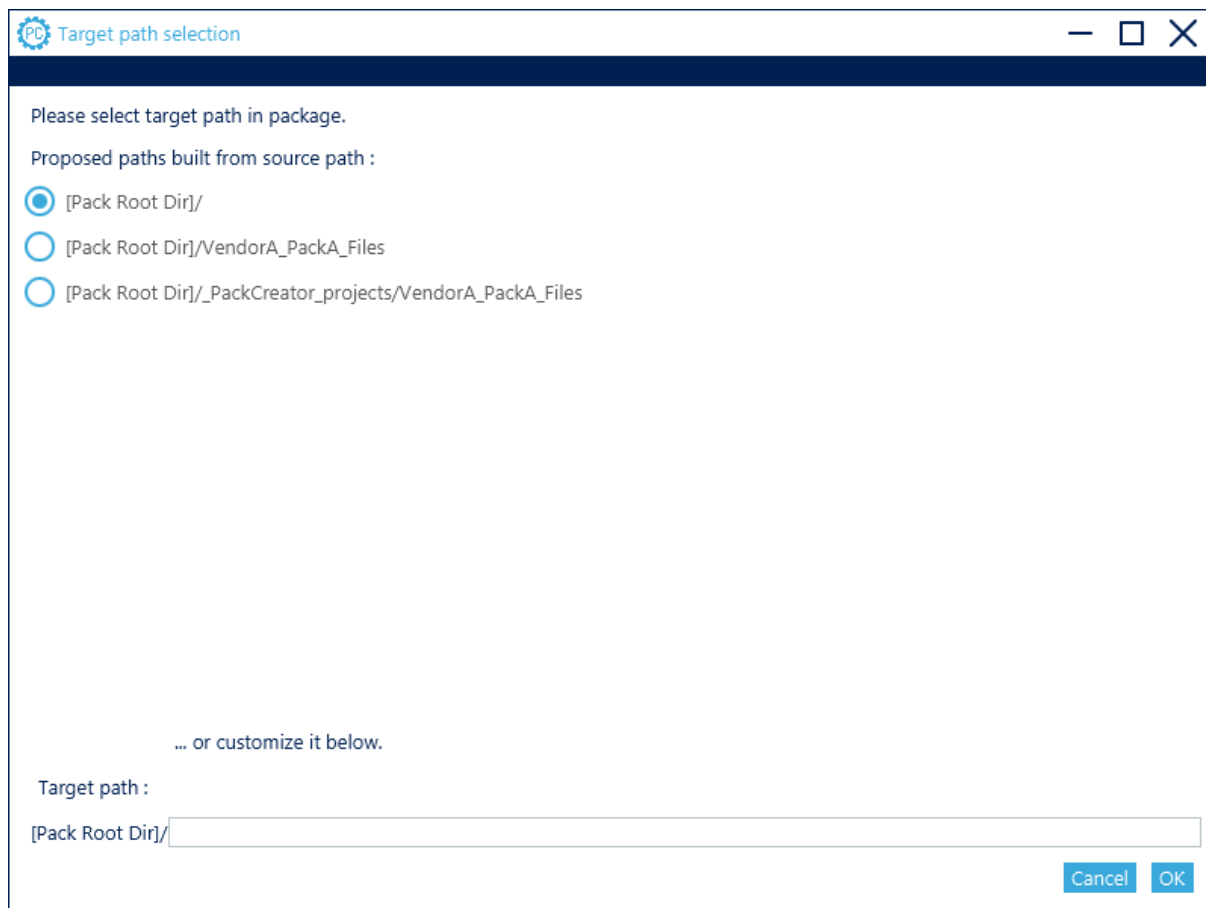
1. Click on browse and select the pack license. Refer to [Figure 16](#).

Figure 16. Selecting the license file

The screenshot shows the STM32PackCreator application window with the CMSIS-Pack view selected. The 'Pack General Attributes' section is expanded. The 'License File' field is set to 'VendorA_License.txt', and a red arrow points to the 'Browse' button next to it. Other fields include Pack Name (PackA), Pack Vendor (VendorA), Device Name, Device Vendor (All), Device Core, TCompiler, Pack Description (PackA of VendorA), Support Contact, and URL (pack location) (http://sw-center.vendorA.com/packs/). At the bottom, there are sections for Pack Releases, Pack Details, and Additional Files.

2. Select the destination folder to be the pack root directory. Refer to Figure 17.

Figure 17. Selecting the pack folder for the license file



The dialog box is titled "Target path selection" and contains the following elements:

- A header bar with a gear icon and the title "Target path selection".
- Text: "Please select target path in package."
- Text: "Proposed paths built from source path :
- Three radio button options:
 - ☒ [Pack Root Dir]/
 - ☐ [Pack Root Dir]/VendorA_PackA_Files
 - ☐ [Pack Root Dir]/_PackCreator_projects/VendorA_PackA_Files
- Text: "... or customize it below."
- Text: "Target path :
- A text input field containing "[Pack Root Dir]/".
- Buttons: "Cancel" and "OK".

3. The License file is now specified. Click apply to apply the changes. Refer to Figure 18.

Figure 18. Project with pack license file attribute specified

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\PackCreator_projects\PackA_project *

▼ Pack General Attributes*

Pack Name* PackA Pack Vendor* VendorA

Pack Description* PackA of VendorA

License File VendorA_License.txt Clear Browse

Support Contact

URL (pack location) http://www.vendorA.com/packs

PACKAGE IS CUBE RULES COMPLIANT ? Cancel Apply

► Pack Releases*

► Pack Details*

► Additional Files (0)

Pack download URL

A URL is required to make the pack public and available for download from the Internet. This can be specified later. This is the URL under which all pack versions and the .pdsc file in its latest revision can be found.

Click **Apply** to apply the changes to the project.

5.3.3 Fill in the release information

Click the left arrow to expand the Pack Releases section.

Initial release

Create the first release by specifying at least (mandatory fields):

- Release version
- Release description
- Release Date

Then Click Apply.

Figure 19. Project with pack release attribute specified

The screenshot shows the STM32PackCreator application window. The 'Pack Releases' section is expanded, showing the 'Current Release' details. The 'Version' field is set to '0.0.1' with a 'New' button next to it. The 'Description' field contains the text 'This is the packA initial version'. The 'Release Date' is set to '2020-06-09'. The 'Release Tag' field is empty. The 'Release URL' field is empty. The 'Deprecated Date' is set to 'yyyy-mm-dd'. The 'Replacement' field is empty. At the bottom of the 'Current Release' section, there are buttons for 'Delete', 'Edit', 'Cancel', and 'Apply'. Below the 'Current Release' section, there is a 'Previous Releases (0)' section, a 'Pack Details*' section, and an 'Additional Files (0)' section.

Other actions

In this section, it is possible to delete the release entry, edit it, or create a new release.

Important information

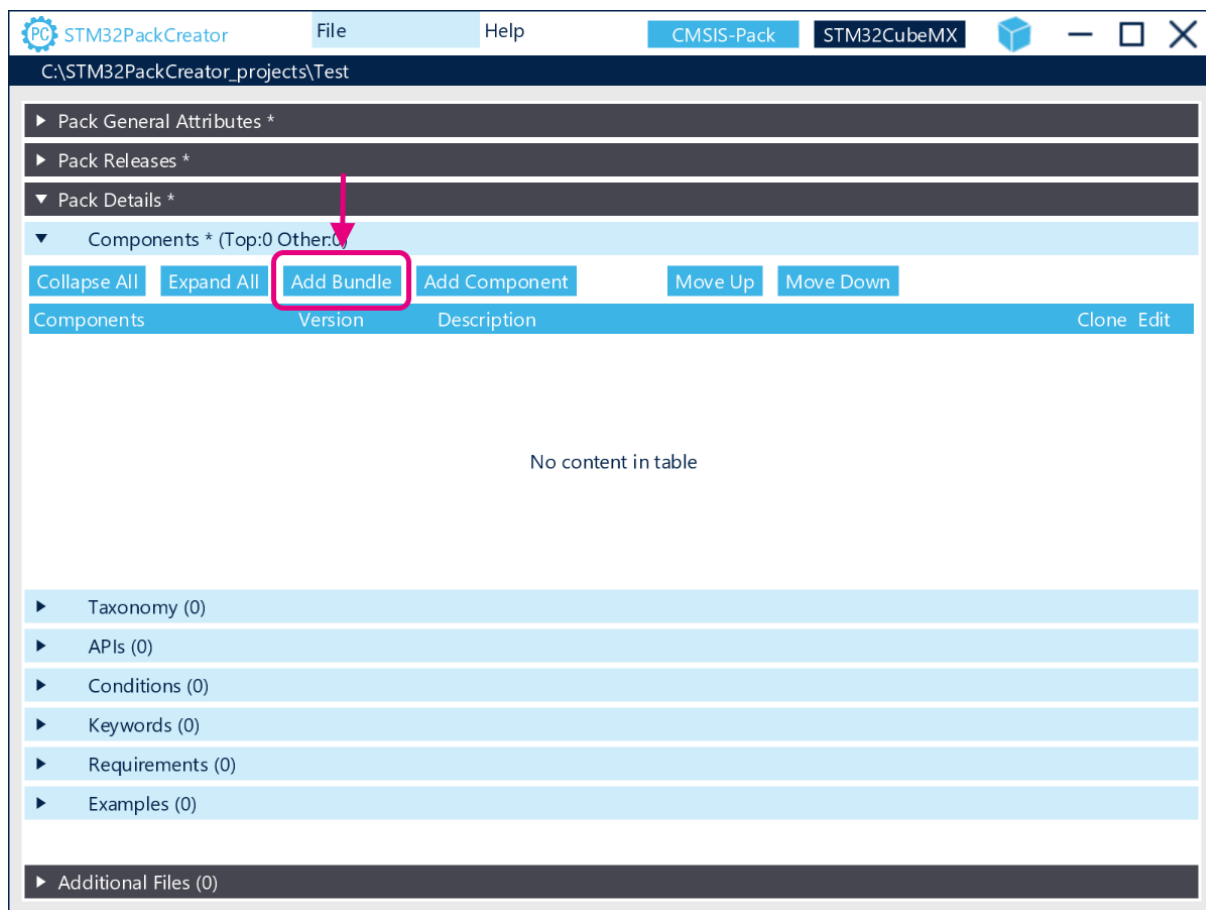
When dealing with official releases, do not delete the release history. According to the CMSIS-Pack standard, each new revision of the pack must come with the full release history. All previous release entries must be kept.

5.3.4 Create a bundle with two components

The first bundle comes with two components inside.

1. Click the left arrow to expand the Pack Details section.
2. Click Add Bundle to open the Component editor window.

Figure 20. Adding a bundle



- Fill in the bundle details, click `Apply`, and close the window.

Figure 21. Filling bundle details

Component editor

Bundle : Data Exchange Bundle1

Bundle's Details Bundle's Components

Vendor* VendorA ▼

Bundle Name* **1** Bundle1

Class* **2** Data Exchange ▼

Version* **3** 1.0.0

Description* **4** DataExchange bundle coming with 2 components

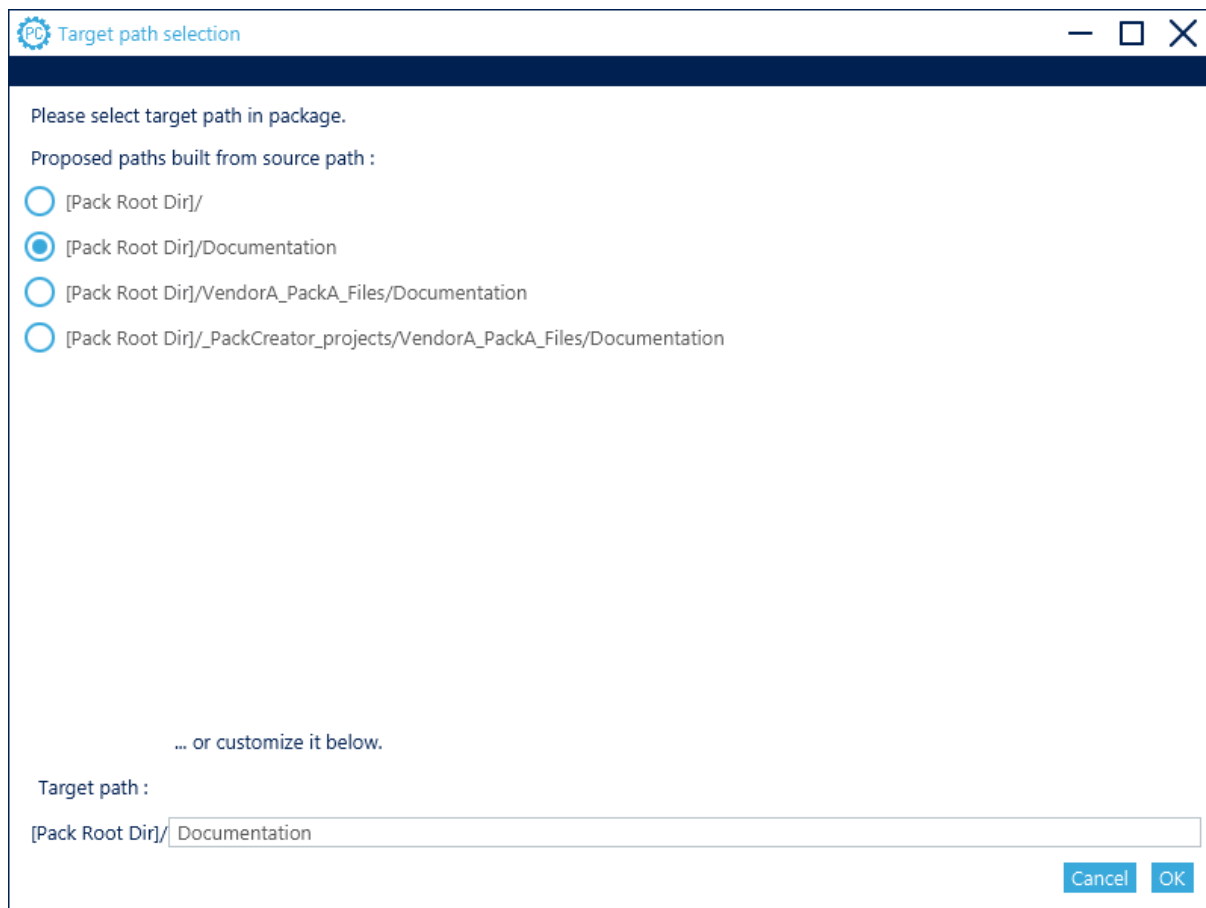
Doc* Documentation/Bundle1.txt **5** Browse

Generator **7**

Delete Bundle ? Cancel Close **6** Apply

For step 5, select the Bundle documentation file and set its target path.

Figure 22. Specifying pack folder for bundle documentation



The dialog box is titled "Target path selection" and contains the following elements:

- A header bar with a gear icon and the title "Target path selection".
- Text: "Please select target path in package."
- Text: "Proposed paths built from source path :
- Four radio button options:
 - ☐ [Pack Root Dir]/
 - ☒ [Pack Root Dir]/Documentation
 - ☐ [Pack Root Dir]/VendorA_PackA_Files/Documentation
 - ☐ [Pack Root Dir]/_PackCreator_projects/VendorA_PackA_Files/Documentation
- Text: "... or customize it below."
- Text: "Target path :
- A text input field containing "[Pack Root Dir]/Documentation".
- Two buttons at the bottom right: "Cancel" and "OK".

4. Create the first component for the bundle.
Back to the Pack Details section, click **Add Component** and fill in the component details, then click **Apply** and close.

Figure 23. Creating component1 for Bundle1

Component editor

Component : ClassToBeDefined GroupToBeDefined

Component's Details Component's Files

Vendor* VendorA

Bundle 1 Data Exchange Bundle1

Class* 2 Data Exchange Version* 1.0.0

Group* 2 Comp1

Subgroup See (?) for help

Description* 3 Component1 of Bundle1

Condition

Need variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui

Generator

Delete Component ? Cancel Close Apply 5 4

5. Create a second component for the bundle.
Back to the Pack Details section, click **Add Component** and fill in the component details, then click **Apply** and close.

Figure 24. Creating component2 for Bundle1

Component editor

Component : ClassToBeDefined GroupToBeDefined

Component's Details Component's Files

Vendor* VendorA

Bundle 1 Data Exchange Bundle1

Class* Data Exchange Version* 1.0.0

Group* 2 Comp2

Subgroup See (?) for help

Description* 3 Component2 of Bundle1

Condition

Need variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui

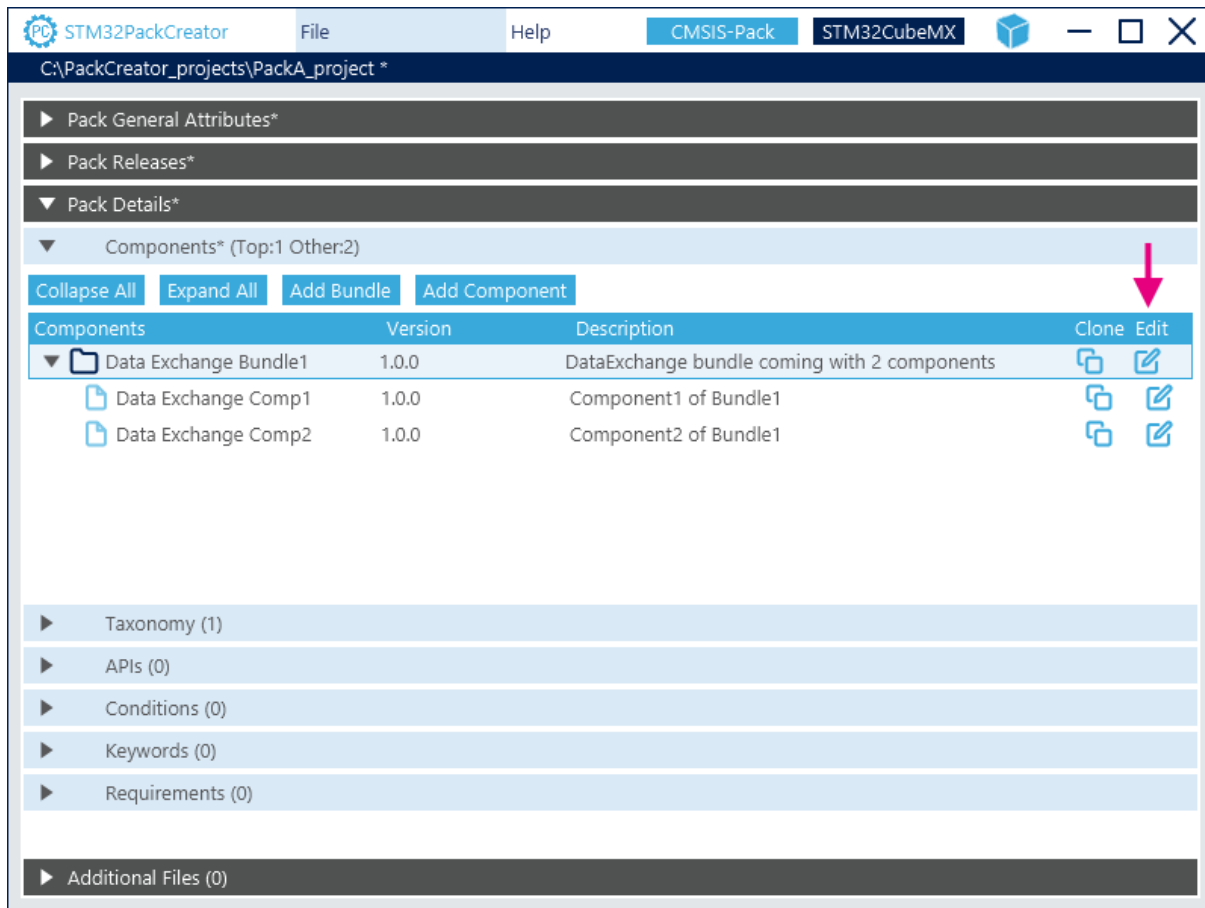
Generator

5 ec 4

Delete Component ? Cancel Close Apply

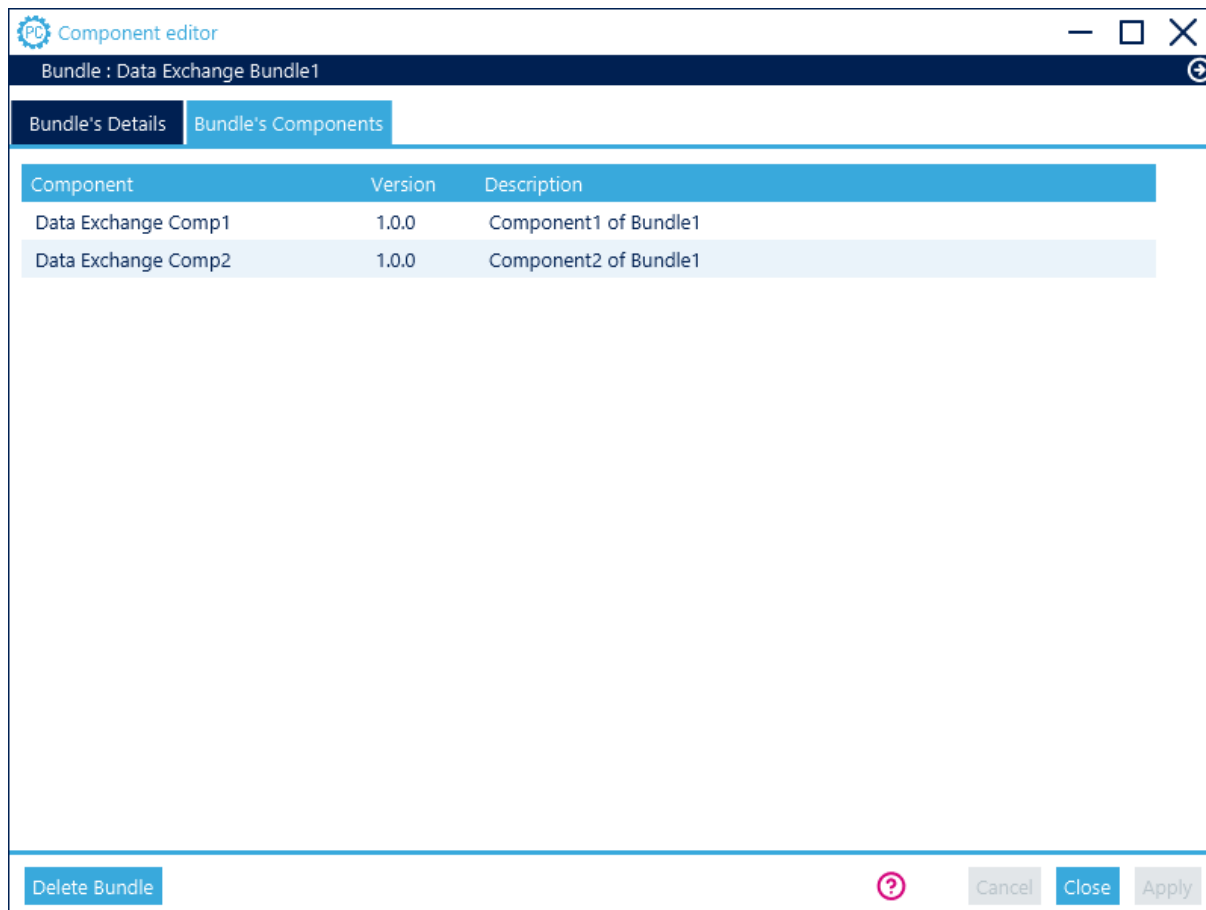
6. Verify that the list of components is shown on the bundle's component tab
Back to the Pack Details section, click the edit icon on the bundle line to open the editor window.

Figure 25. Pack specified with one bundle and two components



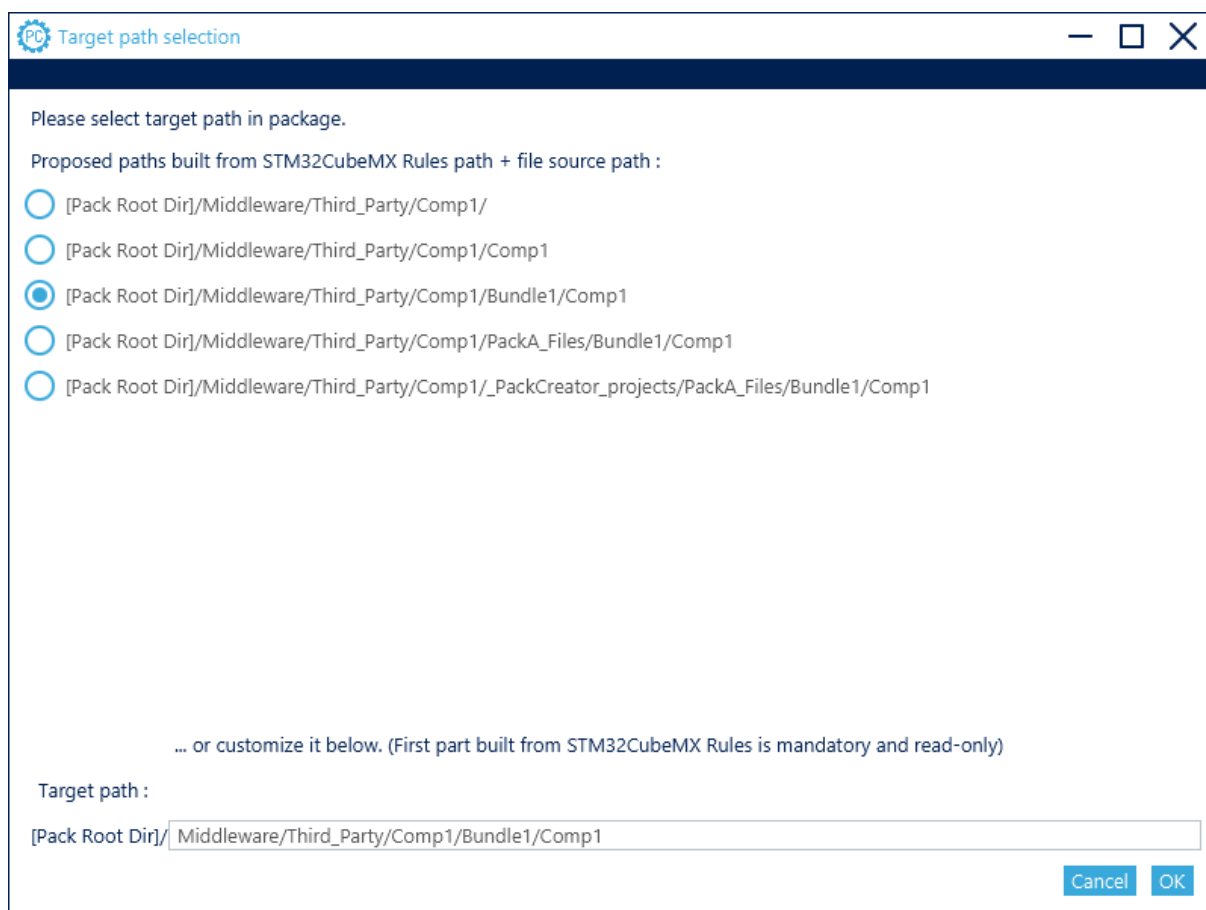
Go to the bundle's components tab and check the two newly created components that appear, then click **Close**.

Figure 26. Bundle's components tab view



7. Add component files to Component 1
Back to the Pack Details section, click the edit icon on the component1 line to open the component editor window.
Go to the Component's file tab and click `Add whole folder` and browse to select the folder holding the component files.
Then, select the path in the pack where the selected files may be found.

Figure 27. Selecting the pack folder for the component1 files



Target path selection

Please select target path in package.

Proposed paths built from STM32CubeMX Rules path + file source path :

- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp1/
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp1/Comp1
- ☒ [Pack Root Dir]/Middleware/Third_Party/Comp1/Bundle1/Comp1
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp1/PackA_Files/Bundle1/Comp1
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp1/_PackCreator_projects/PackA_Files/Bundle1/Comp1

... or customize it below. (First part built from STM32CubeMX Rules is mandatory and read-only)

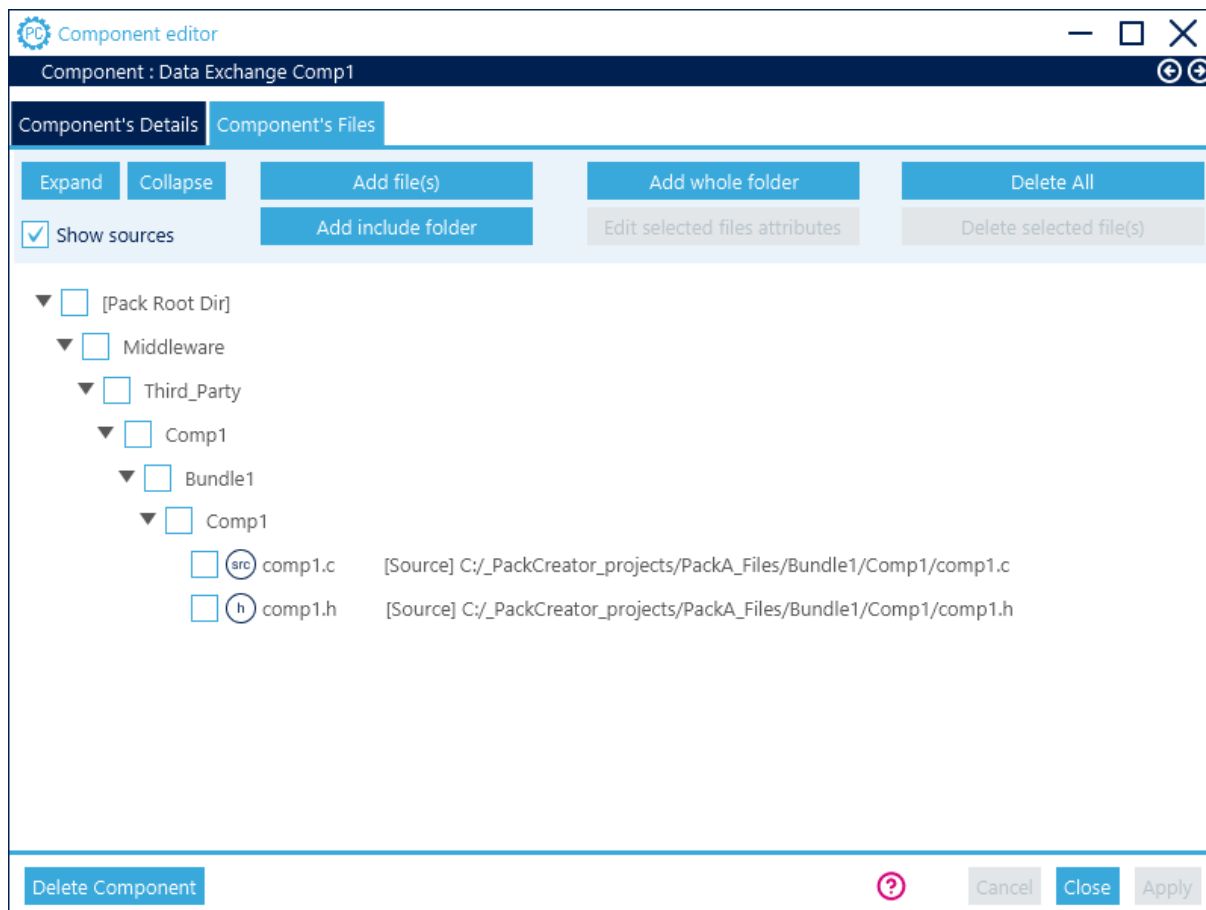
Target path :

[Pack Root Dir]/Middleware/Third_Party/Comp1/Bundle1/Comp1

Cancel OK

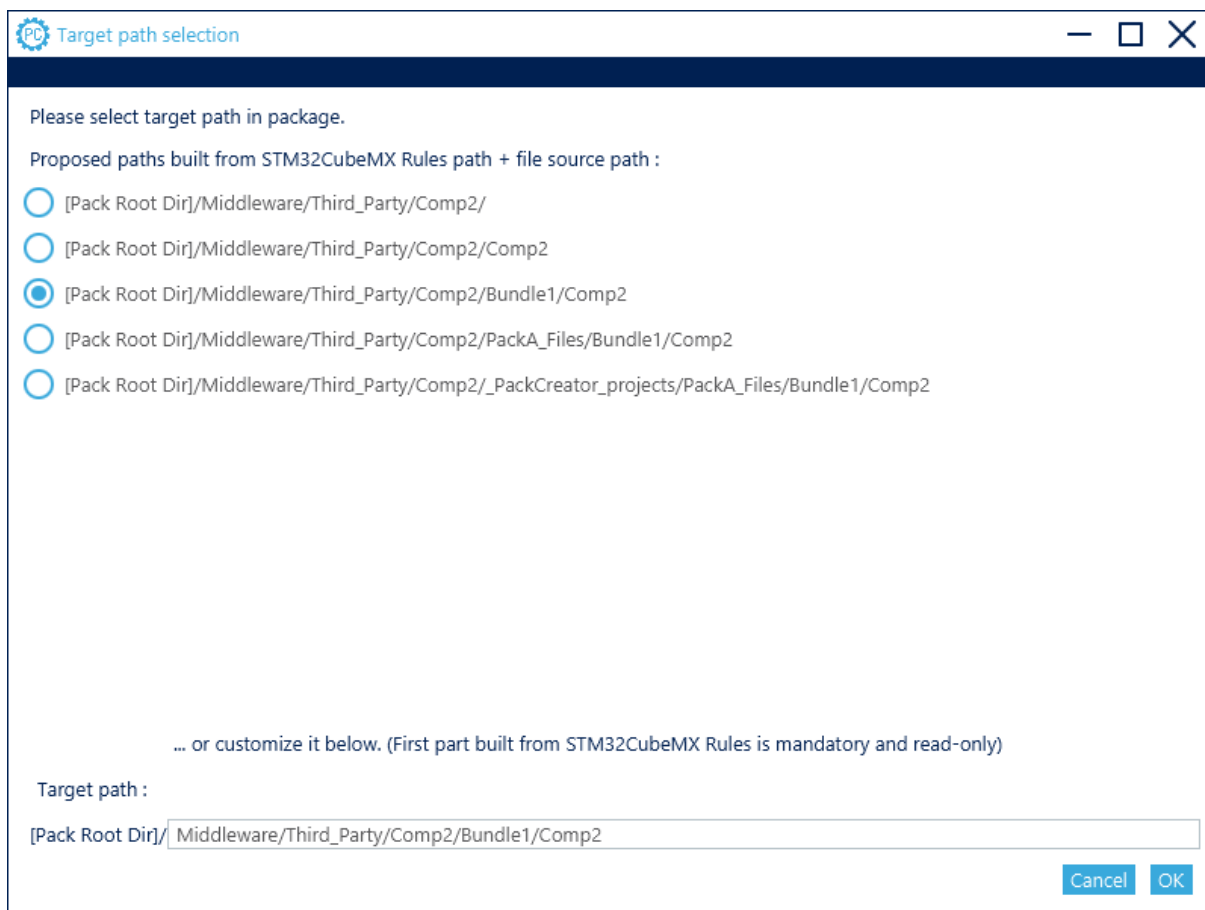
Click Apply and Close.

Figure 28. Component1 with component files specified



8. Add component files to Component 2
Back to the Pack Details section, click the edit icon on the component 2 line to open the editor window. Click `Add whole folder` and browse to select the folder location. Set the path where the selected files may be found in the pack.

Figure 29. Selecting the pack folder for the component2 files



Target path selection

Please select target path in package.

Proposed paths built from STM32CubeMX Rules path + file source path :

- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp2/
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp2/Comp2
- ☒ [Pack Root Dir]/Middleware/Third_Party/Comp2/Bundle1/Comp2
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp2/PackA_Files/Bundle1/Comp2
- ☐ [Pack Root Dir]/Middleware/Third_Party/Comp2/_PackCreator_projects/PackA_Files/Bundle1/Comp2

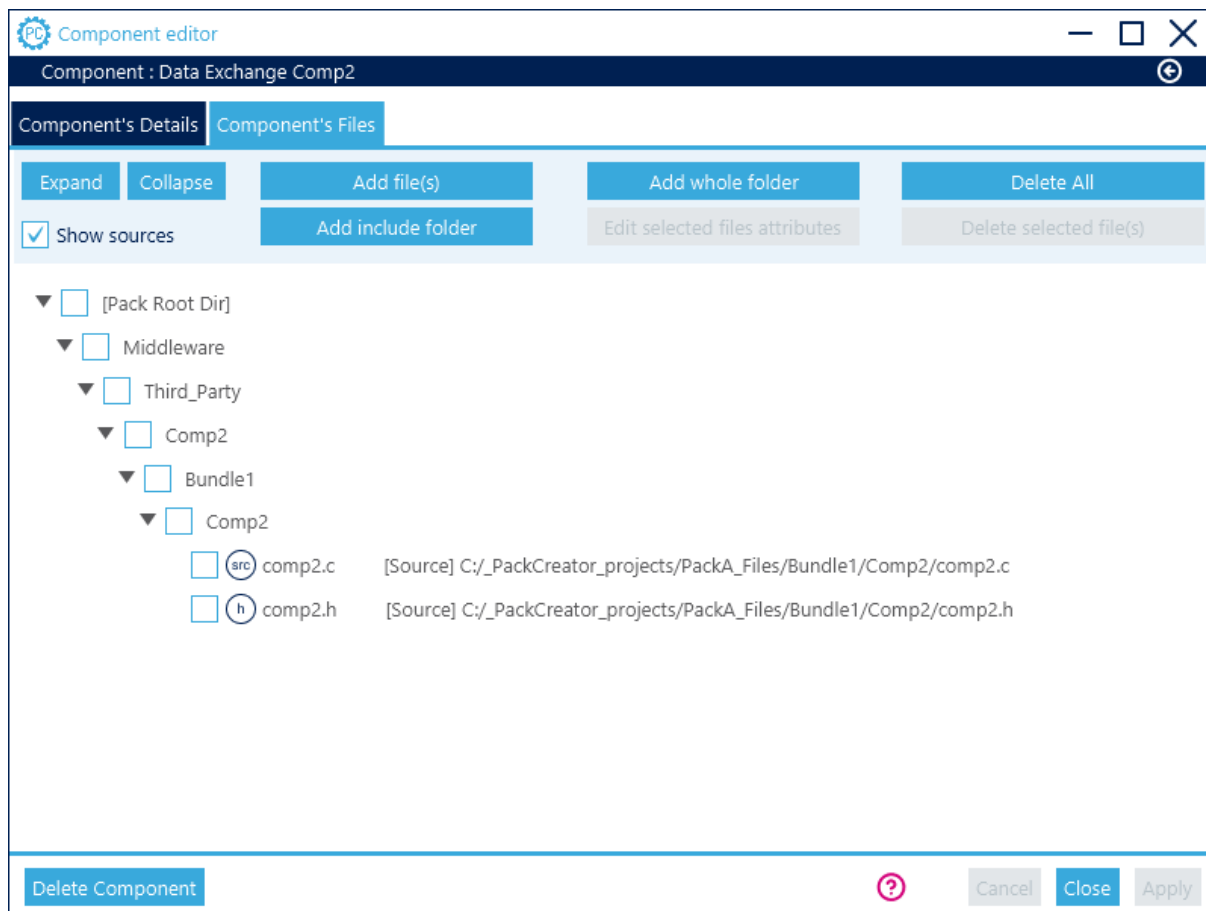
... or customize it below. (First part built from STM32CubeMX Rules is mandatory and read-only)

Target path :

[Pack Root Dir]/

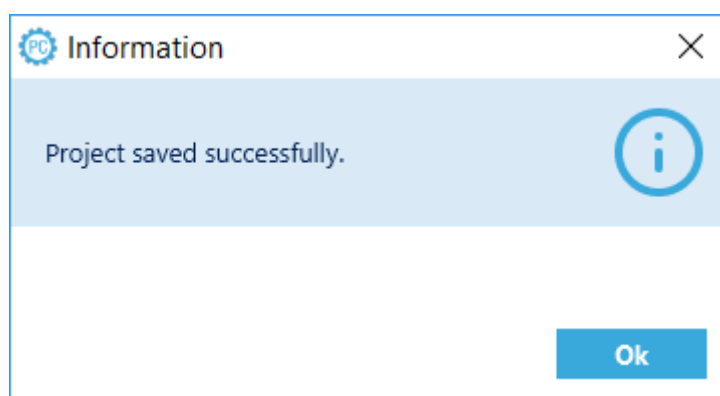
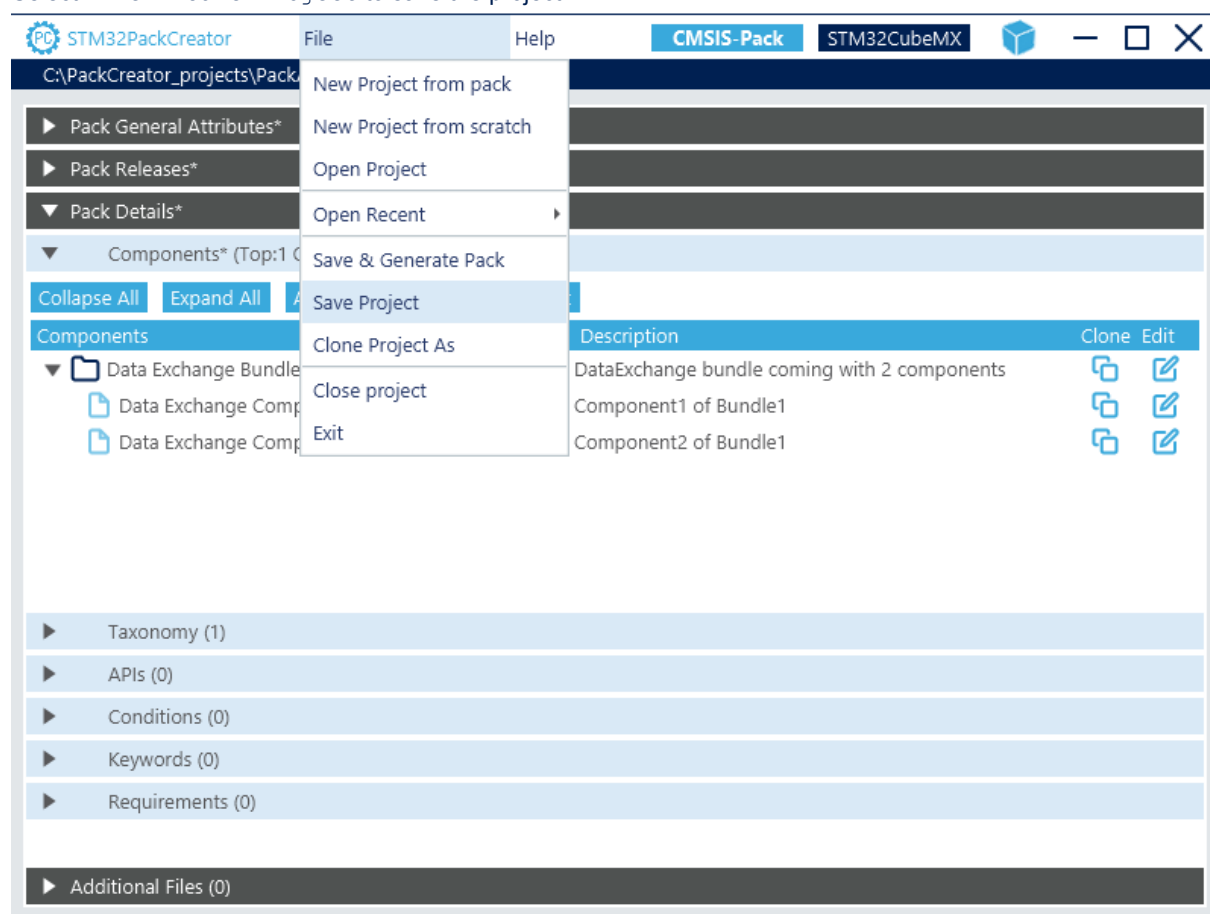
Cancel OK

Figure 30. Component2 with component files specified



9. Save the project

Select **File > Save Project** to save the project.



5.3.5 Create the top component

1. Back to the Pack Details section, click **Add Component** and fill in the component details, then click **Apply**. Since the component is a top component, the bundle is set to **No Bundle**.

Figure 31. Creating a top component

Component editor

Component : File System TopComponent

Component's Details Component's Files

Vendor* VendorA

Bundle - No Bundle -

Class* File System Version* 1.0.0

Group* TopComponent

Subgroup See (?) for help

Description* TopComponent for FileSystem

Condition

Need variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui]

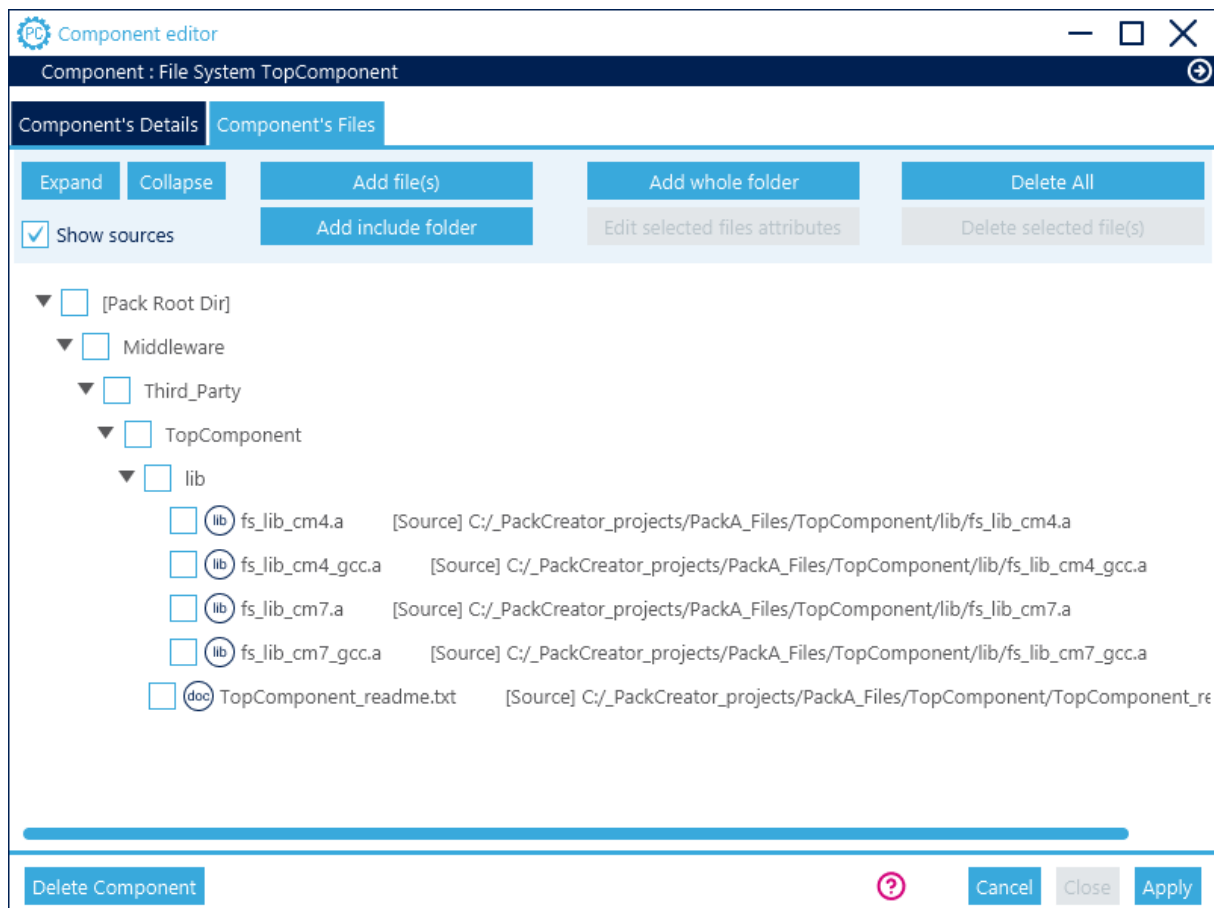
Generator

Depre

Delete Component ? Cancel Close Apply

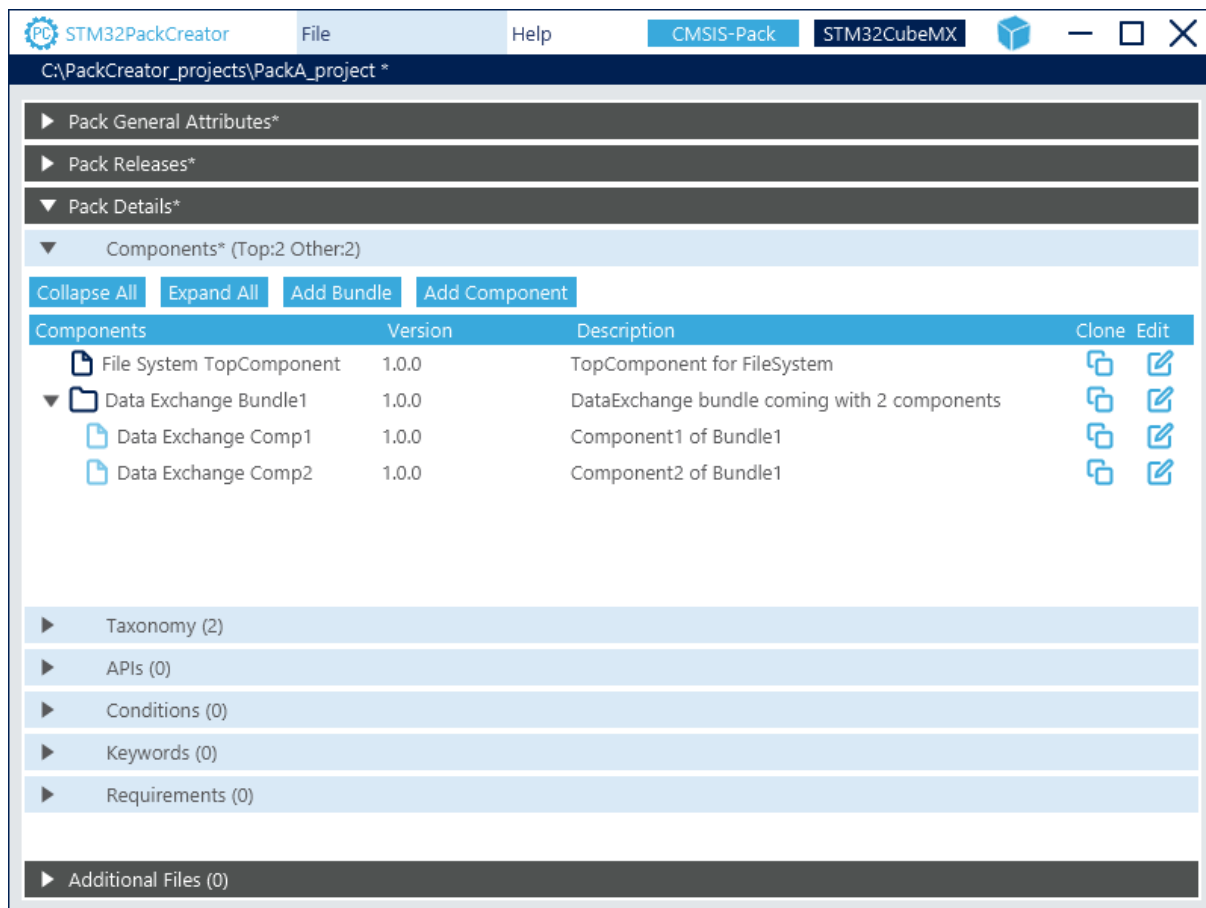
2. Select the Component's files tab then click `Add whole folder:` browse to select the folder location. Set the path in the pack where the selected files may be found.

Figure 32. Adding top component files



3. Click Apply and Close.

Figure 33. Project with one bundle and one top component



4. Save the project by selecting File > Save project.

5.3.6 Create the bundle with application components (specify variant and module names)

Back to the Pack Details section, create a bundle of Class `Device` and two components for that bundle of Group `Application`.

The Application components must come with a Variant field used to specify the application name and a module name that STM32CubeMX uses to generate:

- `MX_<modulename>_Init` and `MX_<modulename>_Process` function calls in `main.c`.
- `App_<modulename>.c` and `App_<modulename>.h` files in module name/app folder.

Note: Some custom templates are also required and are covered when updating the STM32CubeMX view.

Figure 34. Application bundle creation

Component editor

Bundle : ClassToBeDefined BundleToBeDefined

Bundle's Details Bundle's Components

Vendor* 1 VendorA

Bundle Name* 2 AppBundle

Class* 3 Device

Version* 4 1.0.0

Description* 5 Bundle of applications for PackA

Doc* 6 Documentation/AppBundle_readme.txt Browse

Generator

Delete Bundle ? Cancel Close Apply

Figure 35. First application component

Component editor

Component : ClassToBeDefined GroupToBeDefined

Component's Details | Component's Files

Vendor* VendorA

Bundle 1 Device AppBundle

Class* Device Version* 1.0.0

Group* 2 Application

Subgroup See (?) for help ModuleName* 5 dataex

Description* 3 DataExchange application

Condition

Variant 4 DataEx_App ☐ Default Variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui

Generator

Delete Component ? Cancel Close Apply

7 6

Figure 36. Second application component

Component editor

Component : Device Application - DataExFS_App

Component's Details | Component's Files

Vendor* VendorA

Bundle 1 Device AppBundle

Class* Device Version* 1.0.0

Group* 2 Application

Subgroup See (?) for help ModuleName* 5 dataexfs

Description* 3 DataEx w/ Filesystem application

Condition

Variant 4 DataExFS_App ☐ Default Variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui

Generator

Delete Component ? Cancel Close Apply

7 6

Before moving to the next step, save the project by selecting `File > Save project`.

5.3.7

CMSIS-Pack conditions overview and STM32CubeMX restrictions

The CMSIS-Pack standard allows defining conditions and assigning them to components and component files, which then become available only when the condition criteria are met.

Conditions are optional.

Condition types

Conditions come in different types. STM32CubeMX does not manage all possible types. Refer to [Table 5](#).

Table 5. CMSIS-Pack condition types and support in STM32CubeMX

| Attribute type | Attribute name | STM32CubeMX support | Comment |
|-----------------|----------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Device | Dcore | Managed | For example, a component may be available for projects done for a specific core, like Cortex®-M4. |
| - | Dfpu | <i>Not managed</i> | - |
| - | Dmpu | <i>Not managed</i> | - |
| - | Dtz | <i>Not managed</i> | - |
| - | Dsecure | <i>Not managed</i> | - |
| - | Ddsp | <i>Not managed</i> | - |
| - | Dendian | <i>Not managed</i> | - |
| Component | Cvendor | Managed | - |
| - | Cbundle | Managed | - |
| - | Cclass | Managed | - |
| - | Cgroup | Managed | - |
| - | Csub | Managed | - |
| - | Cvariant | Managed | - |
| - | Capiversion | Managed | - |
| T (compiler) | Tcompiler | Managed w/ restriction | CubeMX manages such conditions only if the condition is set at component "file" level. |
| - | Toptions | Managed w/ restriction | Available only for armcc toolchain. Possible values: AC5, AC6 or AC6LTO) CubeMX manages such conditions only if the condition is set at component "file" level. The required Compiler option must be set directly in the IDE (no user setting available in STM32CubeMX UI) |
| Other condition | condition | - | - |

Condition rules

Conditions are made of accept, require, and deny rules.

Table 6. Condition rules

| | |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Accept | At least one accept must be true to signal a true for the complete condition (OR-Rule) |
| Require | All require must be true to signal a true for the complete condition (AND-Rule). |
| Deny | If one denial is true the complete condition becomes false. This element overrides require and accept (AND-NOT-Rule) |

Caution: STM32CubeMX limitation: a rule must not mix conditions of different types ((Device (D), Compiler (T), Component (C)).

Caution: STM32CubeMX limitation: conditions using Dcore may be assigned only on component files at a component or bundle level.

5.3.8 List of pack constraints

PackA components and component files have the following constraints that can be managed using the CMSIS-Pack condition element.

Constraints for Bundle1

Bundle1 components are only accessible for STM32MCUs embedding Arm®Cortex®-M4 or Cortex®-M7 cores: a Dcore condition is necessary.

Constraints for TopComponent files

The files are compiled library files. They are copied according to the compiler and the core selected for the user's project: conditions on Tcompiler and Dcore are necessary.

Constraints for Application components

One Application requires at least one component from Bundle1.

One Application requires at least one component from Bundle1 and the TopComponent component: conditions on components are necessary.

5.3.9

Create condition on Dcore

From the Pack Details section, expand the Conditions sub-section.

Click Add Condition to open the Condition editor.

Enter the condition Name and Description.

Add an accept rule on Cortex-M4.

Figure 37. Creating a condition to accept Cortex-M4-based devices

The screenshot shows the 'Condition editor' window with the following elements and annotations:

- Name:** 'Cortex-M Device' (Annotated with a red circle 1)
- Description:** 'Supported limited to STM32 products using Cortex-M4 and Cortex-M7 cores' (Annotated with a red circle 2)
- Rule Builder:**
 - Buttons: 'Line' (Annotated with a red circle 3) and 'Reset'.
 - Row 1: Attribute 'Dcore' (Annotated with a red circle 3) and Value 'Cortex-M4' (Annotated with a red circle 4).
- Buttons:** 'Add an accept rule' (Annotated with a red circle 5 and a downward arrow), 'Add a deny rule', and 'Add a require rule'.
- Rules Table:**

| Type | Expression |
|------|------------|
| | |
- Footer:** 'Delete Condition' button, a help icon (?), 'Cancel' button, and 'OK' button.

Add an accept rule on Cortex®-M7.

Figure 38. Updating a condition to accept Cortex-M7-based devices

Condition editor

Name
Cortex-M Device

Description
Supported limited to STM32 products using Cortex-M4 and Cortex-M7 cores

Rule Builder
Add Line
Reset

1
Attribute
Dcore
Value
Cortex-M7

Add an accept rule
Add a deny rule
Add a require rule

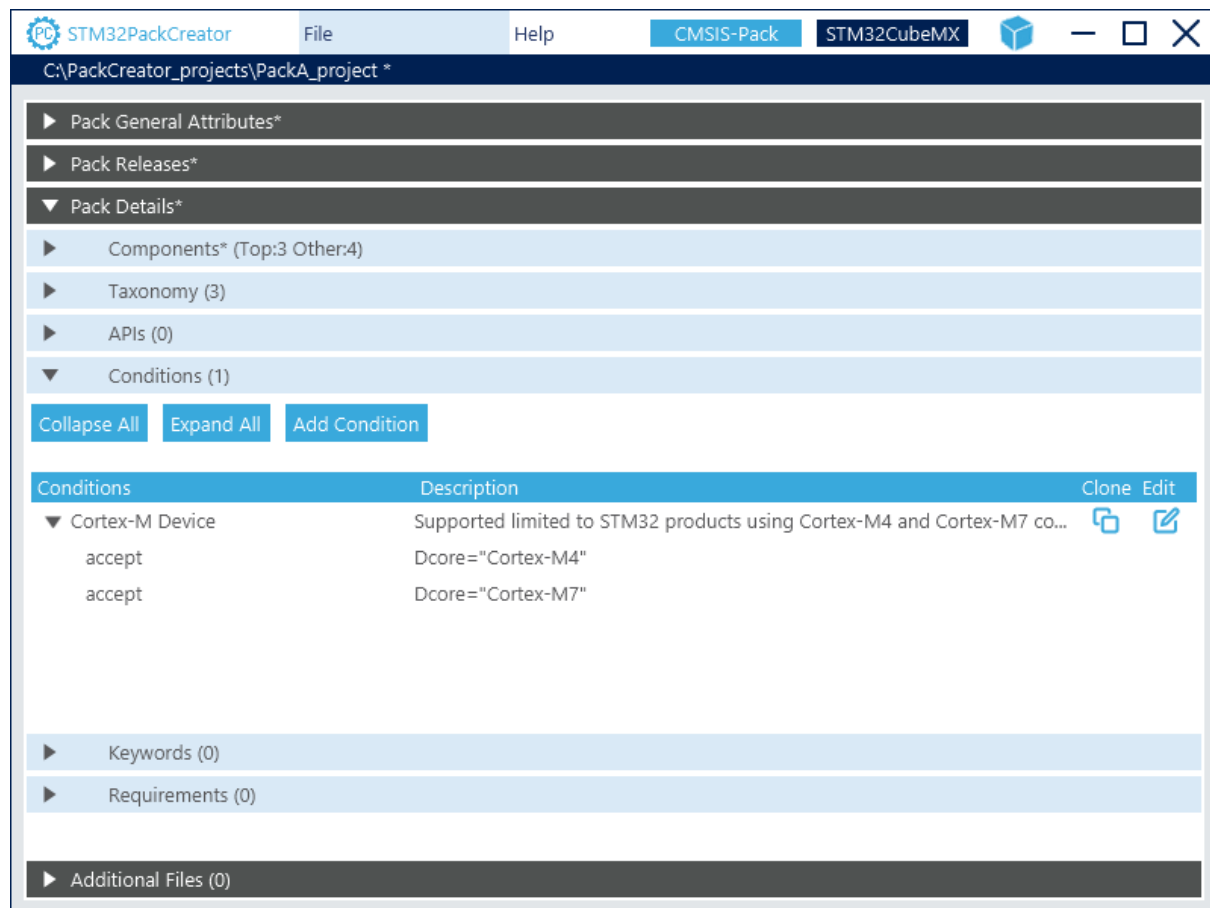
Rules

| Type | Expression | |
|--------|-------------------|--|
| accept | Dcore="Cortex-M4" | |
| accept | Dcore="Cortex-M7" | |

Delete Condition
?
Cancel
OK

Click OK to close. An entry for the newly created condition is displayed under Conditions.

Figure 39. Cortex®-M device condition



5.3.10

Assign Cortex-M device condition on Bundle1 components

Go back to the Components sub-section, edit Bundle1 components details, and select to assign the Cortex-M Device condition. Click Apply and Close.

Figure 40. Assigning condition to component 1

Component editor

Component : Data Exchange Comp1

Component's Details | Component's Files

Vendor* VendorA

Bundle Data Exchange Bundle1

Class* Data Exchange Version* 1.0.0

Group* Comp1

Subgroup See (?) for help

Description* Component1 of Bundle1

Condition

Need variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Build]

Generator

Deleted

Delete Component ? Cancel Close Apply

Figure 41. Condition assigned to component 2

Component editor

Component : Data Exchange Comp2

Component's Details

Component's Files

Vendor*

VendorA

Bundle

Data Exchange Bundle1

Class*

Data Exchange

Version*

1.0.0

Group*

Comp2

Subgroup

See (?) for help

Description*

Component2 of Bundle1

Condition

Cortex-M Device

Need variant

RTE_Components

Max Instances

1

Api Version

MAJOR.MINOR.PATCH[-Pre Rel.][+Buil

Generator

☐ Deprecate

Delete Component

?

Cancel

Close

Apply

5.3.11 Create conditions on Tcompiler

PackA comes with library files that require Keil® compiler and Cortex®-M4 or Cortex®-M7.
From the Conditions sub-section, Click Add Condition to open the Condition editor.

Figure 42. CM4_Keil condition

The screenshot shows the 'Condition editor' window with the following configuration:

- Name:** CM4_Keil
- Description:** Support for Cortex-M4 based devices and Keil compiler
- Rule Build:**
 - 1. Attribute: Tcompiler
 - Value: ARMCC
- Buttons:** Add an accept rule, Add a deny rule, Add a require rule
- Rules Table:**

| Type | Expression |
|---------|-------------------|
| require | Dcore="Cortex-M4" |
- Footer:** Delete Condition, Cancel, OK

Click OK to close and go back to the condition view.

Click the clone icon to quickly create a second condition based on CM4_Keil.

Figure 43. Cloning conditions

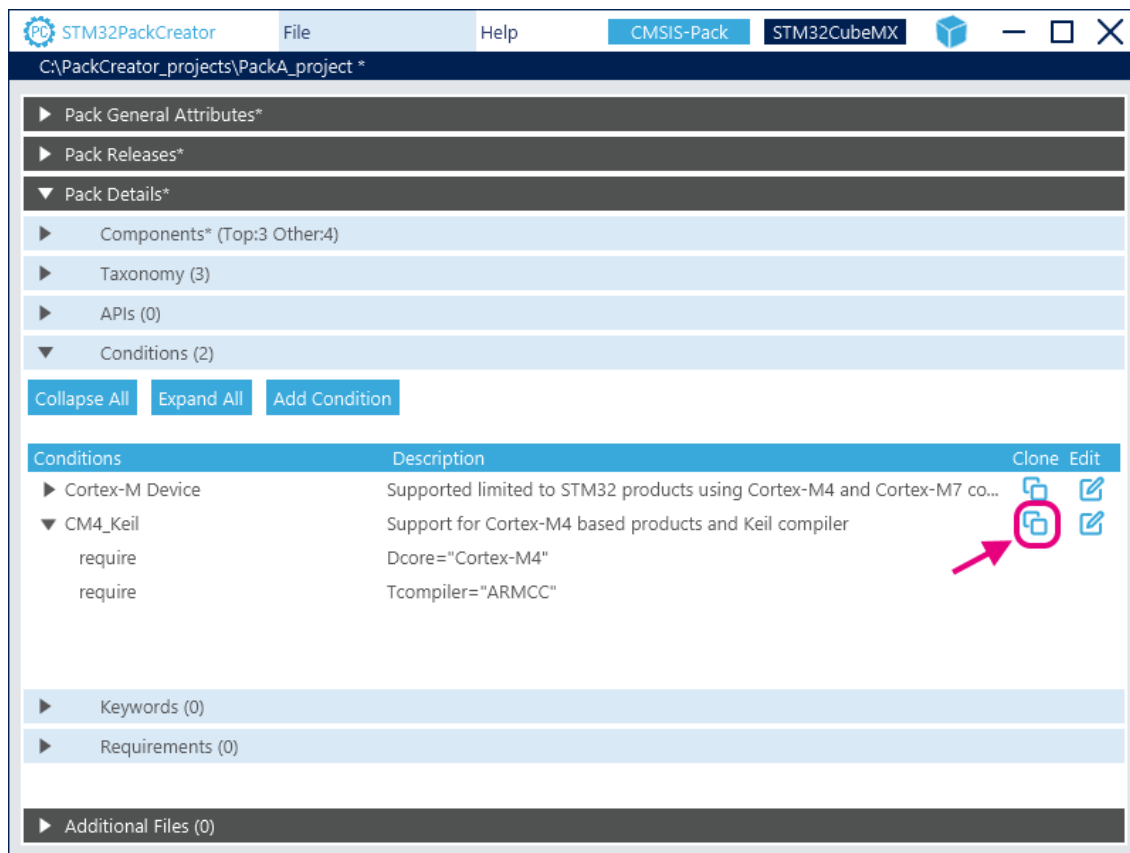
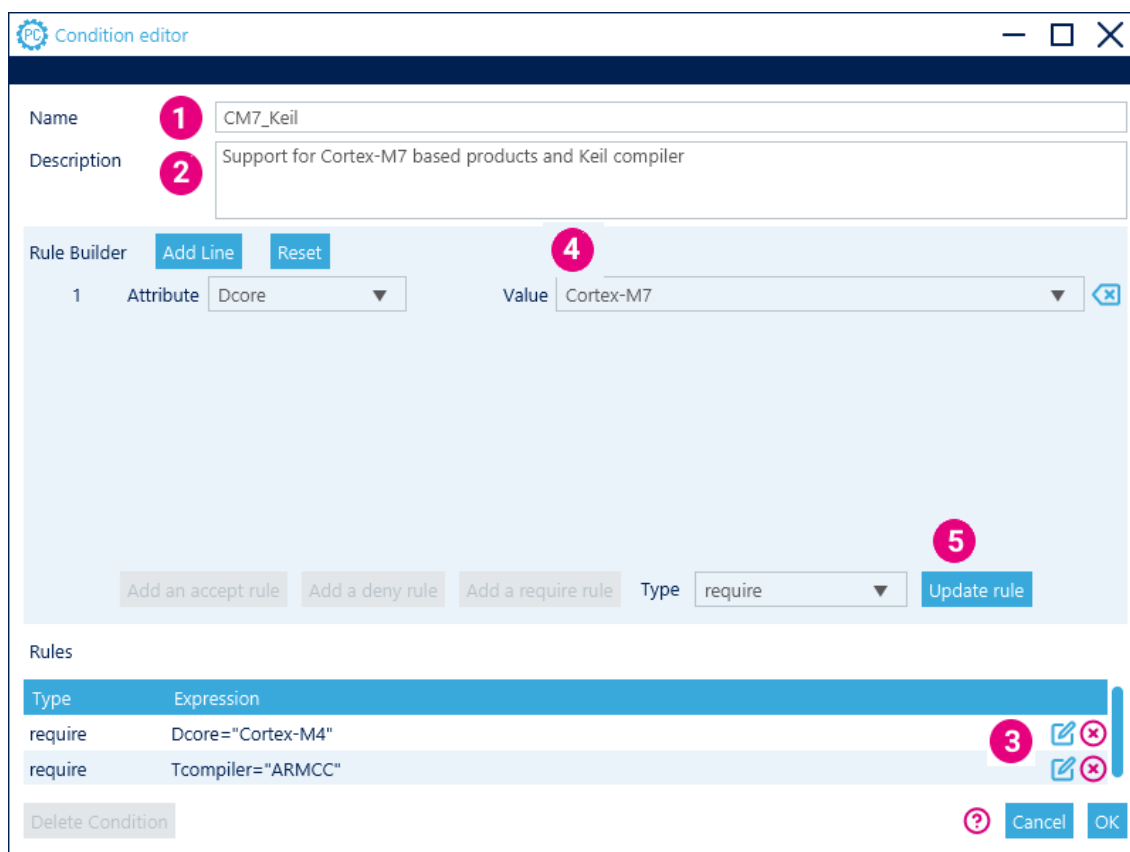


Figure 44. CM7_Keil condition



Click OK to close.

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\PackCreator_projects\PackA_project *

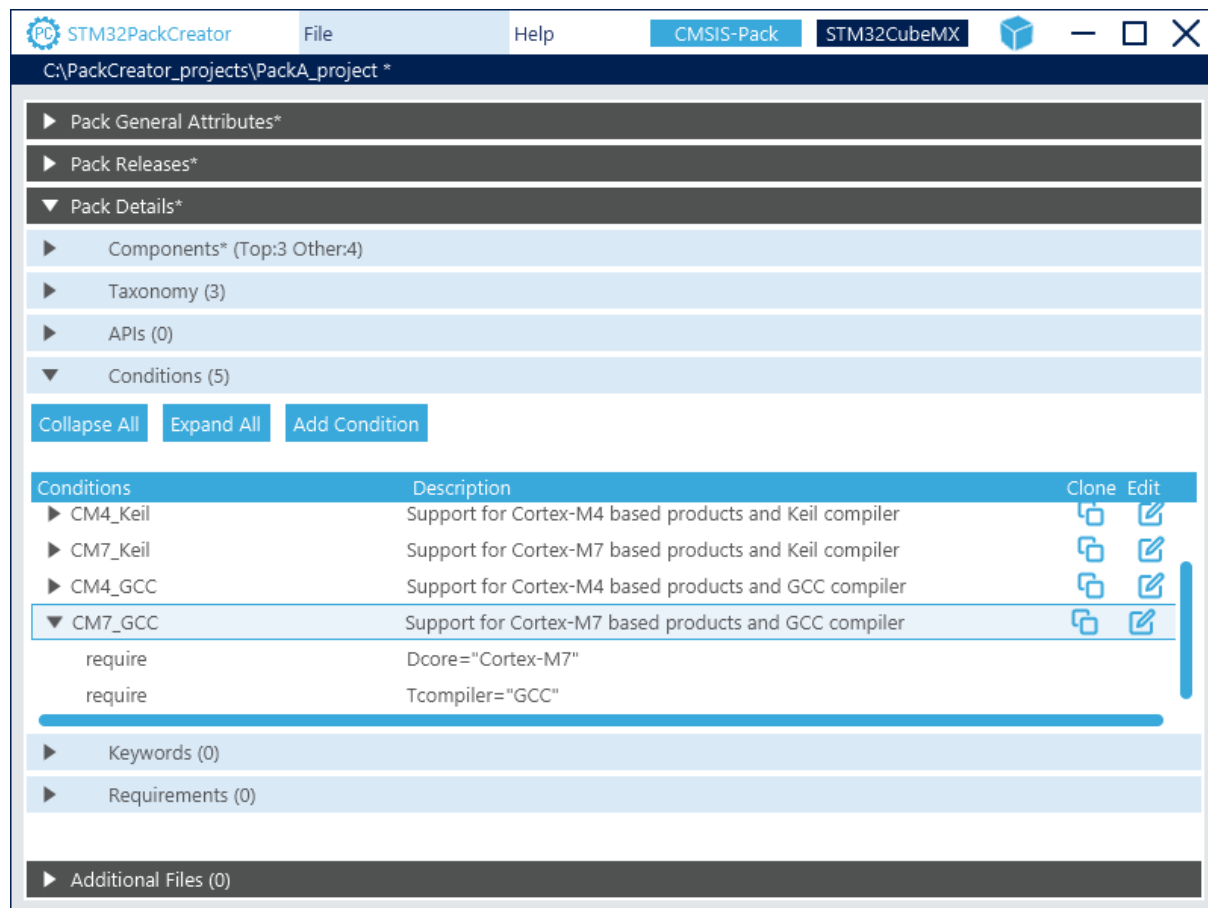
- ▶ Pack General Attributes*
- ▶ Pack Releases*
- ▼ Pack Details*
 - ▶ Components* (Top:3 Other:4)
 - ▶ Taxonomy (3)
 - ▶ APIs (0)
 - ▼ Conditions (3)

Collapse All
Expand All
Add Condition

| Conditions | Description | Clone | Edit |
|-------------------|------------------------------------------------------------------------|-------|------|
| ▶ Cortex-M Device | Supported limited to STM32 products using Cortex-M4 and Cortex-M7 c... | | |
| ▶ CM4_Keil | Support for Cortex-M4 based products and Keil compiler | | |
| ▼ CM7_Keil | Support for Cortex-M7 based products and Keil compiler | | |
| require | Dcore="Cortex-M7" | | |
| require | Tcompiler="ARMCC" | | |
 - ▶ Keywords (0)
 - ▶ Requirements (0)
 - ▶ Additional Files (0)

Proceed similarly to create two more conditions on Cortex®-M4 and GCC compiler, then on Cortex®-M7 and GCC compiler.

Figure 45. Conditions on GCC compiler



5.3.12 Assign CM4_Keil, CM7_Keil, CM4_GCC, and CM7_GCC conditions to component files

Go to the Components sub-section, and click edit on the TopComponent line to open the component editor.

Go to the Component's files tab.

Assign a condition to each of the library files: select it, click `Edit selected file attributes` as shown in Figure 46, and assign the relevant condition (see Figures: assigning conditions on component files). Click OK.

When done for all files, click `Apply` then `Close`.

Save the Project by selecting `File > Save to project`.

Figure 46. Selecting component files to edit file attributes

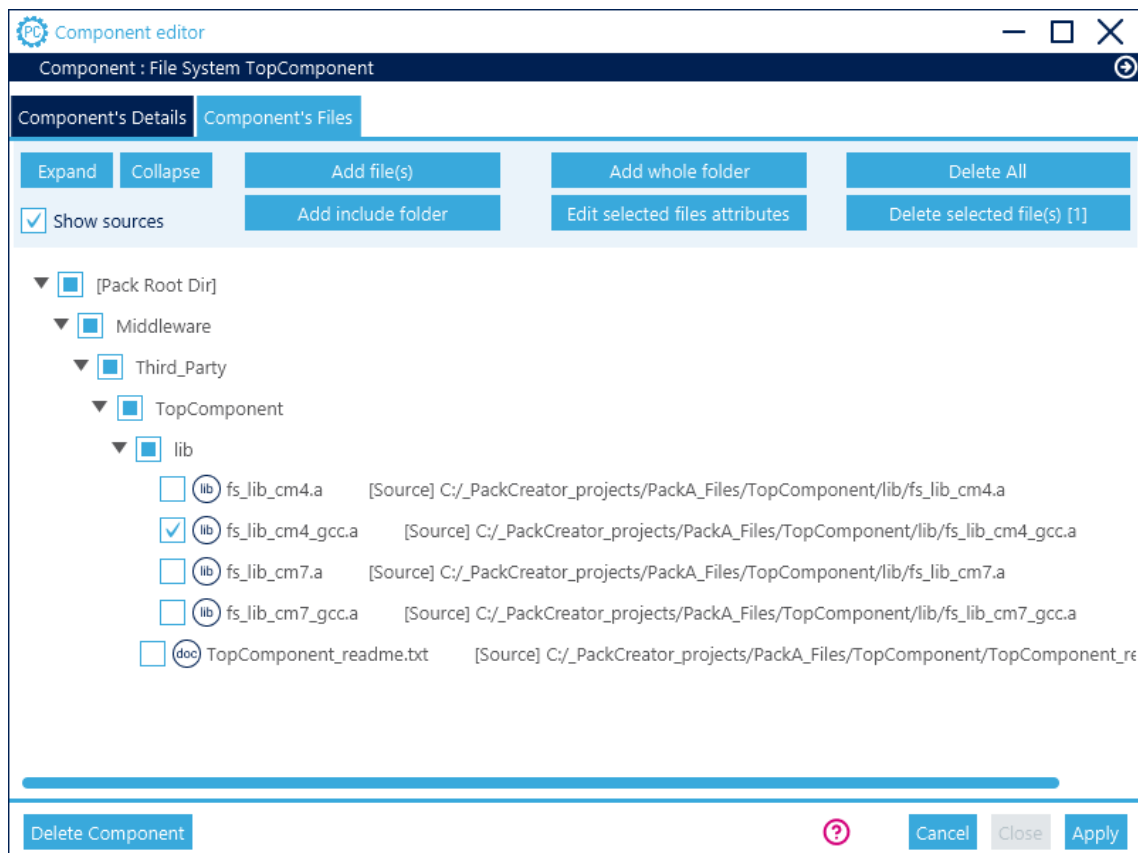


Figure 47. Assigning CM4_GCC condition on file

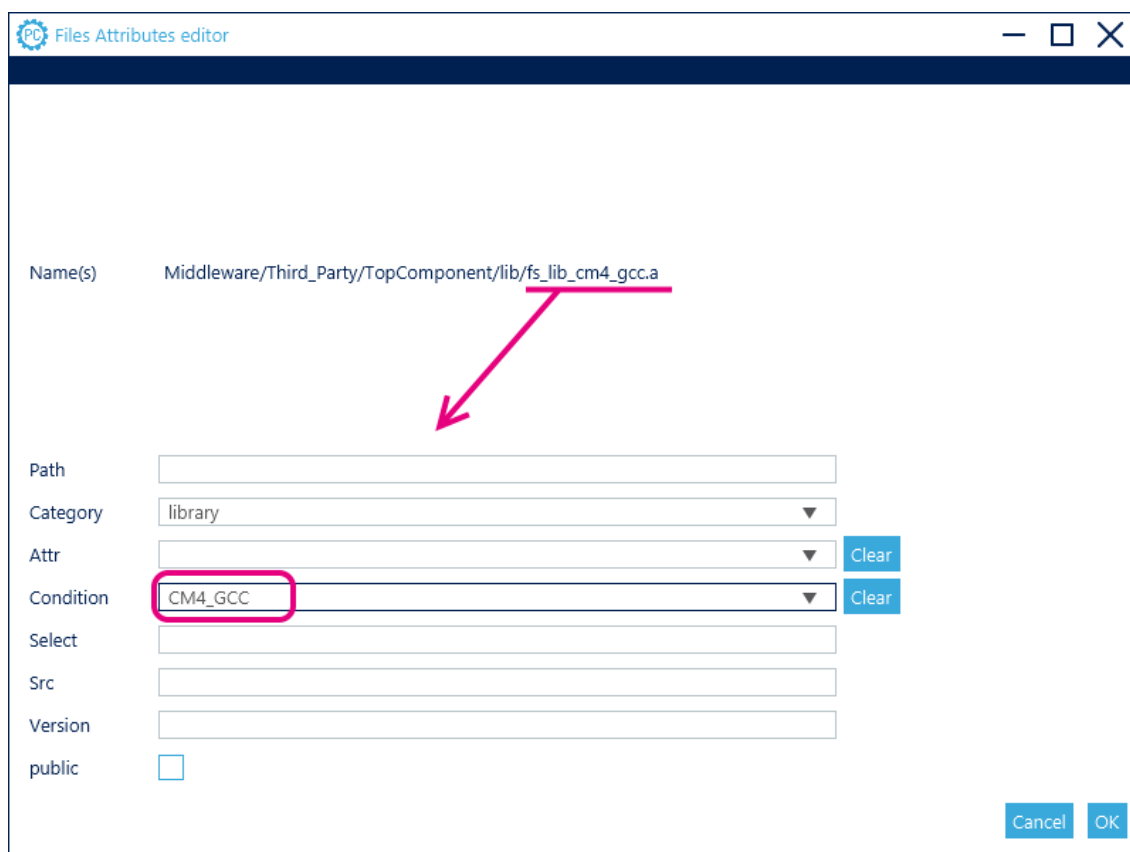


Figure 48. Assigning CM7_Keil condition on file

Files Attributes editor

Name(s) Middleware/Third_Party/TopComponent/lib/fs_lib_cm7.a

Path

Category library

Attr Clear

Condition CM7_Keil Clear

Select

Src

Version

public ☐

Cancel OK

5.3.13 Create conditions on components

Components from the application bundle have the following constraints:

- DataEx_App component requires both components of Bundle1. Bundle1_condition is created for this purpose.
- DataEx_App component requires both components of Bundle1 and TopComponent. Bundle1_TopComponent_condition is created for this purpose.

From the Conditions sub-section, Click Add Condition to open the condition editor. Proceed as shown on Figure 49.

Figure 49. Creating a condition for Bundle 1

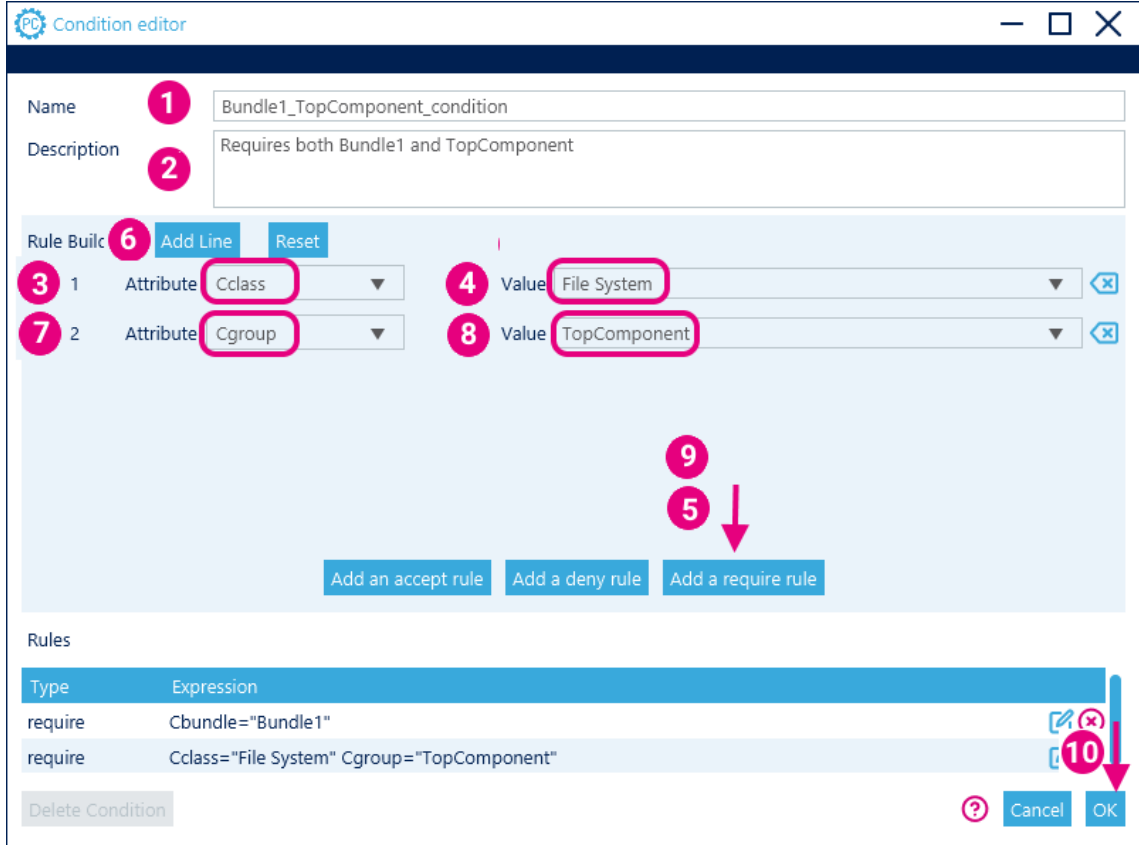
The screenshot shows the 'Condition editor' window with the following elements and annotations:

- Name:** Bundle1_condition (annotated with 1)
- Description:** Bundle1 components are required (annotated with 2)
- Rule Builder:**
 - Attribute:** Cbundle (annotated with 3)
 - Value:** Bundle1 (annotated with 4)
- Buttons:** Add an accept rule, Add a deny rule, Add a require rule (annotated with 5 and a downward arrow)
- Rules Table:**

| Type | Expression |
|---------|-------------------|
| require | Cbundle="Bundle1" |
- Footer:** Delete Condition, Cancel, OK

Click the clone icon  to clone the newly created condition and proceed as shown in Figure 50.

Figure 50. Creating a condition for Bundle 1 and TopComponent



Condition editor

Name **1** Bundle1_TopComponent_condition

Description **2** Requires both Bundle1 and TopComponent

Rule Builder **6** Add Line Reset

3 1 Attribute **7** Cclass **4** Value File System **8** Value TopComponent

9 5 Add an accept rule Add a deny rule Add a require rule

Rules

| Type | Expression |
|---------|--------------------------------------------|
| require | Cbundle="Bundle1" |
| require | Cclass="File System" Cgroup="TopComponent" |

Delete Condition

Cancel OK **10**

5.3.14 Assign conditions on application components

Go back to the Components section and click the edit icon to edit the DataEx_App component. Proceed as shown in Figure 51.

Figure 51. Assigning conditions to the DataEx_App component

The screenshot shows the 'Component editor' window for 'DataEx_App'. The 'Component's Details' tab is active. The 'Condition' field is highlighted with a red circle and a red arrow pointing to it. The 'Apply' button at the bottom right is also highlighted with a red arrow.

| Component : Device Application - DataEx_App | |
|-----------------------------------------------------------------------------------------------------------|------------------------------------------|
| <div> <div>Component's Details</div> <div>Component's Files</div> </div> | |
| Vendor* | VendorA |
| Bundle | Device AppBundle |
| Class* | Device |
| Group* | Application |
| Subgroup | See (?) for help |
| Description* | DataExchange application |
| Version* | 1.0.0 |
| ModuleName* | dataex |
| Condition | Bundle1_condition |
| Variant | DataEx_App |
| | <input type="checkbox"/> Default Variant |
| RTE_Components | |
| Max Instances | 1 |
| Api Version | MAJOR.MINOR.PATCH[-Pre Rel.][+Bui |
| Generator | |
| | <input type="checkbox"/> Deprecate |
| <div> <div>Delete Component</div> <div>?</div> <div>Cancel</div> <div>Close</div> <div>Apply</div> </div> | |

Go back to the Components section and click the edit icon to edit the DataExFS_App component. Proceed as shown in Figure 52.

Figure 52. Assigning conditions to the DataExFS_App component

Component editor

Component : Device Application - DataExFS_App

Component's Details | Component's Files

Vendor* VendorA

Bundle Device AppBundle

Class* Device Version* 1.0.0

Group* Application

Subgroup See (?) for help ModuleName* dataexfs

Description* DataEx w/ Filesystem application

Condition Bundle1_TopComponent_con...

Variant DataExFS_App ☐ Default Variant

RTE_Components

Max Instances 1

Api Version MAJOR.MINOR.PATCH[-Pre Rel.][+Bui]

Generator ☐ Deprecate

Delete Component ? Cancel Close Apply

5.3.15

Specify the example projects to be delivered with the pack

Pre-requisite: a folder holding the example project files for a given board and one or more toolchains.

From the CMSIS pack view:

- Expand the *pack details* section.
- Expand the *Examples* section (refer to Figure 53).
- Click *Add example* to open the example creation window (refer to Figure 54).

From the example creation panel:

- Click ? to open the Help panel and be guided through the process of describing the example project.
- Update the fields describing the example project (refer to Figure 55) such as board name, example name, and example folder.
 - STM32PackCreator places the example folder in the pack under Projects\<Board name>\Applications\<Example name>
- STM32PackCreator parses the example folder for project files and proposes for each a default toolchain association (refer to Figure 56).
- Click Apply to close the window and go back to the CMSIS-Pack main panel (refer to Figure 8).

Figure 53. Adding an example project to the pack

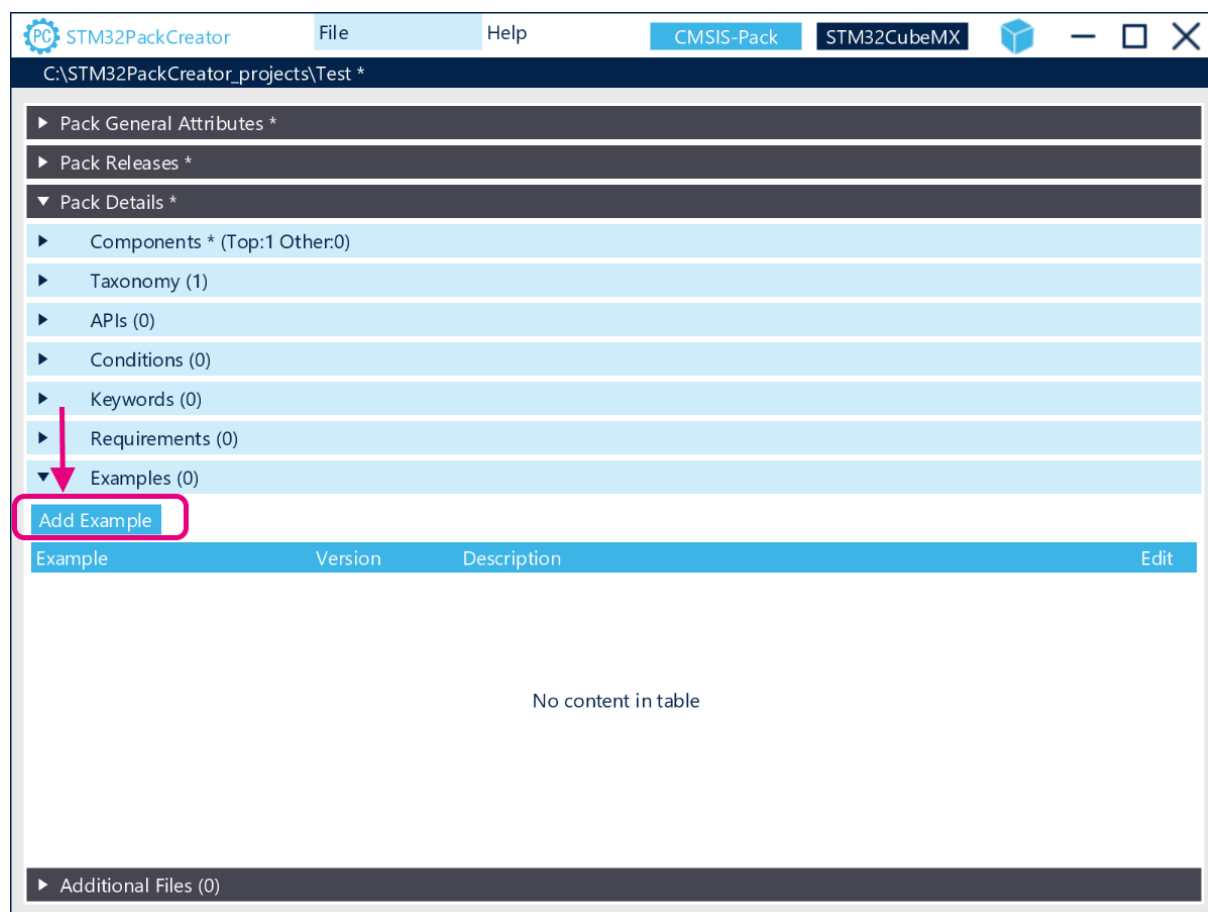


Figure 54. Specifying example details – part 1

Example editor

Name * 1 SampleApp

Board Name * 3 STM32F401RE-Nucleo

Folder Source * 5 C:\STM32PackCreator_project Browse

Device Vendor 6 STMicroelectronics:13

Doc * 8 readme.txt Browse

Description * 9 Sample application project

Type * 2 Applications

Board Vendor 4 STMicroelectronics

Destination * Projects/STM32F401RE-Nucleo/Applicatio

Version 7 1.0.0

Archive Browse

Public ☐

Keyword Automatically discovered from the example source folder

Category

Project files *

| Project Folder | Project File | Toolchain |
|----------------|-------------------|--------------|
| EWARM | SampleApp.ewp | iar |
| MDK-ARM | SampleApp.uvprojx | uv |
| SampleApp | SampleApp.ioc | stm32cubemx |
| STM32CubeIDE | .cproject | stm32cubeide |

Components

Pack Components

- Data Exchange Comp1
- Data Exchange Comp2
- Device Application - DataEx_App
- Device Application - DataExFS_App
- File System TopComponent

Components used in Example

Delete Example

Get Help → ? Cancel Close Apply

Figure 55. Specifying example details – part 2

Example editor

Name *
SampleApp
Type *
Applications

Board Name *
STM32F401RE-Nucleo
Board Vendor *
STMicroelectronics

Folder Source *
C:\STM32PackCreator_project
Browse
Destination *
Projects/STM32F401RE-Nucleo/Applicatio

Device Vendor
STMicroelectronics:13
Version
1.0.0

Doc *
readme.txt
Browse
Archive
Browse

Description *
Sample application project
Public

Keyword
10
Sample

Category
11
Application

Project files *

| Project Folder | Project File | Toolchain |
|----------------|-------------------|--------------|
| EWARM | SampleApp.ewp | iar |
| MDK-ARM | SampleApp.uvprojx | uv |
| SampleApp | SampleApp.ioc | stm32cubemx |
| STM32CubeIDE | .cproject | stm32cubeide |

Components

Pack Components

+
Data Exchange Comp1
Data Exchange Comp2
Device Application - DataExFS_App
File System TopComponent

Components used in Example

Device Application - DataEx_App

Delete Example
Cancel
Close
13
Apply

Figure 56. New example added to the pack

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\STM32PackCreator_projects\PackA_project *

- ▶ Pack General Attributes *
- ▶ Pack Releases *
- ▼ Pack Details *
 - ▶ Components * (Top:3 Other:4)
 - ▶ Taxonomy (3)
 - ▶ APIs (0)
 - ▶ Conditions (7)
 - ▶ Keywords (0)
 - ▶ Requirements (0)
 - ▼ Examples (1)

Add Example

| Example | Version | Description | Edit |
|-------------------------------------------------|---------|----------------------------|------|
| SampleApp STM32F401RE-Nucleo STMicroelectronics | 1.0.0 | Sample application project | |

▶ Additional Files (0)

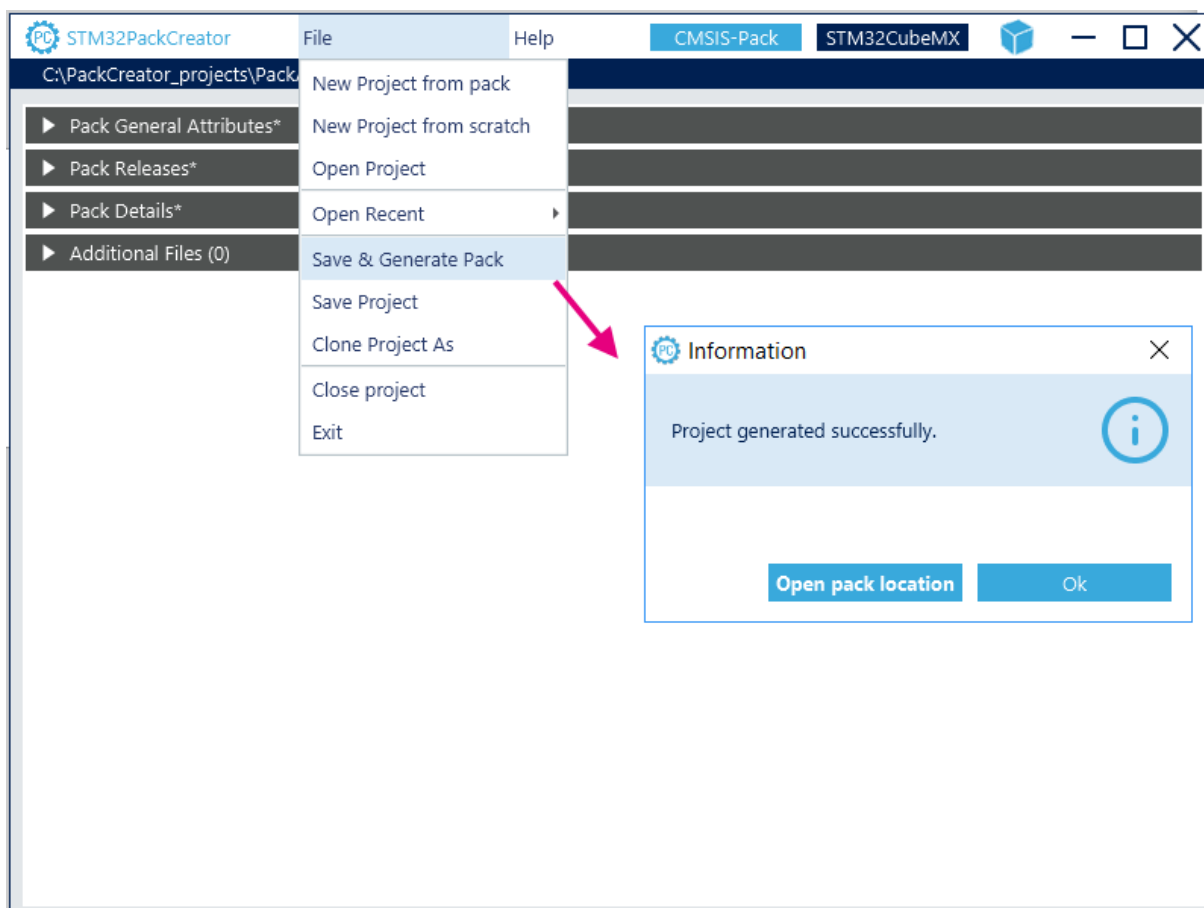
5.4 Save the software pack and use it in STM32CubeMX

5.4.1 Generate and install the pack

Select *File > Save & Generate Pack*.

An information window is displayed and indicates the project is successfully generated. Refer to [Figure 57](#).

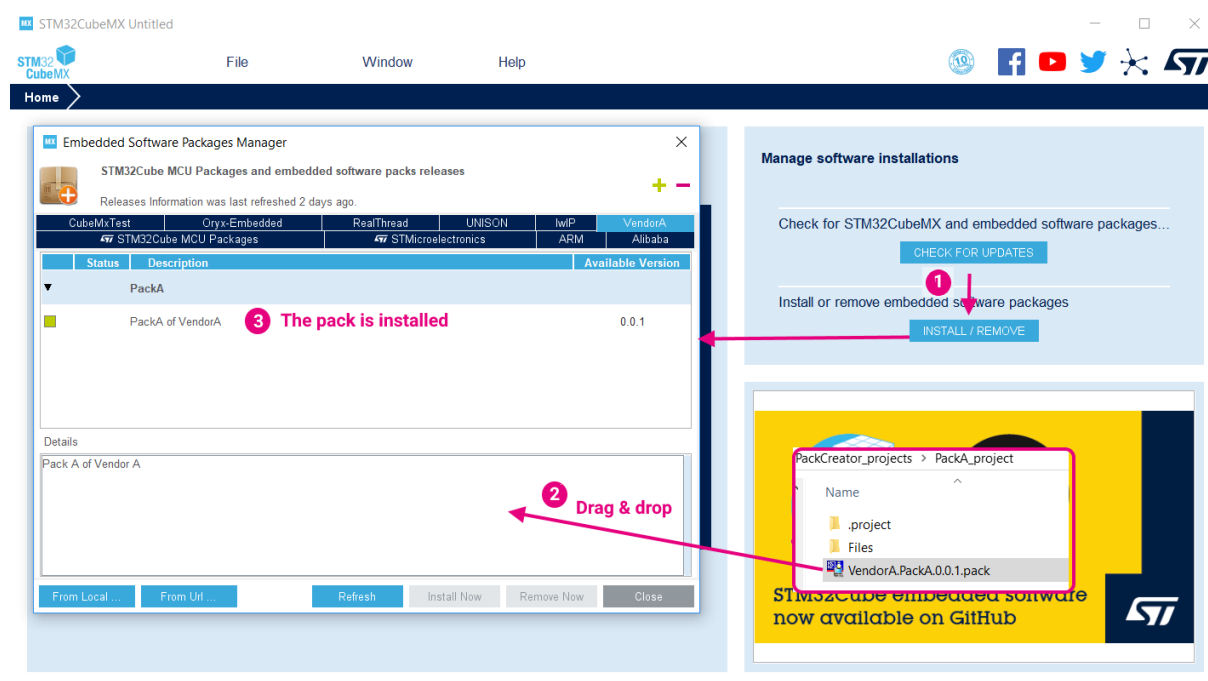
Figure 57. Generating a CMSIS-Pack pack with STM32PackCreator



Click `Open pack location` in the information window.

Install the pack by dragging and dropping the `.pack` file in STM32CubeMX as shown in [Figure 58](#).

Figure 58. Installing the generated pack in STM32CubeMX

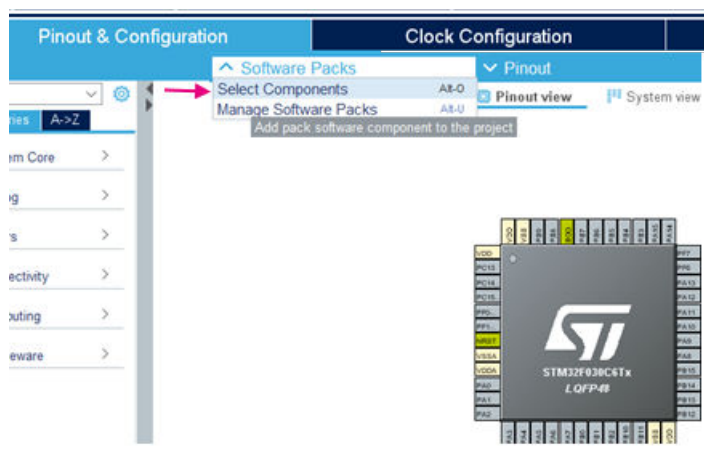


5.4.2 Create a project and check if the pack is available

In STM32CubeMX, start a project by selecting an STM32F0 MCU.

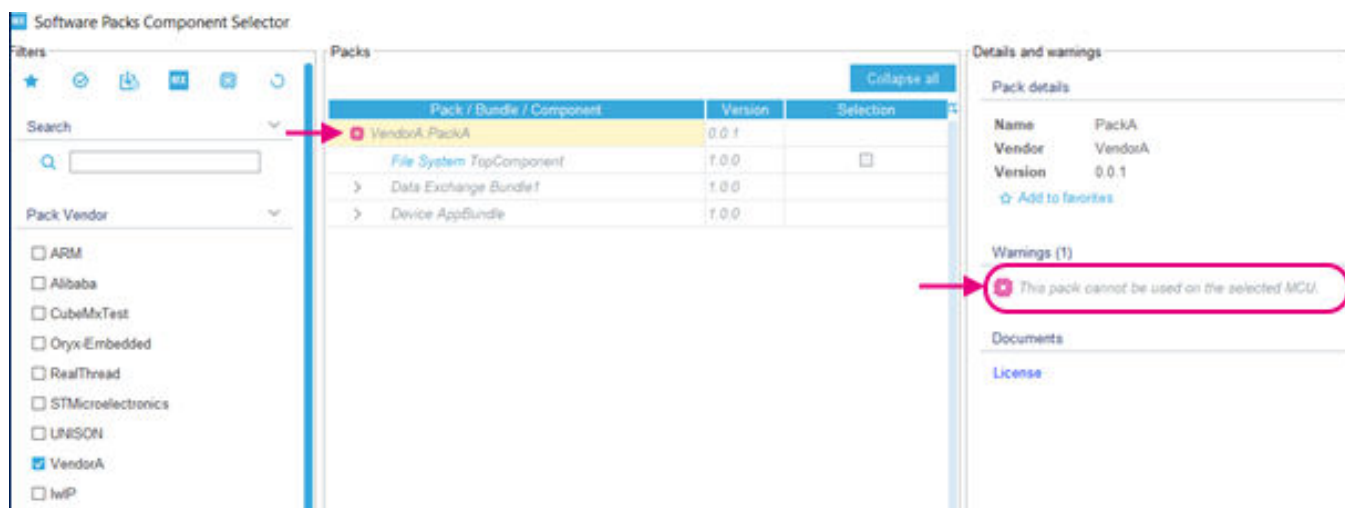
From the Pinout & Configuration view, select Software Packs> Select Components. Refer to Figure 59.

Figure 59. Selecting pack components



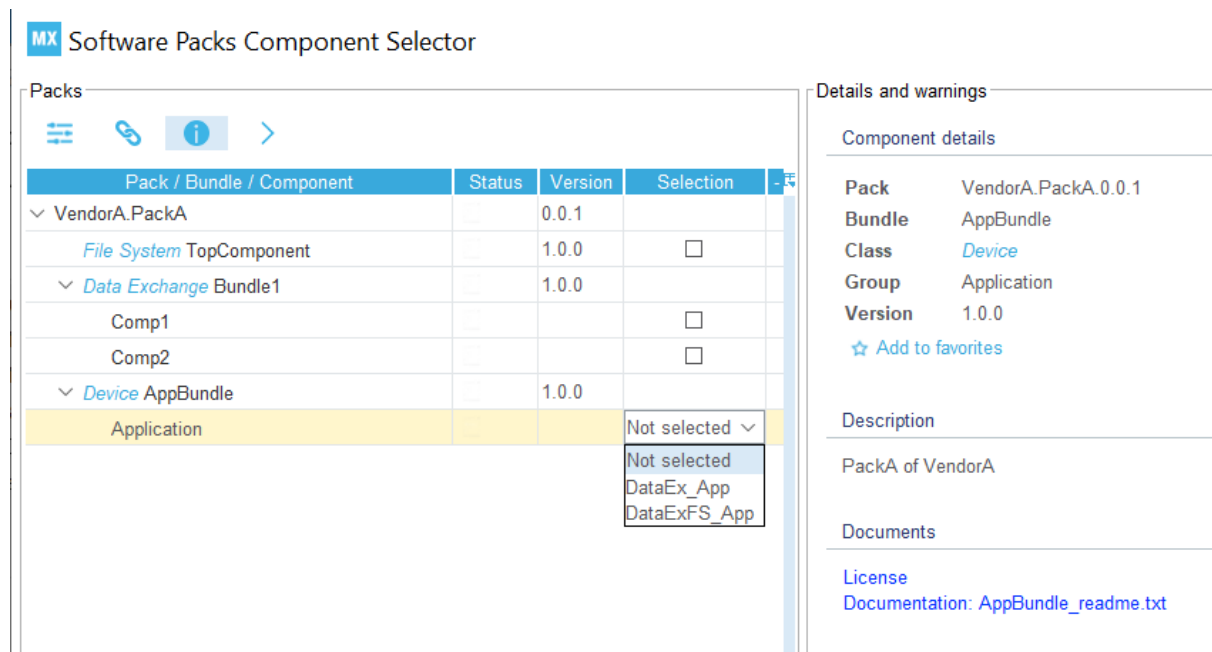
In the component selector, check that the pack components cannot be selected since the Dcore condition is not met: the pack requires an Arm®Cortex®-M4 or Arm®Cortex®-M7 while STM32F0 MCU embeds an Arm®Cortex®-M0 core. Refer to Figure 60.

Figure 60. Pack unavailability when the Dcore condition is not met



Close the project and create a new project for an STM32F4 MCU. From the Pinout & Configuration view, the pack components can now be selected since the Dcore condition is met. Refer to Figure 61.

Figure 61. Pack availability when Dcore condition is met



5.4.3 Select pack components, check and solve dependencies

Still on a project using an STM32F4 MCU, from the component selector window,

1. Enable the *DataExFS_App*.
2. Click the *DataExFS_App* component line to highlight dependencies: Top component and Bundle1 are highlighted as required components. Refer to Figure 62.

Figure 62. Highlighting dependencies on components

The screenshot shows the 'Software Packs Component Selector' window. The main table lists components and their status. The 'Application' component is highlighted, and its dependencies are shown in the 'Component dependencies' section. The 'DataExFS_App' component is selected, and its dependencies are listed as 'Requires: component bundle Bundle1' and 'Requires: component class File System, group TopComponent'. The 'Details and warnings' panel on the right shows the component details for 'VendorA.PackA.0.0.1' and a warning that 'This component has unresolved dependencies. There are solutions within and out of this pack.'

| Pack / Bundle / Component | Status | Version | Selection |
|---------------------------|--------|---------|--------------------------|
| VendorA.PackA | ⚠ | 0.0.1 | |
| File System TopComponent | | 1.0.0 | <input type="checkbox"/> |
| Data Exchange Bundle1 | | 1.0.0 | |
| Comp1 | | | <input type="checkbox"/> |
| Comp2 | | | <input type="checkbox"/> |
| Device AppBundle | ⚠ | 1.0.0 | |
| Application | ⚠ | | DataExFS... |

| Component dependencies | Show | Resolve |
|-------------------------------------------------------------------|------|-----------|
| Component Application DataExFS_App (from bundle Device AppBundle) | | |
| Requires: component bundle Bundle1 | | ⚠ Missing |
| Solutions in VendorA.PackA.0.0.1: | | |
| Component Comp1 | Show | Select |
| Component Comp2 | Show | Select |
| Requires: component class File System, group TopComponent | | ⚠ Missing |
| Solutions in VendorA.PackA.0.0.1: | | |
| Component TopComponent | Show | Select |

Details and warnings

Component details

Pack VendorA.PackA.0.0.1

Bundle AppBundle

Class Device

Group Application

Version 1.0.0

⚠ Add to favorites

Warnings (1)

⚠ This component has unresolved dependencies. There are solutions within and out of this pack.

Description

PackA of VendorA

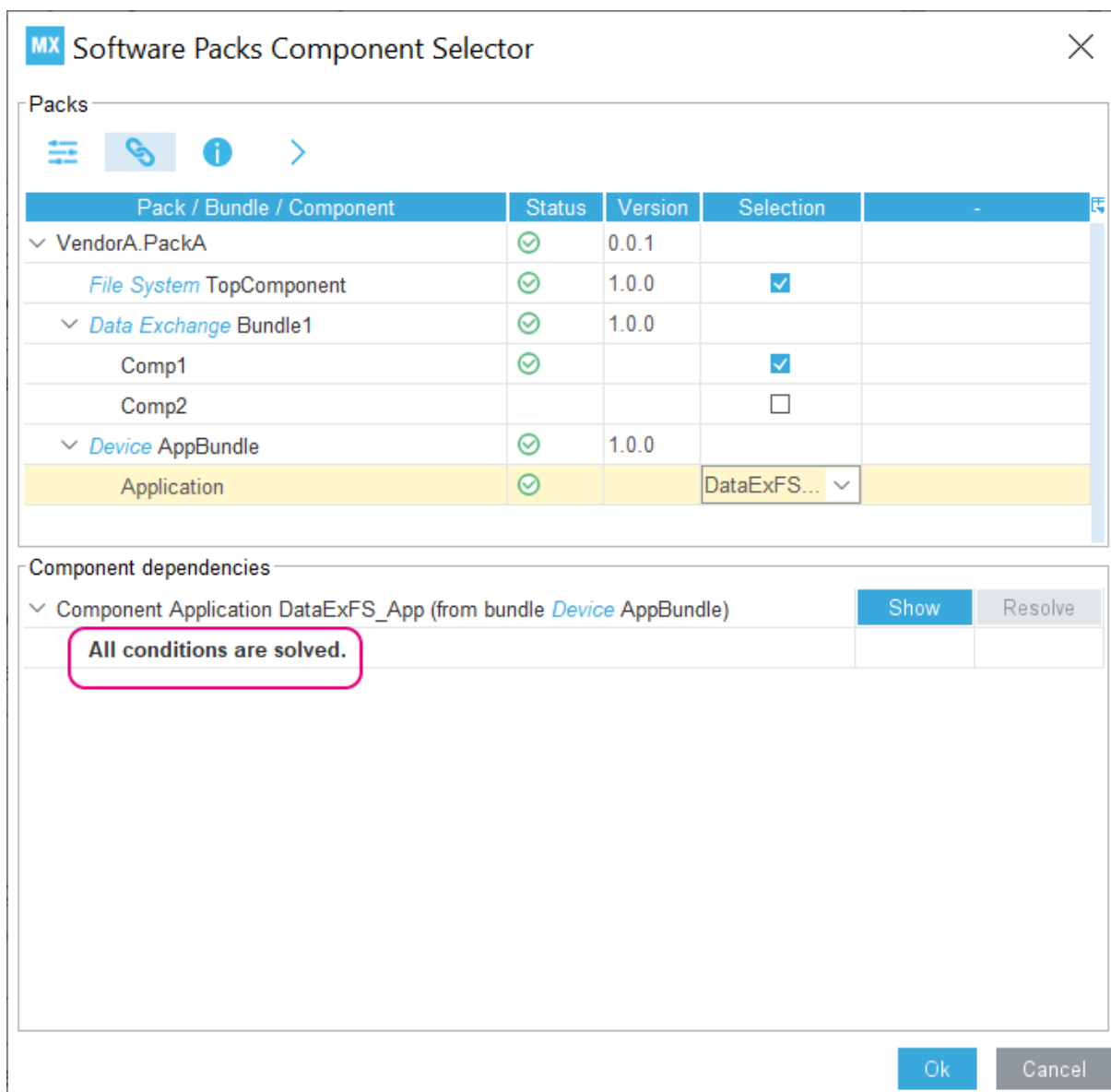
Documents

License

Ok Cancel

3. Click the Resolve button to automatically resolve dependencies: this is possible when only one choice is available. Click to choose a component among others that can solve a dependency. The dependency panel is emptied when no dependency remains to be solved.

Figure 63. Solving one dependency on components

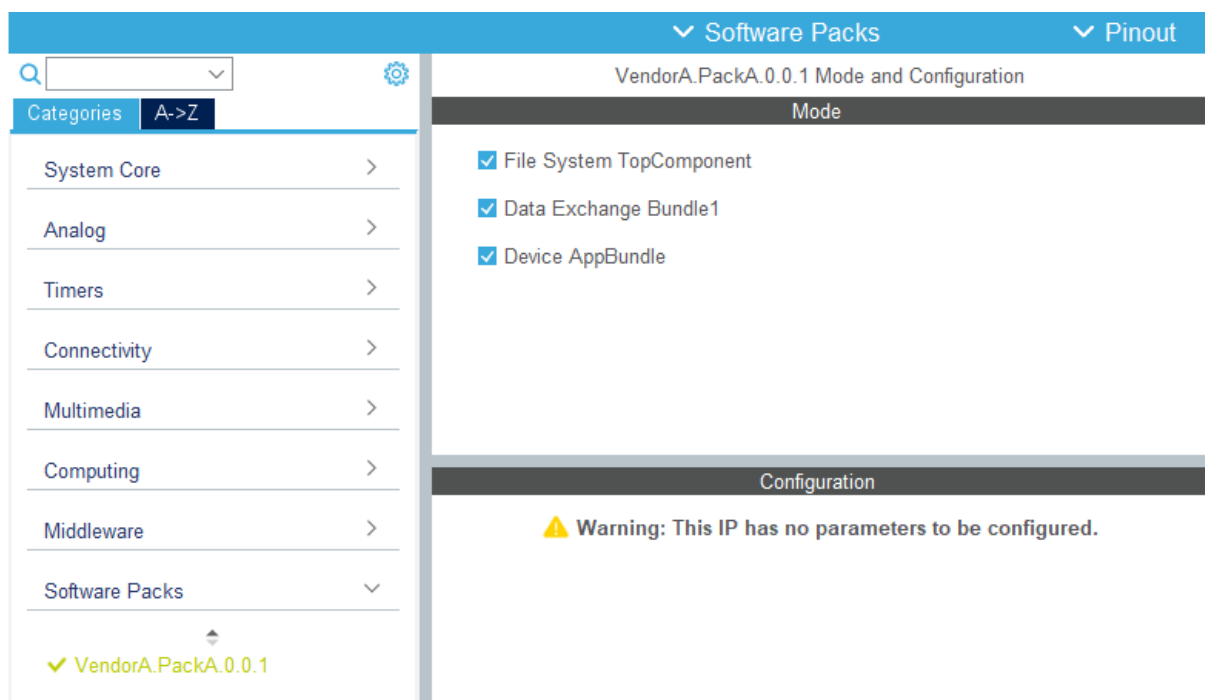


5.4.4 Check the pack in the STM32CubeMX pinout configuration view

From the Pinout & Configuration view, on the left panel expand the Software Packs category.

1. Click the Vendor A Pack A
2. Check that the top components are shown as modes in the Mode panel.
3. Enable all of them and check that there are no configuration parameters in the Configuration panel. Refer to Figure 64.

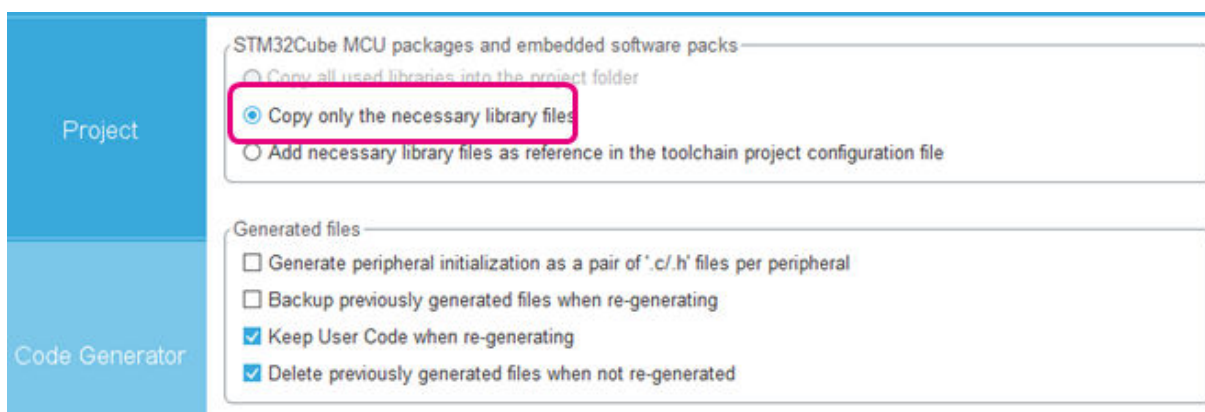
Figure 64. Unconfigurable software pack in STM32CubeMX user interface



5.4.5 Generate the project

1. Go to the Project Manager's view.
2. On the Code generator tab, select the option to copy only the necessary library files.

Figure 65. Code generation settings in STM32CubeMX



3. On the **Project** tab, enter a project name and select the **STM32CubeIDE** toolchain.

Figure 66. Project settings in STM32CubeMX

The screenshot shows the 'Project Settings' dialog in STM32CubeMX. The left sidebar has two tabs: 'Project' (selected) and 'Code Generator'. The 'Project' tab contains the following fields:

- Project Name:** pack_not_configurable_project
- Project Location:** C:\STM32CubeMX_Projects
- Application Structure:** Advanced (dropdown menu)
- Toolchain Folder Location:** C:\STM32CubeMX_Projects\pack_not_configurable_project\
- Toolchain / IDE:** STM32CubeIDE (dropdown menu)
- Generate Under Root:** ☒

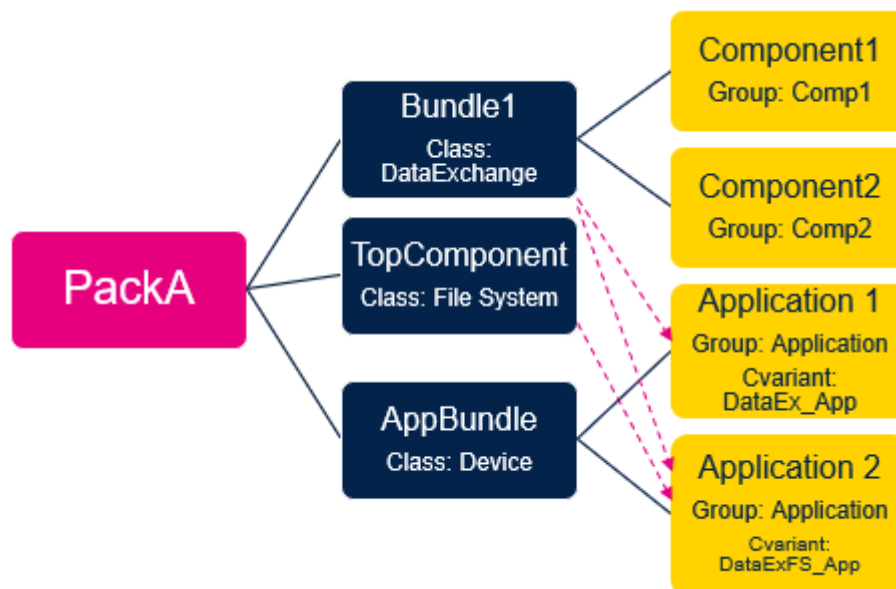
4. Click **GENERATE CODE** to generate the project. If requested, the user must accept to download the STM32Cube MCU Package relevant to the project.
5. Check that the pack files are copied accordingly under the folder `Middlewares\Third_Party`. Specifically, check that among the four library files of TopComponent, only the file `fs_lib_cm4_gcc.a` which has the condition on core Cortex®-M4 and the GCC compiler is copied.
6. Change the toolchain to use Keil® MDK-ARM toolchain and check that this time, only `fs_lib_cm4.a` file is copied since the condition on core Cortex®-M4 and armcc compiler is met.

5.5 Enhance the pack with configuration parameters

5.5.1 Introduction

At this stage, the pack is available and can already be used in STM32CubeMX.

Figure 67. PackA outline

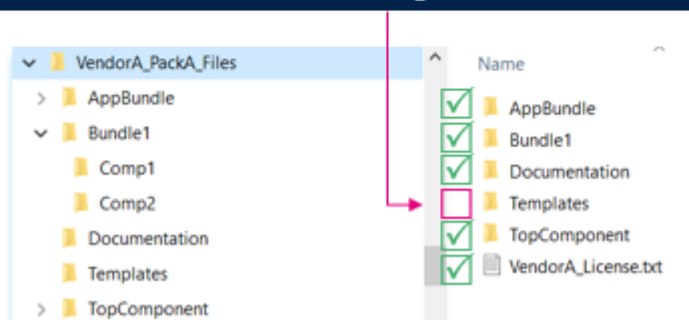


The next step is to enhance the pack to be configurable through STM32CubeMX.

Figure 68. PackA not yet enhanced for STM32CubeMX

| | | | |
|-------------------------------------|---------------------------------------------------------------------|--------------------------|----------------------------|
| <input checked="" type="checkbox"/> | CMSIS-Pack elements | <input type="checkbox"/> | CubeMX enhancements |
| | 1 Bundle with 2 components | | Configuration Parameters |
| | 1 Top component | | Platform Setting |
| | 2 Application variants | | Custom Templates |
| | Conditions on components -----> | | Conditions on parameters |
| | Conditions on component files | | |

Templates will be used at this stage
They are freemarker .ftl files used to generate custom code.

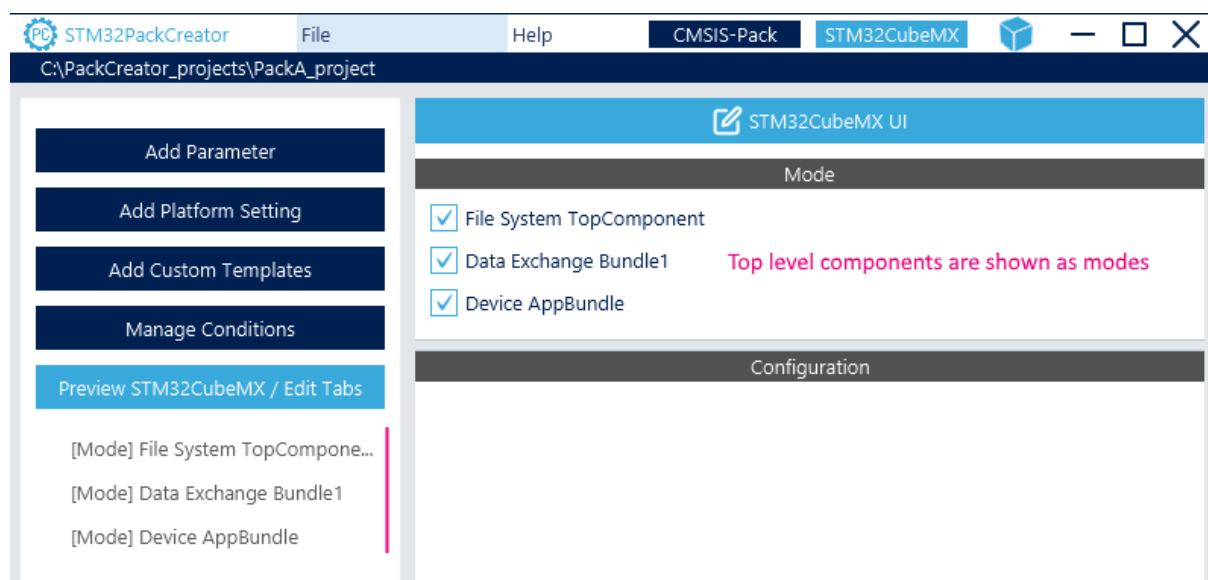


5.5.2 Switch to STM32CubeMx view

Back in the STM32PackCreator project, click **STM32CubeMX** to switch to the view that allows enhancing the pack to be configurable through STM32CubeMX.

Click STM32CubeMX preview and check how the pack currently shows when enabled for a project in STM32CubeMX.

Figure 69. STM32CubeMX preview in STM32PackCreator



5.5.3 Add the first parameter for Bundle1

Create the first parameter as follows. Refer to Figure 70.

1. Enter the parameter name.
 - This name must not contain spaces and be unique per project, so it is advised to include the Pack name into the Parameter name.
2. Choose for which top-level components the parameter may be available.
 - Here: available only if Bundle1 is enabled
 - It can also be available on conditions (detailed later in the slides)
3. Choose the parameter type.
4. Specify a default value.
5. Specify a label name that is used for displaying the parameter in STM32CubeMX UI (configuration panel).
6. Enter a description that is displayed in the configuration panel as well.

Figure 70. Adding the first parameter to Bundle1

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\STM32PackCreator_projects\PackA_project *

Parameter Details

Name * PackA_Param1 **1** ☒ Visible

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|---------------------------|
| File System TopComponent | Never |
| Data Exchange Bundle1 | Never |
| Device AppBundle | Never |
| | Always |
| | Create a new Condition... |

Param condition Expression See (?) for help

Diagnostic Leading or trailing spaces forbidden

Type * String **3** Default value * enabled **4**

Displayed name * ParameterOne **5**

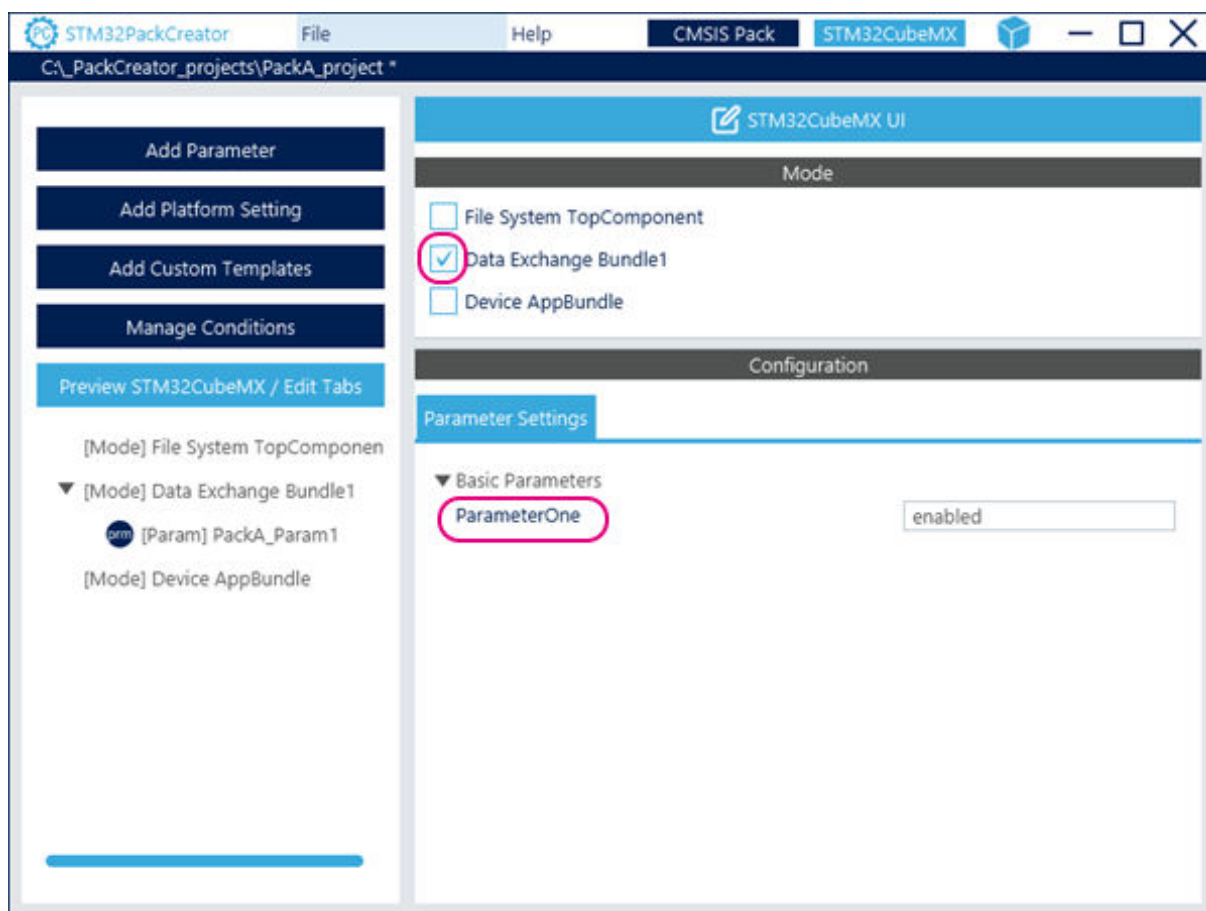
Help message **6** This parameterOne is used to configure ...

Tab * Parameter Settings

Group in Tab * Basic Parameters

Delete **7** Clone Cancel Apply

7. Switch to STM32CubeMX preview to check the parameter appears when Bundle1 is enabled.

Figure 71. Preview the first parameter in STM32CubeMX UI


5.5.4 Add a second parameter for Bundle1 and TopComponent

Create a second parameter as shown in Figure 72.

1. Enter the parameter name
2. Choose for which top-level components the parameter may be available.
 - Select Always for TopComponent and Bundle1.
3. Choose the parameter type.
 - Select Integer.
 - Enter Default, Min, and Max values.
4. Specify the label.
5. Enter a description that is displayed in the configuration panel as well.

Figure 72. Adding the second parameter to Bundle1 and TopComponent

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\STM32PackCreator_projects\PackA_project *

Add Parameter

Add Platform Setting

Add Custom Templates

Manage Mode Conditions

Preview STM32CubeMX / Edit Tabs

[Mode] File System TopComponent

▼ [Mode] Data Exchange Bundle1

 [Param] PackA_Param1

[Mode] Device AppBundle

Parameter Details

Name * PackA_Param2

Visible

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|-------------------------|
| File System TopComponent | Always |
| Data Exchange Bundle1 | Always |
| Device AppBundle | Never |

Param condition Expression See (?) for help

Diagnostic Leading or trailing spaces forbidden

Type * Integer

Default value * 10

Min * and Max * 1 20

Value Condition Expression See (?) for help

Diagnostic Leading or trailing spaces forbidden

Displayed name ParameterTwo

Help message This parameter will be used to configure... It is available for both Bundle1 and TopComponent.

Tab * Parameter Settings

Group in Tab * Basic Parameters

Delete

Clone Cancel Apply

6. Switch to STM32CubeMX preview. Refer to Figure 73. ParameterTwo is displayed when either one of Bundle1 or TopComponent are selected.

Figure 73. Preview the second parameter in STM32CubeMX UI

The figure consists of two screenshots of the STM32CubeMX UI, showing the configuration of parameters. Both screenshots have a blue header bar with the STM32CubeMX UI logo and a 'Mode' section with three radio buttons: 'File System TopComponent', 'Data Exchange Bundle1', and 'Device AppBundle'. The 'Parameter Settings' section is expanded, showing 'Basic Parameters' with two parameters: 'ParameterTwo' and 'ParameterOne'. In the top screenshot, 'File System TopComponent' is selected, and 'ParameterTwo' is highlighted with a red circle, showing a value of 10. In the bottom screenshot, 'Data Exchange Bundle1' is selected, and 'ParameterTwo' is highlighted with a red circle, showing a value of 10, while 'ParameterOne' is also visible, showing a value of 'enabled'.

5.5.5 Add the third parameter for TopComponent in the user-defined tab and group

Create a third parameter as follows. Refer to Figure 74.

1. Enter the parameter name.
2. Select `Always` for TopComponent.
3. Choose the parameter type as a read-only string. Set its default to `enabled`.
4. Specify its label.
5. Enter its description.
6. For the Tab, select `Create a new Tab` and name it `Advanced Settings`.
7. For the Group in Tab, select `Create a new Group` and name it `Advanced Parameters`.

Figure 74. Adding the third parameter to TopComponent in the user-defined Tab and group

Parameter Details

Name * 1 ☒ Visible

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|----------------------------------------------------|
| File System TopComponent | 2 <input type="text" value="Always"/> |
| Data Exchange Bundle1 | <input type="text" value="Never"/> |
| Device AppBundle | <input type="text" value="Never"/> |

Param condition Expression

Diagnostic

Type * 3 Default value * 4

Displayed name * 5

Help message 6

Tab *

Group in Tab *

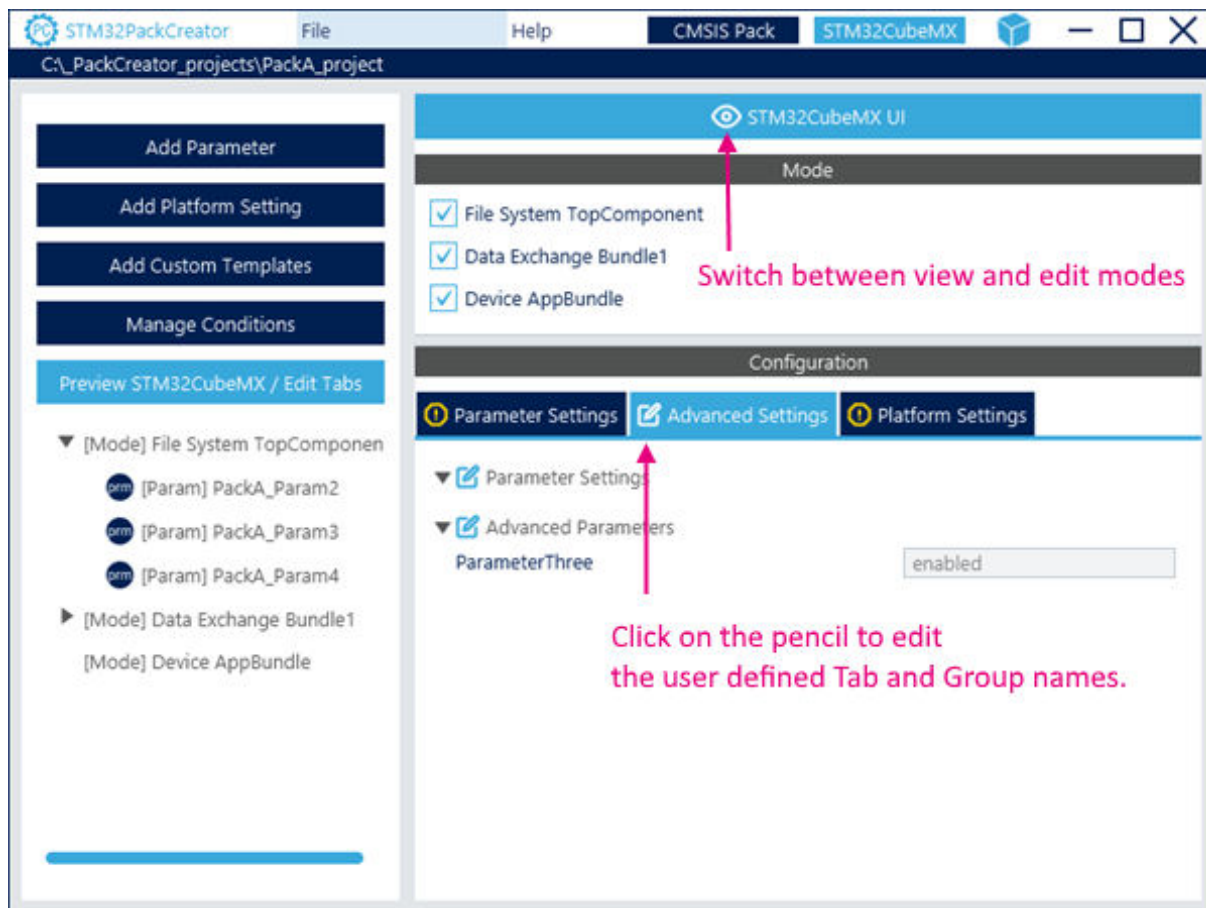
New group

Tab 7

Group Name

8. Switch to STM32CubeMX preview and click between pencil and eye icon to switch between edit and view modes. Refer to Figure 75.

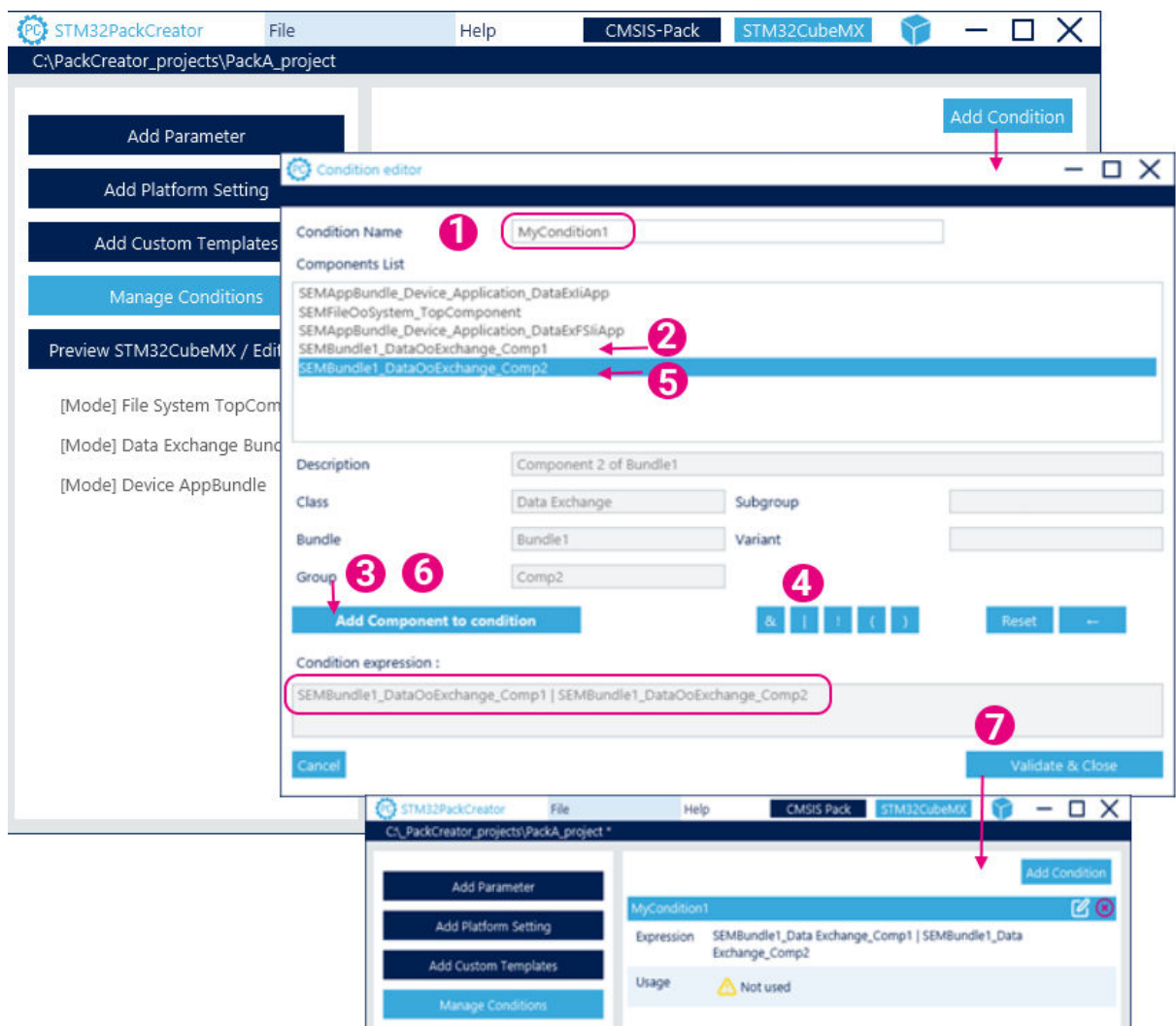
Figure 75. Preview the third parameter in STM32CubeMX UI



5.5.6 Create a condition and assign it to a parameter

Create the condition that comp1 or comp2 of Bundle1 must be present.

- Select **Manage Mode Conditions** on the left panel to open the condition view and click **Add Condition** to open the condition editor.
- Enter the condition name.
- Select the first component (Comp1).
- Add it to the condition.
- Select the OR logical operator: "|".
- Select the second component (Comp2).
- Add it to the condition.
- Validate & Close.



5.5.7 Create a fourth conditional parameter

Create a fourth parameter that is visible for the TopComponent only if a condition is satisfied.

1. Enter the parameter details.
2. Assign it to TopComponent on the condition MyCondition1 is met. Refer to Figure 76.

Figure 76. Creating parameters with conditions

Parameter Details

Name *

PackA_Param4

Visible

STM32CubeMX Mode

File System TopComponent

Data Exchange Bundle1

Device AppBundle

Condition of Assignment

MyCondition1

MyCondition1

Never

Always

Create a new Condition...

Param condition

Expression

See (?) for help

Diagnostic

Leading or trailing spaces forbidden

Type *

List

Default value *

Enabled

Possible Values *

Enabled = Enabled

Disabled = Disabled

Displayed name *

ParameterFour

Help message

This parameter is used to configure ...

It is available for TopComponent if either one of Bundle1 comp

Tab *

Advanced Settings

Group in Tab *

Advanced Parameter Setti...

Delete

Clone

Cancel

Apply

3. Create a list parameter: click the + sign to add values, cross the icon to delete, and the check mark to change the default value. Refer to Figure 77.

Figure 77. Creating a parameter list

5.6 Enhance the pack with platform settings

5.6.1 Platform settings introduction

STM32PackCreator allows pack developers to specify platform requirements for the pack components by clicking

Add Platform Setting

Platform settings are optional and may be used to add BSP constraints to the usage of a software component.

- Example: to work, a software component requires an SPI configured in full-duplex master mode or a GPIO in EXTI mode.

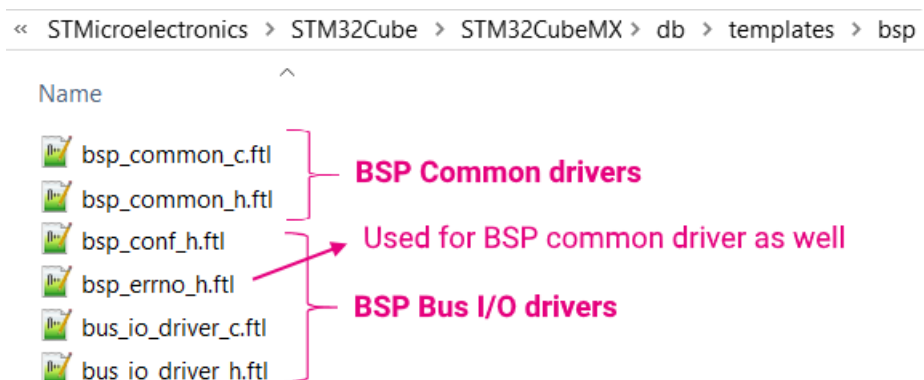
STM32Cube specifications define a set of Board Support Package APIs among which the BSP Bus and BSP Common driver APIs.

- The BSP bus driver API, which exports the transport functions used by the I/O operation components. They can be shared by several IO operations of several external components.
- The BSP common driver API, which provides a set of friendly used APIs for HMI devices, like LEDs, buttons, and joystick, and the board COM ports. The BSP common services are defined only if the corresponding hardware is available on the board.

STM32CubeMX embeds some default templates to generate the BSP Bus I/O and BSP Common drivers files, refer to Figure 78, and can generate accordingly:

- BSP bus driver code for I2C, USART, UART, LPUART, and SPI peripherals.
- BSP common driver code for LED (GPIO_OUT), Button (GPIO_EXTI), USART/UART/LPUART

Figure 78. STM32CubeMX Freemarker templates for platform settings



The names for the generated files and BSP folder uses STM32 board names when the project is started from STM32CubeMX board selector. Refer to Figure 79 and Figure 80. Otherwise, the `custom` convention is used, like `custom_bus.h` file.

The BSP common driver code is found under the `Drivers/BSP` folder. Other BSP code is generated in the user's folder.

Figure 79. STM32CubeMX generated BSP common driver files

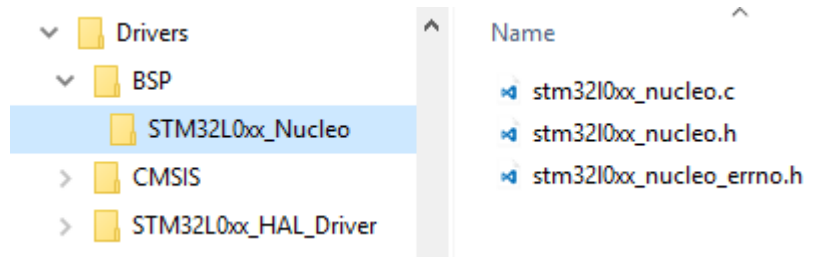


Figure 80. STM32CubeMX generated Bus I/O driver files

```
stm32l0xx_nucleo_bus.c
stm32l0xx_nucleo_bus.h
stm32l0xx_nucleo_conf.h
```

Pack developers can write custom Freemarker templates for CubeMX to generate specific code for BSP bus I/O drivers and other custom BSP APIs. Such templates can be added to the pack by clicking

Add Custom Templates

and then assigned to the relevant platform setting entry.

STM32PackCreator platform settings creation view

In line with STM32CubeMX possibilities, STM32PackCreator allows specifying platform settings for the BSP bus (refer to Figure 81), BSP Common (refer to Figure 82), and custom BSP API (refer to Figure 83).

Figure 81. Platform settings for Bus I/O API

The screenshot shows the STM32PackCreator application window with the 'Platform Setting Details' dialog open. The dialog is titled 'Platform Setting Details' and has a tab for 'BSP_BUS_DRIVER'. The 'Api' dropdown is set to 'BSP_BUS_DRIVER'. The 'Name' field is empty, and the 'Group' dropdown is also empty. Below this, there is a table with two columns: 'STM32CubeMX Mode' and 'Condition of Assignment'. The table has two rows: 'File System TopComponent' with 'Never' in the condition, and 'Data Exchange Bundle1' with 'Never' in the condition. Below the table, there is an 'Add Filter' button. The 'Filter' section shows a list of filters: 'Ip' (USART), 'Mode' (Asynchronous), and 'Help message' (USART, UART[USART]LPUART, UART, I2C, SPI, LPUART). The 'Delete' button is at the bottom left, and 'Cancel' and 'Apply' buttons are at the bottom right.

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|-------------------------|
| File System TopComponent | Never |
| Data Exchange Bundle1 | Never |

Figure 82. Platform settings for BSP common API

STM32PackCreator File Help CMSIS-Pack STM32CubeMX

C:\PackCreator_projects\PackA_project

Add Parameter

Add Platform Setting

Add Custom Templates

Manage Conditions

Preview STM32CubeMX / Edit Tabs

- [Param] PackA_Param4
- [Platform] PackA_platform
- ▼ [Mode] Data Exchange Bundle1
 - [Param] PackA_Param1
 - [Param] PackA_Param2
 - [Platform] PackA_Reset_Li
 - [Template] BUNDLE1_COI
- ▼ [Mode] Device AppBundle
 - [Platform] BSP LED PackA
 - [Platform] BSP BUTTON P

Platform Setting Details

Api: BSP_COMMON_DRIVER Type:

Name: Group:

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|-------------------------|
| File System TopComponent | Never |
| Data Exchange Bundle1 | Never |

Add Filter

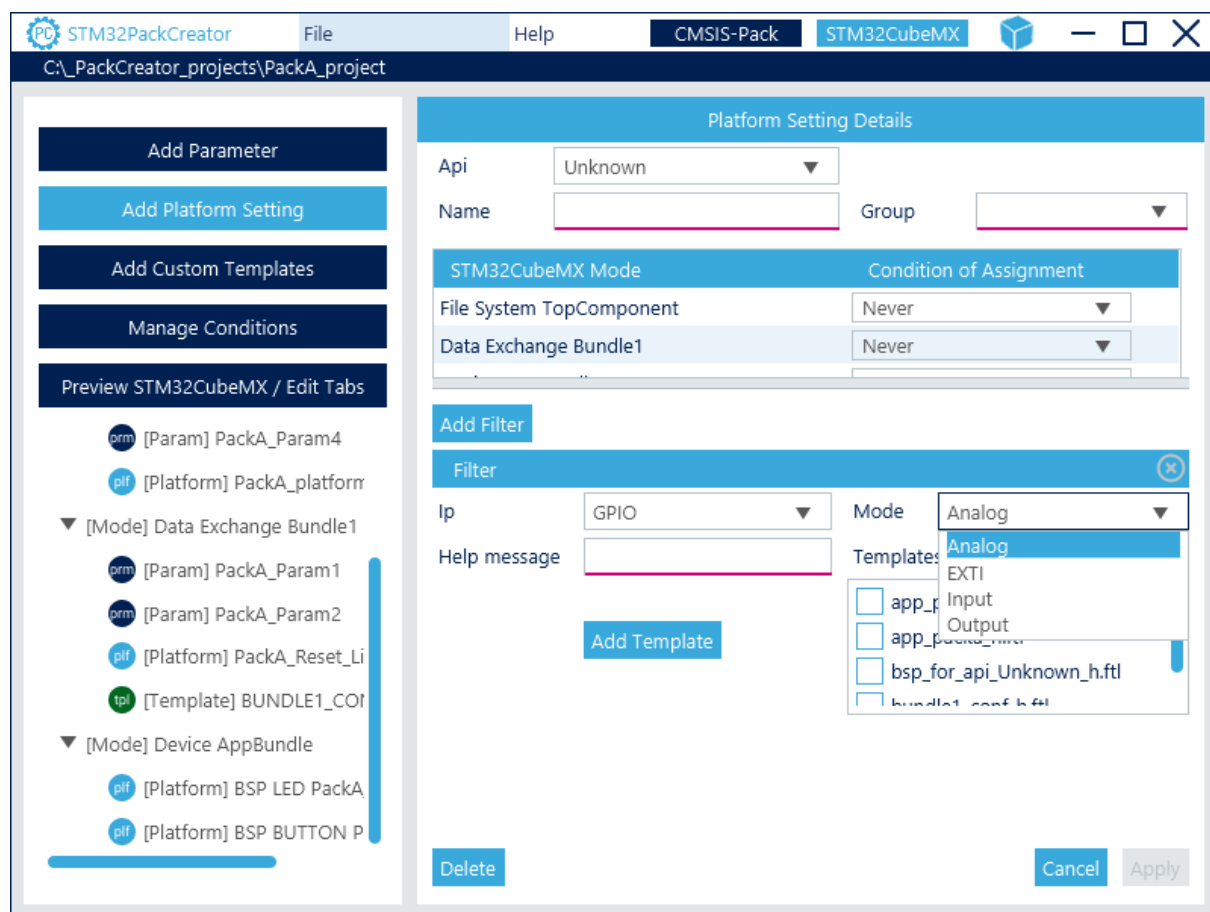
Filter

Ip: USART Mode: Asynchronous

Help message: USART
USART|USART|LPUART
USART
LPUART
GPIO

Delete Cancel Apply

Figure 83. Platform settings with custom BSP API



Unknown API advanced feature

The unknown API comes with an advanced feature. It is possible to edit the IP and mode field to specify peripherals and peripheral modes other than the default GPIO and GPIO modes. It requires the use of the exact STM32CubeMX naming conventions used in `<ip name>_modes.xml` files found in STM32CubeMX database.

- Example for Timer 1 up to Timer 8 peripherals: IP = `TIM1_8`, Mode = `Internal Clock`
- Example: IP = `RTC`, Mode = `Activate Clock Source`

5.6.2 Create platform settings for the bus driver API

The Topcomponent component interfaces with a bus driver API that can use either one of the SPI or I²C peripherals.

Click **Add Platform Setting** to switch to the platform settings creation view. Refer to [Figure 84](#).

Figure 84. Creating a new platform setting for an SPI bus driver

The screenshot shows the 'Platform Setting Details' window in STM32PackCreator. The left sidebar contains buttons for 'Add Parameter', 'Add Platform Setting', 'Add Custom Templates', 'Manage Conditions', and 'Preview STM32CubeMX / Edit Tabs'. The main area is divided into two sections: 'Platform Setting Details' and 'Filter'. The 'Platform Setting Details' section includes fields for 'Api' (BSP_BUS_DRIVER), 'Name' (PackA_platform1), and 'Group' (BSP). Below these is a table with two columns: 'STM32CubeMX Mode' and 'Condition of Assignment'. The table has two rows: 'File System TopComponent' with 'Always' and 'Data Exchange Bundle1' with 'Never'. The 'Filter' section includes an 'Add Filter' button, a 'Filter' header, and fields for 'Ip' (SPI), 'Mode' (Full-Duplex Master), and 'Help message' (SPI settings for TopComponent). A 'CubeMX template' checkbox is checked. The bottom of the window has 'Delete', 'Cancel', and 'Apply' buttons. Numbered callouts 1-8 are placed over the following elements: 1. Api dropdown, 2. Name text box, 3. Group dropdown, 4. File System TopComponent dropdown, 5. Ip dropdown, 6. Mode dropdown, 7. Help message text box, 8. Apply button.

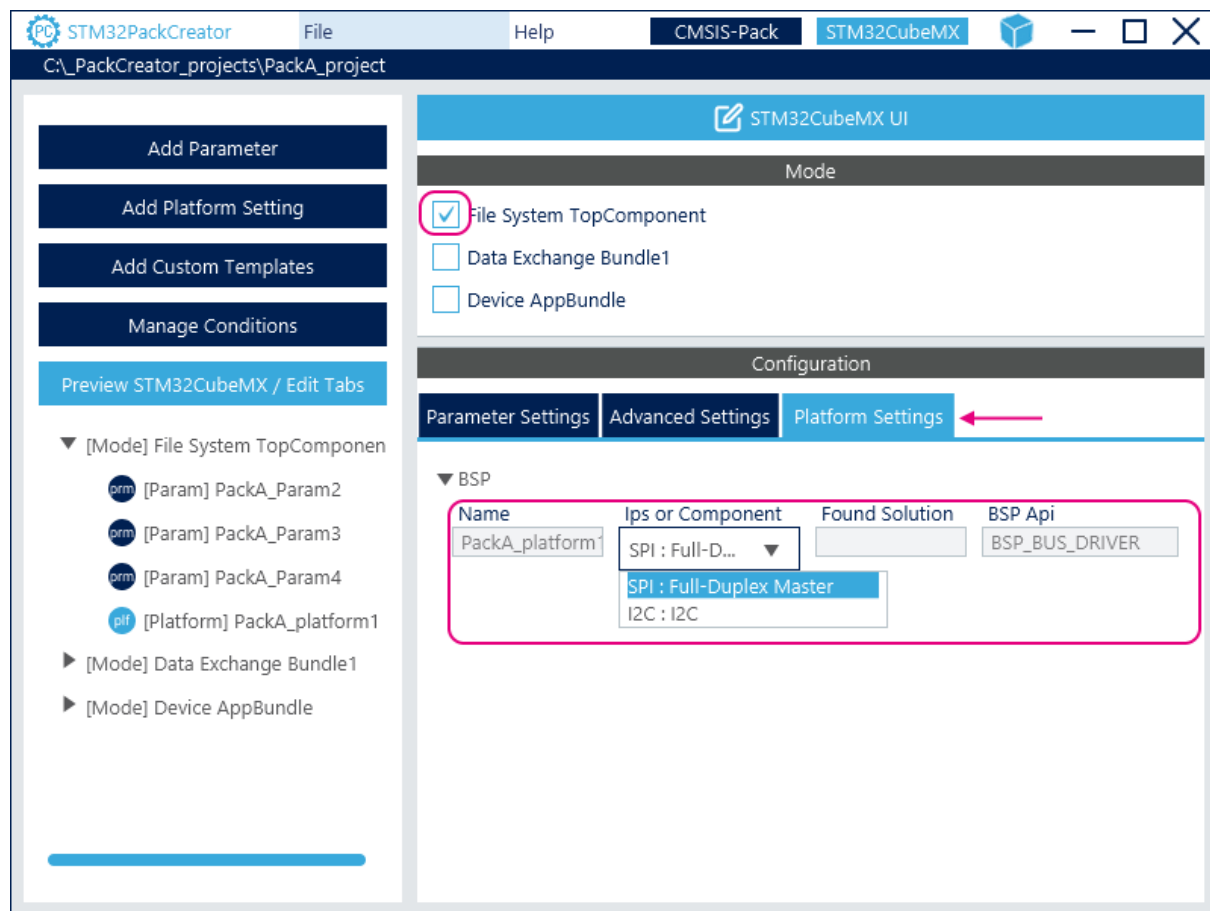
1. Enter the name (unique, no space) PackA_platform1.
2. Select the API BSP_BUS_DRIVER.
3. Select the BSP group.
4. Assign it to a Mode Always on TopComponent.
5. Under Filter, select the peripheral to use SPI.
6. Select the peripheral mode to use the Full-duplex master.
7. Fill in the help message.
8. Click Apply to validate the changes.

Then add the option to use I²C as a new filter for a bus driver platform setting:

1. Select Add Filter
2. Select the peripheral to use I²C
3. Select the peripheral mode to use I²C
4. Fill in the help message
5. Click Apply to validate the changes

Finally, check the outcomes using the STM32CubeMX preview. When the TopComponent is enabled, a new platform setting is available with the possibility to use SPI or I²C. Refer to [Figure 85](#).

Figure 85. Preview platform settings for SPI or I²C bus driver



5.6.3 Create platform settings for bus common drivers (LED and button)

The applications of AppBundle require a LED and a button to be configured.

Click **Add Platform Setting** to open the platform settings creation view and proceed as shown in Figure 86 to add a platform setting for the LED.

Figure 86. Creating platform settings for a LED

The screenshot shows the 'Platform Setting Details' dialog in the STM32PackCreator application. The dialog is titled 'Platform Setting Details' and has a sidebar on the left with buttons: 'Add Parameter', 'Add Platform Setting', 'Add Custom Templates', 'Manage Mode Conditions', and 'Preview STM32CubeMX / Edit Tabs'. The main area contains the following fields and controls:

- Api ***: A dropdown menu with 'BSP_COMMON_DRIVER' selected (marked with a red circle 1).
- Type ***: A dropdown menu with 'LED' selected (marked with a red circle 2).
- Name ***: A text field with 'BSP LED Blue' entered (marked with a red circle 3).
- Group ***: A dropdown menu with 'Application_BSP' selected (marked with a red circle 4).
- STM32CubeMX Mode**: A table with two columns: 'STM32CubeMX Mode' and 'Condition of Assignment'.

| STM32CubeMX Mode | Condition of Assignment |
|--------------------------|-------------------------------------|
| File System TopComponent | Never |
| Data Exchange Bundle1 | Never |
| Device AppBundle | Always (marked with a red circle 5) |
- Add Filter**: A button to add a filter.
- Filter**: A section with the following fields:
 - Ip ***: A dropdown menu with 'GPIO' selected (marked with a red circle 6).
 - Mode ***: A dropdown menu with 'Output' selected (marked with a red circle 7).
 - Help message ***: A text field with 'Blue LED' entered (marked with a red circle 8).
 - Templates**: Two radio buttons: 'default_conf.ftl' (selected) and 'Custom'.
- Buttons**: At the bottom right, there are four buttons: 'Delete', 'Clone', 'Cancel', and 'Apply' (marked with a red circle 9).

1. Select the API `BSP_COMMON_DRIVER`
2. Select the `LED` type
3. Complete the name `BSP LED Blue`
4. Create a new group `Application_BSP`
5. Assign to a Mode `Always` on AppBundle
6. Under IP, select `GPIO`
7. Under Mode, select `Output`
8. Fill in the Help message
9. Click `Apply` to validate the changes

Click Add Platform settings and proceed similarly to add the Button (Type=Button, IP=GPIO, Mode=EXTI) as shown in Figure 87.

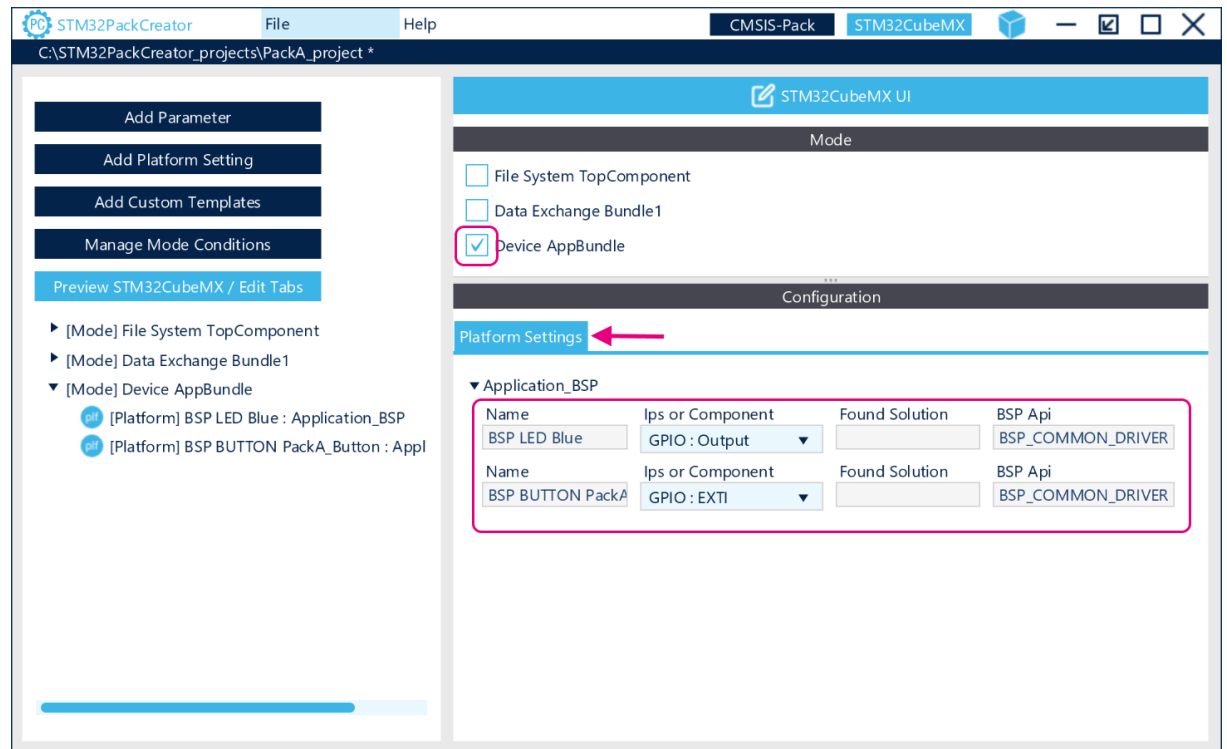
Figure 87. Creating platform settings for a button

The screenshot shows the STM32PackCreator application window with the 'Platform Setting Details' dialog open. The dialog is titled 'Platform Setting Details' and contains the following elements:

- Left Panel:** Contains buttons for 'Add Parameter', 'Add Platform Setting', 'Add Custom Templates', 'Manage Mode Conditions', and 'Preview STM32CubeMX / Edit Tabs'. Below these are expandable sections for '[Mode] File System TopComponent', '[Mode] Data Exchange Bundle1', and '[Mode] Device AppBundle'. The '[Mode] Device AppBundle' section is expanded, showing a sub-item '[Platform] BSP LED Blue : App'.
- Main Area:**
 - Api * (1):** A dropdown menu set to 'BSP_COMMON_DRIVER'.
 - Type * (2):** A dropdown menu set to 'BUTTON'.
 - Name * (3):** A text field containing 'BSP BUTTON PackA_Button'.
 - Group * (4):** A dropdown menu set to 'Application_BSP'.
 - STM32CubeMX Mode:** A section with three rows:
 - File System TopComponent:** A dropdown menu set to 'Never'.
 - Data Exchange Bundle1:** A dropdown menu set to 'Never'.
 - Device AppBundle (5):** A dropdown menu set to 'Always'.
 - Add Filter:** A button to add a new filter.
 - Filter:** A section with two rows:
 - Ip * (6):** A dropdown menu set to 'GPIO'.
 - Mode * (7):** A dropdown menu set to 'EXTI'.
 - Help message * (8):** A text field containing 'Application button'.
 - Templates:** A section with two radio buttons: 'default_conf.ftl' (selected) and 'Custom'.
 - Buttons:** At the bottom, there are buttons for 'Delete', 'Clone', 'Cancel', and 'Apply (9)'.

Finally, check the outcomes using the STM32CubeMX preview. When AppBundle is enabled, a new platform setting is available with the button and LED constraints as shown in Figure 88.

Figure 88. Preview of platform settings for LED and button



5.6.4 Create platform settings for unknown API

Bundle1 uses a reset line but only when the TopComponent is present.

Click **Add Platform Setting** to open the platform settings creation view and proceed as shown in Figure 89 to add a platform setting for the reset line.

Figure 89. Creating platform settings for a reset line custom API

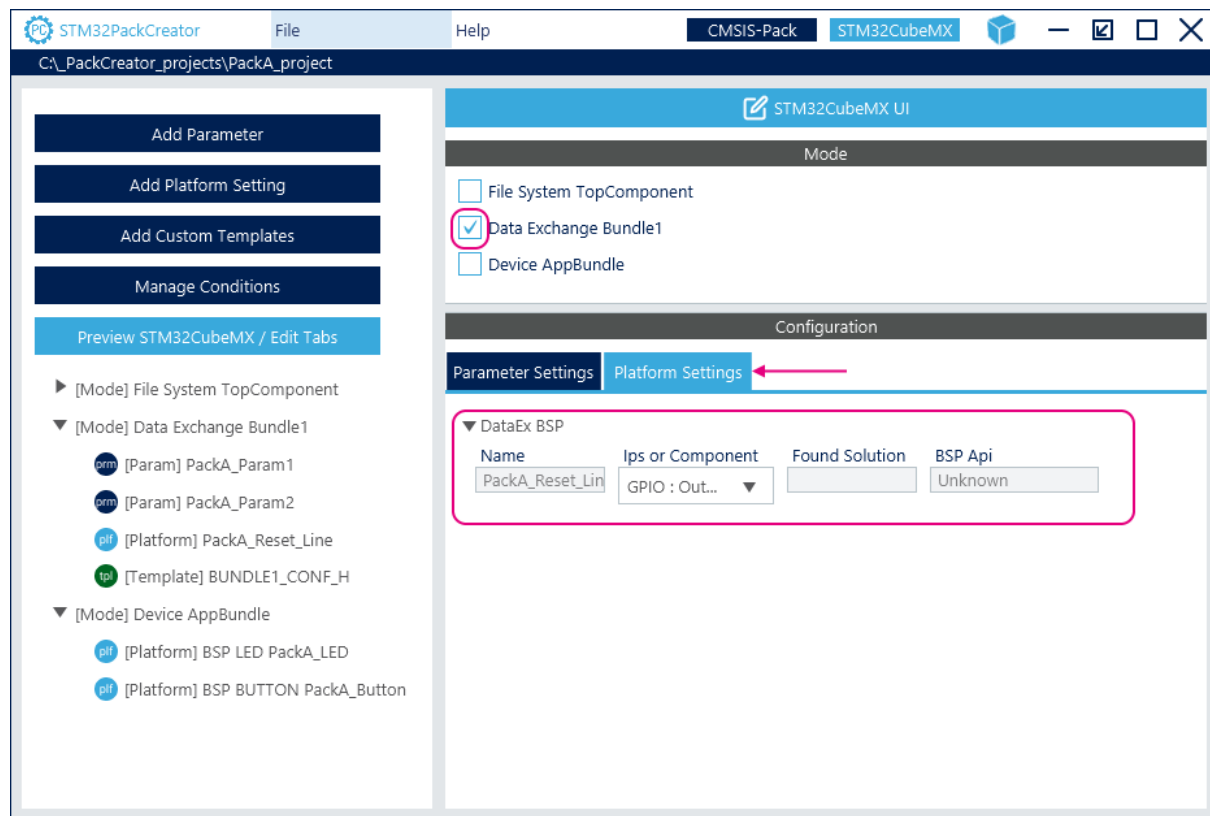
The screenshot shows the STM32PackCreator application window. On the left is a sidebar with navigation options: Add Parameter, Add Platform Setting, Add Custom Templates, Manage Mode Conditions, and Preview STM32CubeMX / Edit Tabs. The main area displays the 'Platform Setting Details' dialog. The dialog has the following elements:

- Api ***: A dropdown menu set to 'Unknown' (callout 1).
- Name ***: A text field containing 'PackA_Reset_Line' (callout 2).
- Group ***: A dropdown menu set to 'DataEx BSP' (callout 3).
- STM32CubeMX Mode**: A table with two columns: 'STM32CubeMX Mode' and 'Condition of Assignment'.
 - Row 1: 'File System TopComponent' with 'Never' in the condition column.
 - Row 2: 'Data Exchange Bundle1' with 'MyCondition2' in the condition column (callout 4).
 - Row 3: 'Device AppBundle' with 'Never' in the condition column.
- Add Filter**: A button to add a new filter.
- Filter**: A section with a table for GPIO and Output mode.
 - Ip ***: A dropdown menu set to 'GPIO' (callout 5).
 - Mode ***: A dropdown menu set to 'Output' (callout 6).
 - Help message ***: A text field containing 'GPIO for reset line' (callout 7).
 - Templates (1)**: A list of template files. The first item is 'bsp_for_api_Unknown_h.ftl' with a checked checkbox (callout 8). The second item is 'default_conf.ftl' with an unchecked checkbox.
- Buttons**: At the bottom right, there are buttons for 'Delete', 'Clone', 'Cancel', and 'Apply' (callout 9).

1. Select the API Unknown.
 2. Enter the name (unique, no space) PackA_Reset_Line.
 3. Create a new group DataEx_BSP.
 4. Assign to the Mode Bundle1.
 - Create a condition: MyCondition2 (TopComponent is present)
 5. Under Filter select GPIO and Output mode.
 6. Fill in the Help message
 7. Select bsp_for_api_Unknown_h.ftl as the template file to be used for code generation.
- Note:* Clicking Add Template adds other template files to be used.
8. Click Apply

Finally, check the outcomes using the STM32CubeMX preview. When Bundle1 is enabled, a new platform setting is available with the reset line constraint, as shown in Figure 90.

Figure 90. Preview of platform settings for a reset line custom API



5.7 Enhance the pack with custom templates

5.7.1 Custom templates introduction

By default, STM32PackCreator generates packs with a default template inside. This allows STM32CubeMX to generate a header file with all pack parameter defines: <Vendor name>.<Pack name>_conf.h.

STM32PackCreator allows as well pack developers to specify their templates for a given mode of the pack and for a selection of parameters: these custom templates are optional and necessary only when custom C code must be generated using STM32CubeMX.

Freemarker is the language used to write Freemarker templates. For details check out: <https://freemarker.apache.org/>. The pack developers can write custom templates, and add them to their packs using STM32PackCreator and check the generated code using STM32CubeMX.

Note: STM32CubeMX adds some design constraints for custom code to be generated. Parameters are accessible as SWIPdatas, as shown in Figure 91, and platform settings as BSPName, as shown in Figure 93.

Figure 91. Freemarker template example for generating parameters defines

```
/* Define to prevent recursive inclusion -----*/
#ifndef BUNDLE1_CONF_H
#define BUNDLE1_CONF_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes -----*/
[#if includes??]
[#list includes as include]
#include "${include}"
[/#list]
[/#if]

#include "${FamilyName?lower_case}xx_hal.h"
#include <string.h>

[#compress]
[#list SWIPdatas as SWIP]
[#if SWIP.defines??]
[#list SWIP.defines as definition]
/*----- [#if definition.comments??]${definition.comments} [/#if] -----*/
#define ${definition.name} #t#t ${definition.value}
[#if definition.description??]${definition.description} [/#if]
[/#list]
[/#list]
```

Figure 92. Generated C code based on the template for parameter defines

```

/* Define to prevent recursive inclusion -----*/
#ifndef BUNDLE1_CONF_H
#define BUNDLE1_CONF_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes -----*/
#include "stm32f4xx_hal.h"
#include <string.h>

/*----- Bundle1 configuration param2 -----*/
#define PackA_Param2 10

/*----- Bundle1 configuration param1 -----*/
#define PackA_Param1 enabled

```

Figure 93. Freemarker template example for generating BSP code (platform settings)

```

...
#ifndef __API_UNKNOWN_H
#define __API_UNKNOWN_H

#define ${bspName}_PIN                ${GPIO_PIN}
#define ${bspName}_PORT                ${GPIO_PORT}
#define ${bspName}_GPIO_CLK_ENABLE()  __HAL_RCC_${GPIO_PORT}_CLK_ENABLE()
#define ${bspName}_GPIO_CLK_DISABLE() __HAL_RCC_${GPIO_PORT}_CLK_DISABLE()

#define ${bspName}_EXTI_LINE            EXTI_LINE_${ExtiLine}

#endif
...

```

Figure 94. Generated C code based on the template for BSP code (platform settings)

```

#ifndef __API_UNKNOWN_H
#define __API_UNKNOWN_H

#define LED_GREEN_PIN                GPIO_PIN_5
#define LED_GREEN_PORT                GPIOA
#define LED_GREEN_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOA_CLK_ENABLE()
#define LED_GREEN_GPIO_CLK_DISABLE() __HAL_RCC_GPIOA_CLK_DISABLE()

#define LED_GREEN_EXTI_LINE           EXTI_LINE_5

#endif

```

5.7.2 Add a custom template to generate C files or code snippets

Bundle1 requires a custom file to be generated.

Click *Add Custom templates* and proceed as shown in Figure 95 to specify a custom template and a set of parameters that are accessible to the template.

Figure 95. Adding a custom template

Custom template Details

Mode * 1 Data Exchange Bundle1

Template file * 2 bundle1_conf_h.ftl Browse

Category * 3 header

Template name * 4 BUNDLE1_CONF_H

Description * 5 Header for bundle1 configuration #define

Condition 6 Always

Generated file name * 7 bundle1_conf .h

Generated file destination* 8

- ☒ App folder (Application code)
- ☐ Target folder (Board specific code)
- ☐ Generated file included in other generated file

| Parameters | Comments |
|--------------------------------------------------|------------------------------|
| <input checked="" type="checkbox"/> PackA_Param1 | Bundle1 configuration param1 |
| <input checked="" type="checkbox"/> PackA_Param2 | Bundle1 configuration param2 |

Parameters added to template 9

Delete Cancel Apply 10

1. Select the Mode `Bundle1`
2. Browse to select a template `.ftl` file from the file system `bundle1_conf_h.ftl`
3. Specify a category `header`
4. Specify a template name `BUNDLE1_CONF_H`
5. Enter a description `header file for Bundle1 defines`
6. Keep the condition to `Always`
7. Specify the name of the generated file `bundle1_conf (.h)`
8. Specify where the file may be generated select the `App folder`.
9. Select the parameters the generated code uses
10. Click `Apply`

About generating code snippets

The generated file destination can be set to generate code to be included in other generated files. This option is available only to template files with the extension `.tmp`. This generates a code snippet that is included in a file generated through another template, which must have an included statement to ensure the code snippet is imported at code generation.

5.8 Enhance the pack with application templates

5.8.1 Application components and application templates introduction

Application components are optional

- A pack may come with one or more Application components.
- Application components can be top components or be part of a bundle.
- Application components come with a module name specified in the CMSIS-Pack view.
- Only one application can be enabled per project.

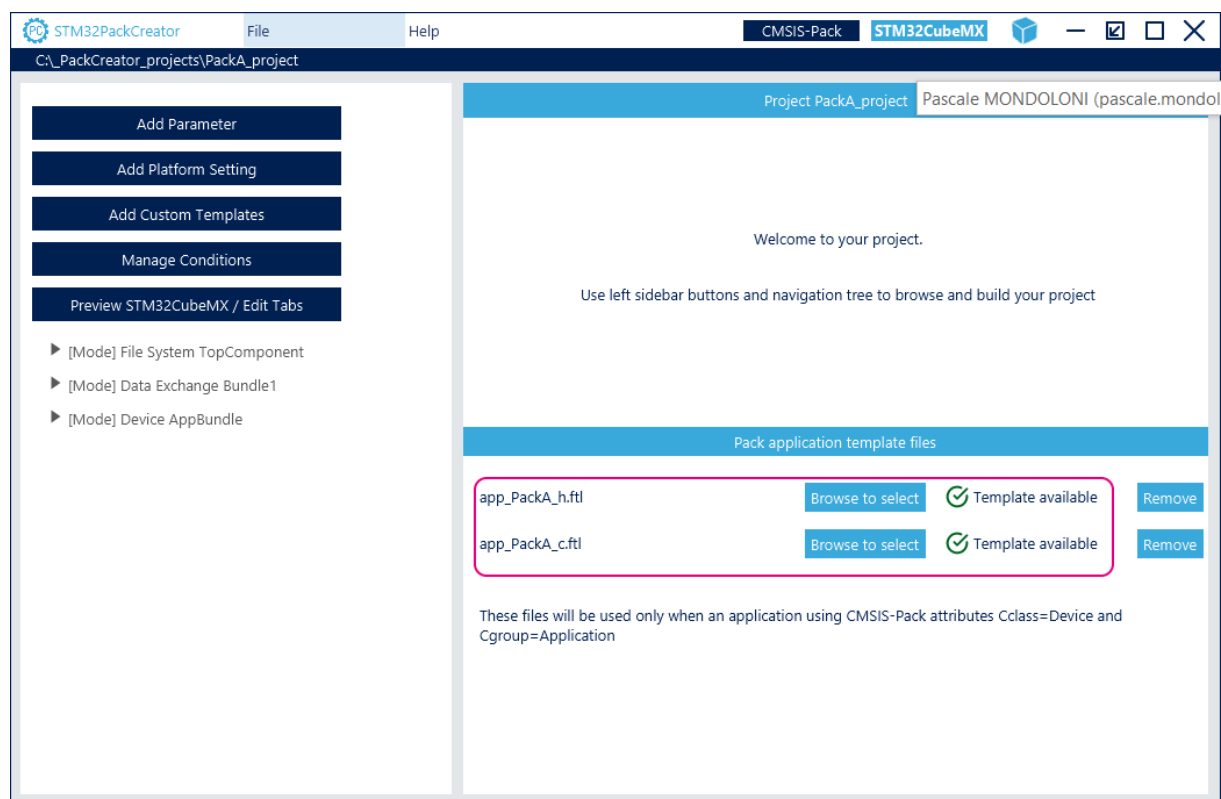
The main purpose of application components is for STM32CubeMX to generate application-ready code

- `app_<modulename>_Init/Process` functions are called by default in `main.c`. This generation can be disabled through STM32CubeMX Project Manager > Advanced settings.
- `app_<modulename>_Init/Process` functions are defined in `app_packname.c/.h` files.

There are several options in STM32PackCreator to specify application templates:

- Specify application templates common to all application components of the pack. From the STM32CubeMX landing page, select `app_<pack name>_h.ftl` and `app_<pack name>_c.ftl` files, as shown in Figure 96. Templates may use if statements to generate code relevant to a given application variant.
- Specify a common `app_<pack name>_h.ftl` from the STM32CubeMX landing page and for `.c` file generation, add a custom template to each application component in the pack.
- Do not specify any common pack application template. Add custom templates to each application component in the pack.

Figure 96. Pack application templates



- STM32CubeMX generates the corresponding `app_<modulename>_h/.c` files. They are generated as empty files if the templates are not available in the pack.

Applications RTE Components or Module names

The custom templates of the application can be designed so that code is generated differently according to the RTE component definitions or module names.

- Use RTE Components defines:
 - RTE Components are specified in the components section of the CMSIS-Pack view.
 - Most useful when all applications share the same module name, which is recommended to support multi-pack configurations. Use the following Freemarker statement: `#if define? contains("<RTE component>")`.
- Use Module names:
 - Module names are specified in the components section of the CMSIS-Pack view.
 - When applications use different module names and a common template, use the following Freemarker statement: `#if ModuleName?contains("<module name>")`

Refer to the X-Cube-BLE2 pack for examples of application templates `app_x-cube-ble2_c/_h.ftl`.

5.9 Save the project and generate the pack

1. Go back to the CMSIS-Pack view and create a new 0.0.2 release.
2. Generate the new pack version using **File > Save & Generate pack**, as shown in Figure 97.

Figure 97. Pack A revision 0.0.2

The screenshot shows the STM32PackCreator application window. The title bar includes the STM32PackCreator logo, 'File', and 'Help' menus. The main window displays the 'Pack General Attributes' section, which is expanded to show 'Current Release*'. The 'Version*' field is set to '0.0.2' with a 'New' button next to it. The 'Description*' field contains the text 'Pack revision enhanced for STM32CubeMX'. Below this, there are fields for 'Release Date*' (set to '2020-06-12'), 'Release Tag', 'Release URL', 'Deprecated Date' (set to 'yyyy-mm-dd'), and 'Replacement'. At the bottom of this section are 'Delete', 'Edit', 'Cancel', and 'Apply' buttons. Below the 'Current Release' section, there are three expandable sections: 'Previous Releases (1)', 'Pack Details*', and 'Additional Files (2)'. The application is running on a Windows operating system, as indicated by the taskbar at the bottom.

5.10 Use the pack in STM32CubeMX

The pack is ready to be installed, enabled in an STM32CubeMX project, configured, and used to generate the C project.

5.10.1 Install the pack

Launch STM32CubeMX and start a new project from a NUCLEO-F401RE board.

From the pinout view, select **Software Packs > Manage Software Packs** and install the pack new revision.

5.10.2 Select the components

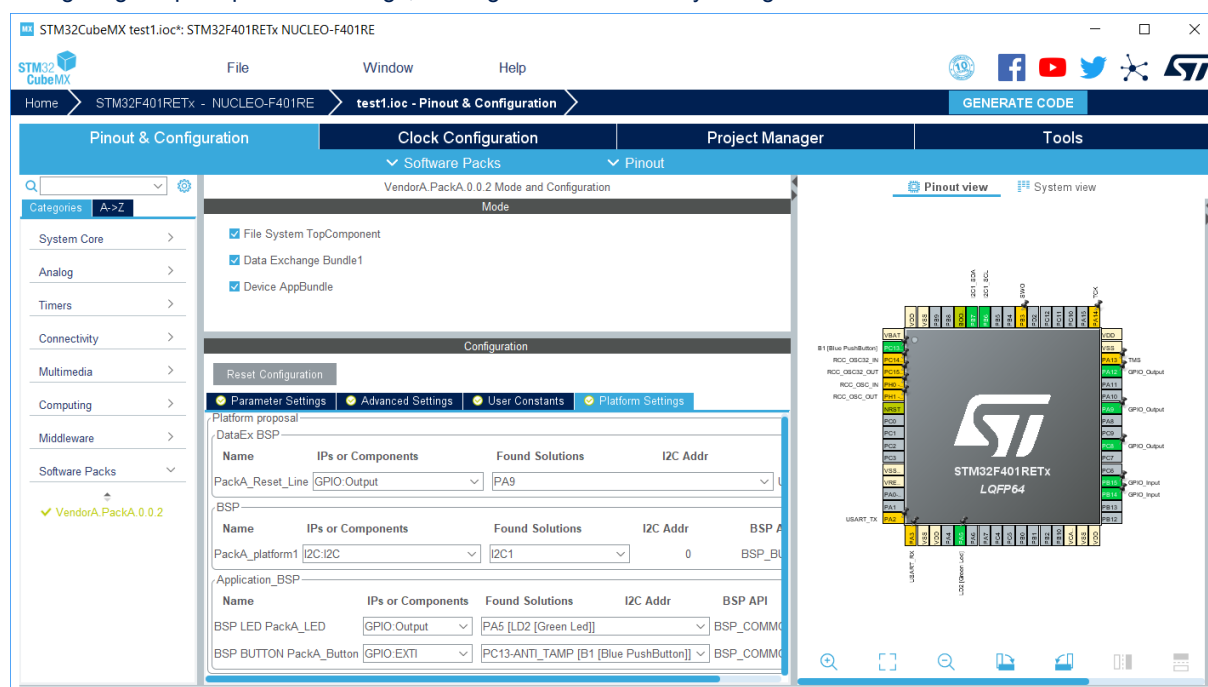
Select the pack components using **Software Packs > Select Components** menu and click **OK** to close the component selector window.

5.10.3 Configure the pack for the project

Back to the pinout view, click **Software packs** on the left panel, and click on the **packA** entry.

From the **Mode** panel, enable all the pack modes in the mode panel.

From the configuration panel, configure the parameters from the parameter settings tabs. Then, configure the platform settings: enable an I2C in I²C mode, configure a pin as **GPIO Output** from the pinout view, and finish configuring the pack platform settings, reusing the GPIOs already configured as LED and user button.



5.10.4 Generate the code and check the generated project

Click generate code and check the files are well generated according to the provided templates:

1. In **Core/Src**:
 - a. The **main.c** file with calls to **Init** and **Process** functions
 - b. The **stm32f4xx_nucleo_bus.c** bus driver source file
2. In **Core/Inc**:
 - a. bus driver headers and configuration files **stm32f4xx_nucleo_bus.h**, **stm32f4xx_nucleo_conf.h**.
3. In **Drivers/BSP/NUCLEO-F401RE**:
 - a. BSP common driver source and header files **stm32f4xx_nucleo.c**, **stm32f4xx_nucleo.h**, and **stm32f4xx_nucleo_errno.h**

4. In the PackA folder VendorA.PackA_conf.h
5. in PackA/App folder:
 - a. The application header and source files app_packa.h and app_packa.c
 - b. Bundle1 configuration file bundle1_conf.h
6. In Pack/Target folder:
 - a. The header file for the code that is target-dependent (BSP code for the unknown API) bsp_for_api_Unknown.h. To generate the .c file, add a second custom template.
7. In Middlewares\Third_Party the pack component files

6 Creating a pack from an existing pack

STM32PackCreator can create new projects starting from an existing pack. Select `Start new project from pack` from the STM32PackCreator home page.

Reminder: it is not possible to create an STM32Cube Expansion Pack starting a project from a pack that is not an STM32Cube Expansion Pack.

7 Cloning a project

STM32PackCreator can save the current project as a new project: Select `File> Clone project as` and enter the new project folder name.

8 Opening a project

Opened recent projects shortcut is available from the STM32PackCreator home page.
To open other projects, select **File > Open project** and specify the project folder name.

9 How to generate an example description file

9.1 Purpose

The aim of this section is to explain how to update the release notes and the readme files to generate the example description file of I-Cubes.

Release notes and readme are mandatory for the correct integration of I-Cube examples in the STM32CubeMX example browser.

This document describes needed steps and tips to update the necessary documents before generating the example description file.

The example PDSC file is generated by the *Generate Examples Description File* functionality integrated in STM32PackCreator tool.

9.2 I-Cube integration flow in STM32CubeMX/STM32CubeIDE

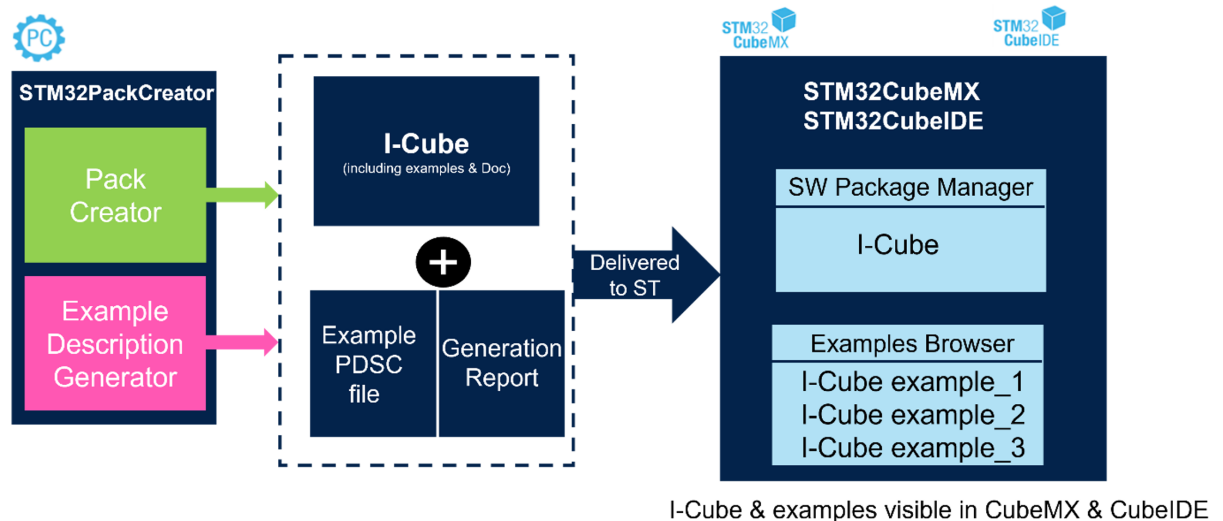
To enhance the visibility of his software, the partner can benefit from the ST ecosystem by making the pack examples visible in the STM32CubeMX/STM32CubeIDE example browser. Another major benefit of pack integration in ST tools is simplifying the user journey by providing a ready-to-use project assembling all needed software components.

To enable the pack integration ST expects in addition to the pack itself a couple of documents from the pack provider:

- The examples description file.
- The PDSC generation report.

Here is a flow chart summarizing the I-Cube life cycle:

Figure 98. I-Cube life cycle



Note that:

- Example description file and the generation report should not be integrated into the pack (but delivered alongside the I-Cube pack).
- I-Cube packs must not be delivered if any error remains during the examples description generation.

9.3 Release notes template

The release notes must follow the template *EmbSW_release_notes_template.html* and the document structure must be kept as it is.

The user can edit the template html file with any text editor and change it according to the package information. The template file contains all the needed information to correctly create the user's release note highlighted in red boxes: ① Guidance item.

The mandatory sections for the *Examples description generator* are the *SW package name*, the *Purpose* and the *Update history*.

ST recommends keeping all sections headers even if they are empty to keep harmonization between the different packages.

Figure 99. Release notes template

The screenshot displays the *EmbSW_release_notes_template.html* file with several sections highlighted in red boxes to indicate mandatory or guidance items:

- Guidance Item:** A box at the top left containing instructions on how to use the template, including the preamble and the form.
- Update history:** A box at the top right containing a table for recording updates, with columns for Version and Date.
- Main changes:** A box in the middle right containing a section for describing the changes, with a table for recording them.
- Additional features:** A box in the middle right containing a section for describing additional features, with a table for recording them.
- Fixed bugs list:** A box in the middle right containing a section for describing fixed bugs, with a table for recording them.
- Purpose:** A box at the bottom left containing a section for describing the purpose of the release.
- Release Notes for <SW name>:** A box in the middle left containing the title and copyright information for the release notes.

- The package must contain three release notes files: the first one in the root package, the second one in the project folder, and the third one in the `_htmresc` folder.
- Only the release notes under the root of the package are mandatory for the *Examples description generator* to correctly handle the I-Cube examples.

9.4 README template

To correctly extract the examples, the *Examples description generator* needs also a README.md file, it must be aligned with ST templates: README_repo_template.md.

Every example project folder must contain its own README file.

This template is provided as a guideline, the user needs to update it according to his component. The user can edit it through a text editor to add or remove parts except for the mandatory ones which are listed below:

- The description headline
- Keywords
- Hardware and software environment to mention the supported device

Be careful to respect the syntax of the template otherwise, the *Examples description generator* cannot extract needed information. For example, to separate the sections, it is necessary to use ####, and every headline must use __Headline__.

Figure 100. README example

```
## __PPP_Example | Application | Demonstration-i-Name> <Example | Application | Demonstration> Description__

< Add here the example description, need to provide at minimum the following information:
- highlight the main features
- how to use step by step
- list any precondition to run the example and/or any known limitations >
Guidance (to be removed): when higher indention is needed please use #### or ##### etc...

### __Keywords__

<Add here significant keywords of the example for CubeMX Examples browser utility >

### __Directory contents__

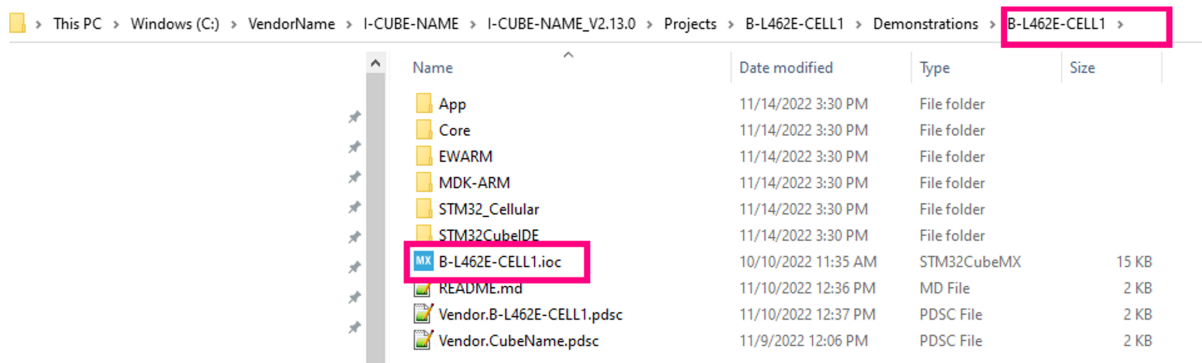
< Add here the files list with brief description >
```

9.5 How to generate the example description file

Before start building the example description file, be careful to have the correct naming for the folder structure, meaning:

- The name of folder containing the package must be <I-Cube-name>_VX.Y.Z>, for example I-CUBE-NAME_V2.13.0.
- The name of the example folder must be the same as the ioc name, for example, if the ioc name is B-L462E-CELL1, the example project folder must be Demonstrations\B-L462E-CELL1.

Figure 101. Folder structure



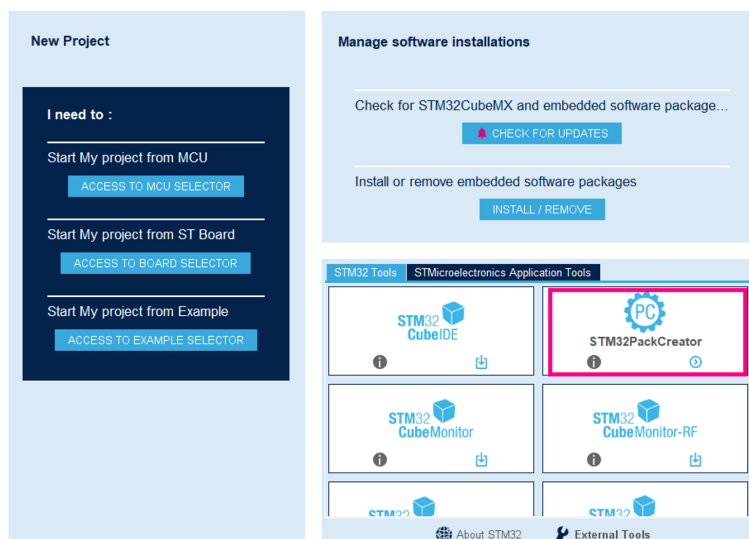
After correctly updating the documents, the user can now generate the example description file following the next three steps.

9.5.1 Launch STM32PackCreator

There are two ways to launch STM32PackCreator:

- From STM32CubeMX install folder: STM32CubeMX\utilities\STM32PackCreator
- Directly from STM32CubeMX home page

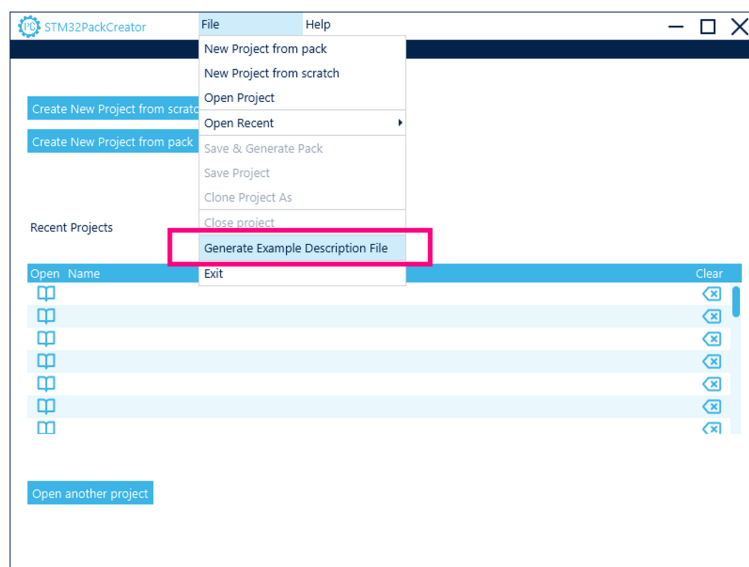
Figure 102. Launch STM32PackCreator



9.5.2 Examples description generator tab

Go to *File* then select *Generate Example Description File*.

Figure 103. Selecting *Generate Example Description File*



- The user must select the package in *Pack Path*.
- The *Output Folder* and *Output PDSC File Name* fields are filled automatically. If the user wants to change them, he can uncheck the *Auto* combo box.

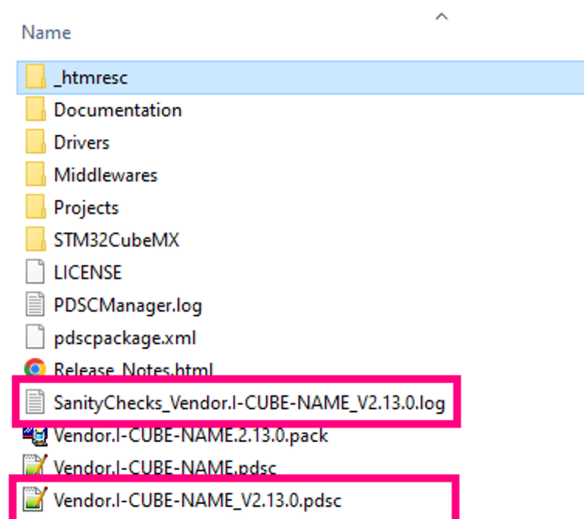
Figure 104. Generate Example Description File options

- The *Middlewares List File* is an optional field. If the pack includes a component that needs to be identified as middleware in example browser left panel, the user must add it in this field.
- The tool generates a short report containing the number of examples and the total warning and errors. The generation is considered successful only if there are no errors in the log file.

9.5.3 Generate example description file

The description file and the full log are generated under the root pack folder.

Figure 105. Generated example description file



Revision history

Table 7. Document revision history

| Date | Revision | Changes |
|-------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12-Jul-2020 | 1 | Initial release. |
| 2-Nov-2020 | 2 | <p>Deleted compatibility limitation with CMSIS-pack standard in <i>Section 4.3 Rules and limitations</i></p> <p>Updated:</p> <ul style="list-style-type: none"> JRE™ requirements in <i>Section 4.4 System requirements</i> Program launch in <i>Section 4.5.1 From STM32CubeMX user interface</i> and <i>Section 5.2 Create a new project from scratch</i> <i>Figure 5, Figure 7, Figure 8, Figure 9, Figure 15, Figure 19, and Table 3</i> handling example in pack details <p>Added <i>Section 5.3.15 Specify the example projects to be delivered with the pack</i></p> |
| 11-Feb-2021 | 3 | <p>Updated:</p> <ul style="list-style-type: none"> JRE™ requirements in <i>Section 4.4 System requirements</i> Command lines in <i>Section 4.5.2 Standalone option</i> <i>Table 4. STM32CubeMX view - Main features</i> <i>Section 5.4.3 Select pack components, check and solve dependencies</i> <i>Section 5.4.5 Generate the project</i> <i>Figure 13, Figure 14, Figure 54 to Figure 56, Figure 61 to Figure 63, Figure 70, Figure 72, Figure 74, Figure 76, and Figure 86 to Figure 89</i> <p>Removed:</p> <ul style="list-style-type: none"> Former <i>Figure 63</i> |
| 03-Jul-2023 | 4 | Added Section 9 How to generate an example description file |

Contents

| | | |
|----------|----------------------------------------------------------------------------------|-----------|
| 1 | References | 2 |
| 2 | STM32Cube overview | 3 |
| 3 | Software pack overview | 4 |
| 3.1 | Definition and CMSIS-Pack standard | 4 |
| 3.2 | STM32Cube Expansion Packages | 5 |
| 3.3 | Software pack creation cycle | 5 |
| 4 | STM32PackCreator overview | 6 |
| 4.1 | Principles | 6 |
| 4.2 | Key features | 6 |
| 4.3 | Rules and limitations | 6 |
| 4.4 | System requirements | 9 |
| 4.5 | Launching STM32PackCreator | 10 |
| 4.5.1 | From STM32CubeMX user interface | 10 |
| 4.5.2 | Standalone option | 10 |
| 4.6 | Main menus | 11 |
| 4.7 | Main views | 12 |
| 4.7.1 | CMSIS-Pack view | 12 |
| 4.7.2 | STM32CubeMX view | 14 |
| 5 | Creating a pack from scratch (step-by-step procedure) | 16 |
| 5.1 | Prepare for pack creation | 16 |
| 5.1.1 | General approach | 16 |
| 5.1.2 | Demonstration pack overview | 17 |
| 5.2 | Create a new project from scratch | 18 |
| 5.3 | Describe the pack using the CMSIS-Pack view | 21 |
| 5.3.1 | General information and tips | 21 |
| 5.3.2 | Fill in the pack general attributes | 22 |
| 5.3.3 | Fill in the release information | 25 |
| 5.3.4 | Create a bundle with two components | 26 |
| 5.3.5 | Create the top component | 38 |
| 5.3.6 | Create the bundle with application components (specify variant and module names) | 41 |
| 5.3.7 | CMSIS-Pack conditions overview and STM32CubeMX restrictions | 43 |
| 5.3.8 | List of pack constraints | 44 |
| 5.3.9 | Create condition on Dcore | 45 |
| 5.3.10 | Assign Cortex-M device condition on Bundle1 components | 48 |
| 5.3.11 | Create conditions on Tcompiler | 50 |

| | | |
|--------|--------------------------------------------------------------------------------------|------------|
| 5.3.12 | Assign CM4_Keil, CM7_Keil, CM4_GCC, and CM7_GCC conditions to component files . | 53 |
| 5.3.13 | Create conditions on components | 56 |
| 5.3.14 | Assign conditions on application components | 58 |
| 5.3.15 | Specify the example projects to be delivered with the pack. | 60 |
| 5.4 | Save the software pack and use it in STM32CubeMX | 64 |
| 5.4.1 | Generate and install the pack | 64 |
| 5.4.2 | Create a project and check if the pack is available | 65 |
| 5.4.3 | Select pack components, check and solve dependencies. | 67 |
| 5.4.4 | Check the pack in the STM32CubeMX pinout configuration view | 69 |
| 5.4.5 | Generate the project | 69 |
| 5.5 | Enhance the pack with configuration parameters | 71 |
| 5.5.1 | Introduction | 71 |
| 5.5.2 | Switch to STM32CubeMx view | 73 |
| 5.5.3 | Add the first parameter for Bundle1. | 74 |
| 5.5.4 | Add a second parameter for Bundle1 and TopComponent | 76 |
| 5.5.5 | Add the third parameter for TopComponent in the user-defined tab and group | 78 |
| 5.5.6 | Create a condition and assign it to a parameter | 80 |
| 5.5.7 | Create a fourth conditional parameter | 81 |
| 5.6 | Enhance the pack with platform settings | 82 |
| 5.6.1 | Platform settings introduction | 82 |
| 5.6.2 | Create platform settings for the bus driver API | 87 |
| 5.6.3 | Create platform settings for bus common drivers (LED and button) | 89 |
| 5.6.4 | Create platform settings for unknown API | 92 |
| 5.7 | Enhance the pack with custom templates | 94 |
| 5.7.1 | Custom templates introduction | 94 |
| 5.7.2 | Add a customs template to generate C files or code snippets | 97 |
| 5.8 | Enhance the pack with application templates | 98 |
| 5.8.1 | Application components and application templates introduction | 98 |
| 5.9 | Save the project and generate the pack. | 99 |
| 5.10 | Use the pack in STM32CubeMX. | 100 |
| 5.10.1 | Install the pack | 100 |
| 5.10.2 | Select the components | 100 |
| 5.10.3 | Configure the pack for the project | 100 |
| 5.10.4 | Generate the code and check the generated project. | 100 |
| 6 | Creating a pack from an existing pack | 102 |
| 7 | Cloning a project | 103 |
| 8 | Opening a project. | 104 |

| | | |
|----------|-----------------------------------------------------------|------------|
| 9 | How to generate an example description file | 105 |
| 9.1 | Purpose | 105 |
| 9.2 | I-Cube integration flow in STM32CubeMX/STM32CubeIDE | 105 |
| 9.3 | Release notes template | 106 |
| 9.4 | README template | 107 |
| 9.5 | How to generate the example description file | 107 |
| 9.5.1 | Launch STM32PackCreator | 108 |
| 9.5.2 | Examples description generator tab | 108 |
| 9.5.3 | Generate example description file | 109 |
| | Revision history | 110 |
| | List of tables | 114 |
| | List of figures | 115 |

List of tables

| | | |
|-----------------|----------------------------------------------------------------|-----|
| Table 1. | File menu | 11 |
| Table 2. | Help menu | 11 |
| Table 3. | CMSIS-Pack standard elements | 13 |
| Table 4. | STM32CubeMX view - Main features | 15 |
| Table 5. | CMSIS-Pack condition types and support in STM32CubeMX. | 43 |
| Table 6. | Condition rules | 43 |
| Table 7. | Document revision history | 110 |

List of figures

| | | |
|-------------------|--------------------------------------------------------------------------|----|
| Figure 1. | Keil® tutorial cover page | 4 |
| Figure 2. | Creation cycle for software pack enhanced for STM32CubeMX | 5 |
| Figure 3. | CMSIS-Pack standard compatibility | 7 |
| Figure 4. | Activation of STM32Cube compliant option | 8 |
| Figure 5. | Access the Tools view | 10 |
| Figure 6. | Launch STM32PackCreator | 11 |
| Figure 7. | Project with STM32CubeExpansion rules enabled | 12 |
| Figure 8. | CMSIS-Pack view | 12 |
| Figure 9. | STM32CubeMX view - Example of pack enhanced for STM32CubeMX | 14 |
| Figure 10. | Demonstration pack overview | 17 |
| Figure 11. | Input folders and files used to produce the demonstration pack | 18 |
| Figure 12. | Create a new project from scratch | 18 |
| Figure 13. | New project from scratch 1/2 | 19 |
| Figure 14. | New project from scratch 2/2 | 20 |
| Figure 15. | Project created and opened in CMSIS-Pack view | 21 |
| Figure 16. | Selecting the license file | 22 |
| Figure 17. | Selecting the pack folder for the license file | 23 |
| Figure 18. | Project with pack license file attribute specified | 24 |
| Figure 19. | Project with pack release attribute specified | 25 |
| Figure 20. | Adding a bundle | 26 |
| Figure 21. | Filling bundle details | 27 |
| Figure 22. | Specifying pack folder for bundle documentation | 28 |
| Figure 23. | Creating component1 for Bundle1 | 29 |
| Figure 24. | Creating component2 for Bundle1 | 30 |
| Figure 25. | Pack specified with one bundle and two components | 31 |
| Figure 26. | Bundle's components tab view | 32 |
| Figure 27. | Selecting the pack folder for the component1 files | 33 |
| Figure 28. | Component1 with component files specified | 34 |
| Figure 29. | Selecting the pack folder for the component2 files | 35 |
| Figure 30. | Component2 with component files specified | 36 |
| Figure 31. | Creating a top component | 38 |
| Figure 32. | Adding top component files | 39 |
| Figure 33. | Project with one bundle and one top component | 40 |
| Figure 34. | Application bundle creation | 41 |
| Figure 35. | First application component | 42 |
| Figure 36. | Second application component | 42 |
| Figure 37. | Creating a condition to accept Cortex-M4-based devices | 45 |
| Figure 38. | Updating a condition to accept Cortex-M7-based devices | 46 |
| Figure 39. | Cortex®-M device condition | 47 |
| Figure 40. | Assigning condition to component 1 | 48 |
| Figure 41. | Condition assigned to component 2 | 49 |
| Figure 42. | CM4_Keil condition | 50 |
| Figure 43. | Cloning conditions | 51 |
| Figure 44. | CM7_Keil condition | 51 |
| Figure 45. | Conditions on GCC compiler | 53 |
| Figure 46. | Selecting component files to edit file attributes | 54 |
| Figure 47. | Assigning CM4_GCC condition on file | 54 |
| Figure 48. | Assigning CM7_Keil condition on file | 55 |
| Figure 49. | Creating a condition for Bundle 1 | 56 |
| Figure 50. | Creating a condition for Bundle 1 and TopComponent | 57 |
| Figure 51. | Assigning conditions to the DataEx_App component | 58 |
| Figure 52. | Assigning conditions to the DataExFS_App component | 59 |
| Figure 53. | Adding an example project to the pack | 60 |

| | | |
|-------------|------------------------------------------------------------------------------|-----|
| Figure 54. | Specifying example details – part 1 | 61 |
| Figure 55. | Specifying example details – part 2 | 62 |
| Figure 56. | New example added to the pack | 63 |
| Figure 57. | Generating a CMSIS-Pack pack with STM32PackCreator | 64 |
| Figure 58. | Installing the generated pack in STM32CubeMX | 65 |
| Figure 59. | Selecting pack components | 65 |
| Figure 60. | Pack unavailability when the Dcore condition is not met | 66 |
| Figure 61. | Pack availability when Dcore condition is met | 66 |
| Figure 62. | Highlighting dependencies on components | 67 |
| Figure 63. | Solving one dependency on components | 68 |
| Figure 64. | Unconfigurable software pack in STM32CubeMX user interface | 69 |
| Figure 65. | Code generation settings in STM32CubeMX | 69 |
| Figure 66. | Project settings in STM32CubeMX | 70 |
| Figure 67. | PackA outline | 71 |
| Figure 68. | PackA not yet enhanced for STM32CubeMX | 72 |
| Figure 69. | STM32CubeMX preview in STM32PackCreator | 73 |
| Figure 70. | Adding the first parameter to Bundle1 | 74 |
| Figure 71. | Preview the first parameter in STM32CubeMX UI | 75 |
| Figure 72. | Adding the second parameter to Bundle1 and TopComponent | 76 |
| Figure 73. | Preview the second parameter in STM32CubeMX UI | 77 |
| Figure 74. | Adding the third parameter to TopComponent in the user-defined Tab and group | 78 |
| Figure 75. | Preview the third parameter in STM32CubeMX UI | 79 |
| Figure 76. | Creating parameters with conditions | 81 |
| Figure 77. | Creating a parameter list | 82 |
| Figure 78. | STM32CubeMX Freemarker templates for platform settings | 82 |
| Figure 79. | STM32CubeMX generated BSP common driver files | 83 |
| Figure 80. | STM32CubeMX generated Bus I/O driver files | 83 |
| Figure 81. | Platform settings for Bus I/O API | 84 |
| Figure 82. | Platform settings for BSP common API | 85 |
| Figure 83. | Platform settings with custom BSP API | 86 |
| Figure 84. | Creating a new platform setting for an SPI bus driver | 87 |
| Figure 85. | Preview platform settings for SPI or I ² C bus driver | 88 |
| Figure 86. | Creating platform settings for a LED | 89 |
| Figure 87. | Creating platform settings for a button | 90 |
| Figure 88. | Preview of platform settings for LED and button | 91 |
| Figure 89. | Creating platform settings for a reset line custom API | 92 |
| Figure 90. | Preview of platform settings for a reset line custom API | 93 |
| Figure 91. | Freemarker template example for generating parameters defines | 94 |
| Figure 92. | Generated C code based on the template for parameter defines | 95 |
| Figure 93. | Freemarker template example for generating BSP code (platform settings) | 95 |
| Figure 94. | Generated C code based on the template for BSP code (platform settings) | 96 |
| Figure 95. | Adding a custom template | 97 |
| Figure 96. | Pack application templates | 98 |
| Figure 97. | Pack A revision 0.0.2 | 99 |
| Figure 98. | I-Cube life cycle | 105 |
| Figure 99. | Release notes template | 106 |
| Figure 100. | README example | 107 |
| Figure 101. | Folder structure | 107 |
| Figure 102. | Launch STM32PackCreator | 108 |
| Figure 103. | Selecting <i>Generate Example Description File</i> | 108 |
| Figure 104. | <i>Generate Example Description File</i> options | 109 |
| Figure 105. | Generated example description file | 109 |

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2023 STMicroelectronics – All rights reserved